Sensur av hovedoppgaver

Universitetet i Sørøst-Norge
Fakultet for teknologi og maritime fag

Prosjektnummer: **2020-07**
For studieåret: **2019/2020**
Emnekode: **SFHO3201-1 19H Bacheloroppgave**

Prosjektnavn
Low Altitude Pathfinding Service

Utført i samarbeid med: Kongsberg Defence & Aerospace

Ekstern veileder: Jan Dyre Bjerknes

Sammendrag:
We have researched whether machine learning can be used as a faster replacement for traditional pathfinding algorithms on very large graphs. We have written a web service which can be used to find paths and get a feel for how well the algorithms perform.

Stikkord:
- Artificial intelligence
- Web development
- Patfinding

Tilgjengelig: JA

Prosjekt deltagere og karakter:

Navn	Karakter
Even Thonhaugen Røraas	
Henrik Thue Strocka	
Håkon Jordet	
Vetle Andre Hoffmeyer Neumann	

Dato: 15. juni 2020

_____	_____	_____
Henning Gundersen	Karoline Moholth	Anders Ronningstad
Intern Veileder	Intern Sensor	Ekstern Sensor

LAPS Group

Bachelor Report: LAPS

KONGSBERG

25th May 2020

Abstract

Pathfinding algorithms find a path from one node to another in a data structure known as a graph. They are heavily used in GPS devices. For GPS devices in cars, a graph is created from the world's road networks, which can then be used to find a path from the user's current location to their destination.

To build an autonomous flying drone, it has to find it's way around the world. Because they do not have to follow a road network, it may be possible to take a direct path. However, this ignores the terrain, and it may naively fly over a mountain, spending a lot of energy climbing to the required altitude. Airborne LIDAR scanning can create very high resolution surface maps. By converting these maps into a graph, we can use a pathfinding algorithm to find the path of least resistance. This allows the drone to fly *around* buildings and hills rather than above them.

However, the problem with this is that these graphs become very large. For roads, they can be heavily simplified, but this is not the case for terrain. This means that pathfinding algorithms become extremely slow. To be as efficient as possible, the paths should be rounded out, which requires post-processing.

The assignment given to us by Kongsberg Defence & Aerospace is to investigate whether a machine-learning based approach could work in this case. The hope is that a machine-learned approach can be run in a fraction of the time, and not require post-processing. To make investigation easier, we had to build a web service which allows us to run and compare the different implementations we came up with.

In this report, we show how the service is built, and what the final result was. We explain the different machine-learning approaches we have experimented with. The chosen process model, using the principles of Kanban, is explained in detail.

In the end, we show that the idea definitely has potential, and that further research in this field is warranted. We were unable to draw any meaningful conclusions as to the viability of this approach.

Contents

List of Figures

List of Listings

Part I.

The Problem

1. Introduction

1.1. Node Graphs

When going by car, one can use a GPS device to find a path to one's destination. When selecting a destination, it will show a map of the area, and show which turns to make at each intersection to get to the destination. This path is typically the shortest which was found.

How does the computer represent and think about the map?

Figure 1.1.: Satellite view on Google Maps

When finding the shortest path, one considers which directions to choose at the intersections. Therefore we need a way to represent the intersections. Intersections are connected to each other with roads. This maps almost perfectly onto a *graph*.

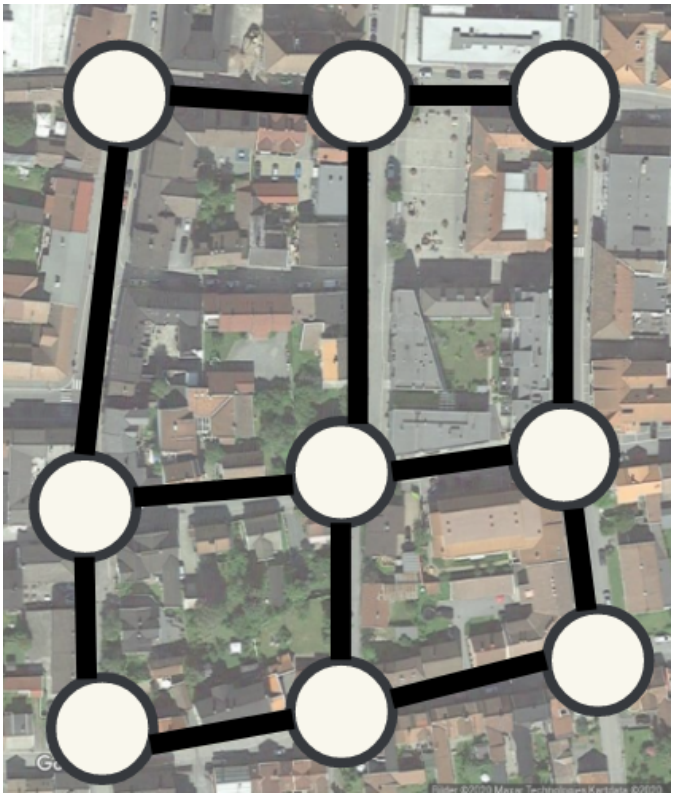


Figure 1.2.: Intersections represented as nodes

A graph consists of nodes and edges. The nodes, sometimes called vertices, are connected to each other by the edges. In the case of converting a map into a graph, intersections map to nodes, and the edges are the roads connecting these intersections.

We see that a path starts at one node and stops at another. The path between them can then be represented as a list of edges to follow. Back to roads and intersections, this tells us which roads to take to get to the ending intersection.

Since we are looking for the shortest path, we need some way to calculate how long a path is. We assign a weight to each edge, creating a *weighted* graph. We can use the weights to calculate the path's cost, by summing the weights of all edges in the path.

By changing how we assign weights, we can get a path with many different characteristics. If an edge indicates a road, then the weight can represent the time it takes to travel across that road. The paths we calculate will be the one which takes the least time. We could also optimise for distance to save fuel, or even combine them.

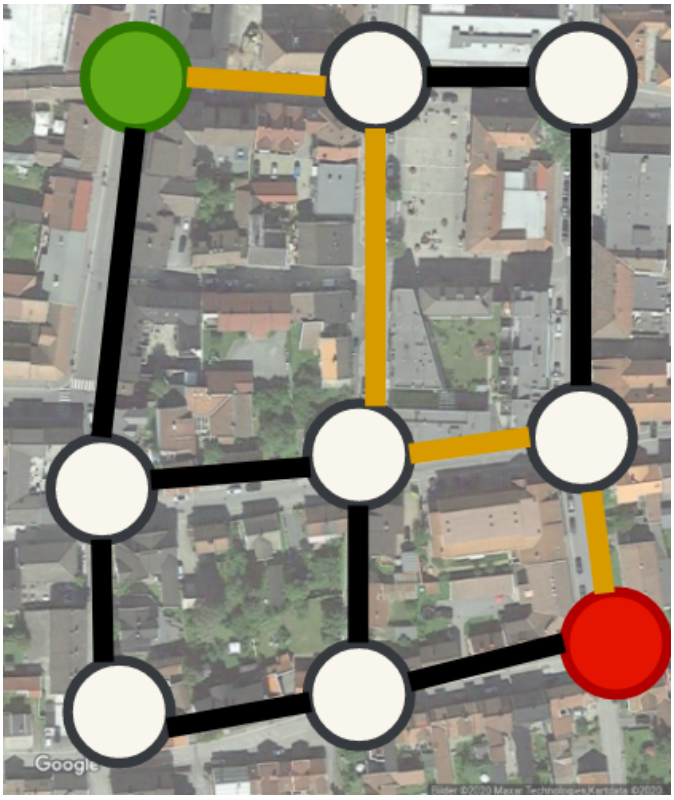


Figure 1.3.: The start and stop nodes, as well as the edges making up the path, colourised.

1.1.1. Node Indexing

Because it is helpful to have a system for referring to specific nodes in the node graph, each node has a unique number. This number is called an index. No two nodes can have the same index.

We can use this idea to represent a specific edge. For this, the notation (a, b) will be used to represent an edge connecting node a and b. For example, $(2, 4)$ would be the edge connecting the node with index 2 and 4.

1.2. Pathfinding Algorithms

Given a node graph, how can one tell if two nodes are connected? We can do it by starting at the first node, and listing all the edges connected to the current node. These edges will indicate the nodes the current node is connected to.

We apply this operation to all the nodes in the list. We stop when there are no new nodes to find, or we have found the node we were looking for. This method of moving through the

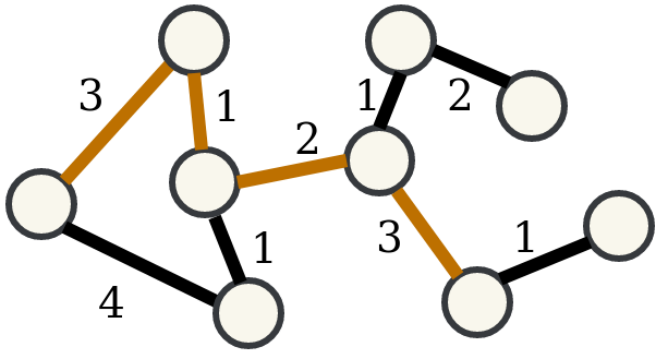


Figure 1.4.: A weighted graph, marked in yellow is a path with 9 cost.

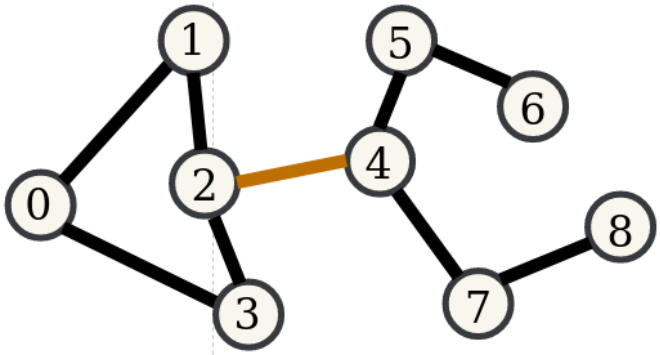


Figure 1.5.: An indexed node graph with nodes 0 through 8, with edge (2, 4) marked.

network by connected nodes is called graph traversal.

We can use a similar approach for finding the path with the lowest cost in a weighted graph. By keeping track of two additional properties of a node while traversing the graph, we can find the cheapest path:

- Cost of the cheapest path from the start node, to this node.
- Which edge that is a part of the cheapest path.

By keeping track of the cheapest cost for getting to a node from the start node, you can use this to find the cheapest cost of getting to the connected nodes.

The latter property is required, because the cheapest path can be followed backwards. By starting at the last node of the path, and following this edge and repeating the process, one ends up at the starting node.

These properties are shown in Figure 1.7.

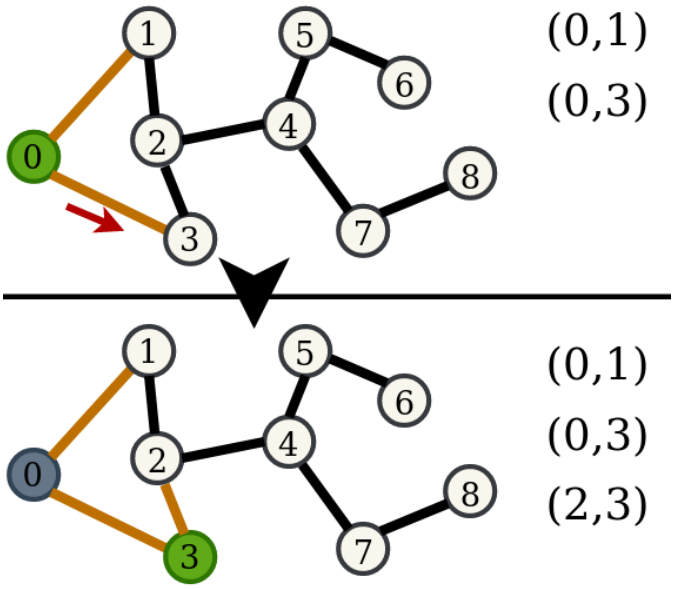


Figure 1.6.: Traversing a graph. List of edges on right side, current node marked with green, and visited nodes grey.

Algorithms which traverse weighted graphs to find the cheapest paths are called pathfinding algorithms. The order in which new nodes are traversed, affect the outcome of the algorithm. This is one place where the algorithms differ from each other.

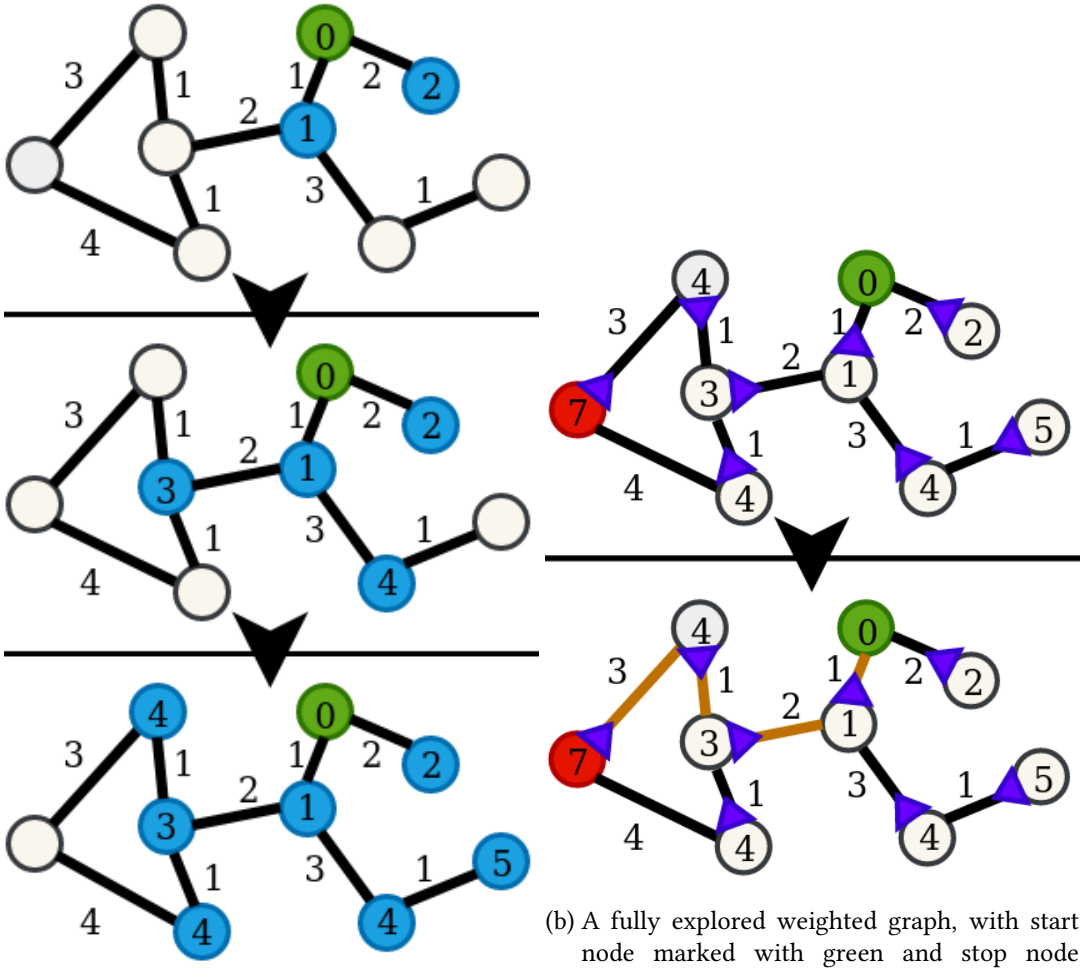
Dijkstra's algorithm is commonly used to find the most efficient path to a node in a graph. However, this comes at the expense of speed, as it has to traverse a large number of nodes.

Other pathfinding algorithms, such as A*, may not always find the most efficient path. A* is a variant of Dijkstra's algorithm which takes the distance between the current node and the destination into account. The result is that the algorithm always tries to move towards the end point. This helps reduce the time taken by trying the more relevant paths first.

1.3. Drone Paths

How can we apply these techniques to a flying drone? These drones are not restricted to following a road network, and can fly above obstacles.

In such a scenario, the drone can be at any point on the map, and move to any adjacent spot. Mapping this to a graph is trivial, each point on the map is one node, connected to the points around it.



(a) Traversing the graph, calculating the cheapest costs of getting to new nodes.

(b) A fully explored weighted graph, with start node marked with green and stop node marked with red. Each node indicating the cheapest cost of getting to that node, and the edge to that path.

Figure 1.7.: Node properties needed to find cheapest path.

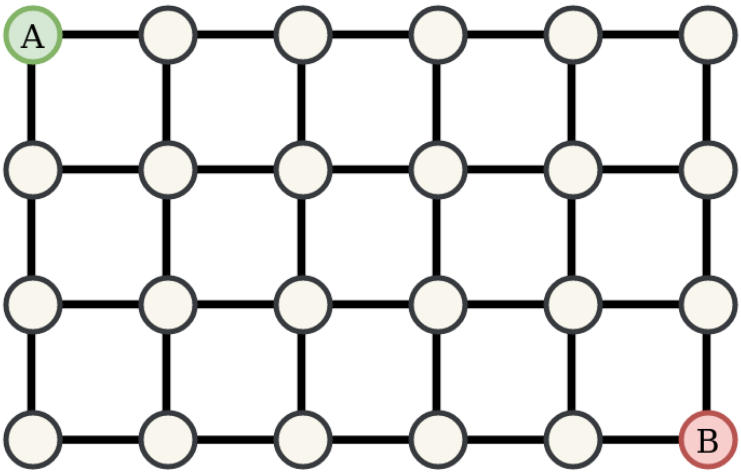


Figure 1.8.: A node graph in the shape of a grid, representing a map with unrestricted movement.

The pathfinding we will be doing in this bachelor assignment, is finding flight paths for drones. The maps to be used are heightmaps of areas in Norway, from the Norwegian Mapping Authority (NMA). Because drones are not restricted in where on a map they can fly, the grid representation of the map as described earlier is necessary.

Cost associated with each edge will be the height at each position in the heightmap. When doing pathfinding on such a weighted graph, the most efficient paths from one point to another will be the paths closest to the ground.

1.3.1. The Flaws

There are some problems with using traditional pathfinding to find flight paths. First of all, because of the grid structure of the node graph, the number of edges will be massive. As shown in Figure 1.9, adding four points to a map increased the edge count by 40%.

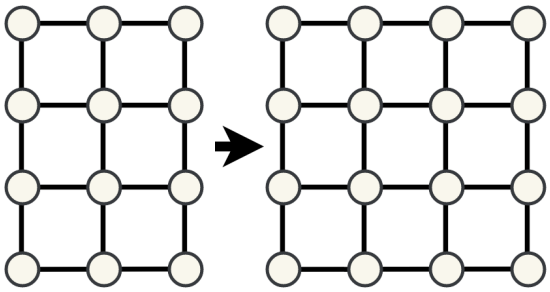


Figure 1.9.: Going from a 4x3 grid to a 4x4 grid adds 7 more edges, for a total of 24.

Because pathfinding algorithms have to traverse the node graph, the more edges there are, the longer it will take. When the amount of edges significantly increases, so does the processing time and processing resources needed to calculate the path.

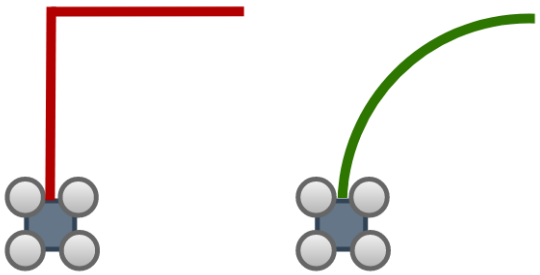


Figure 1.10.: A drone cannot move in right angles like on the left. It has to turn in arcs like on the right.

In addition, with a path in a grid node graph such as this, any turns are sharp 90 degree turns. This is impossible for the drone to achieve while maintaining momentum. This means that the drone either has to move very slowly, or be forced to perform powered braking, wasting energy. For the drone to move at a reasonable pace, turns have to be smooth curves as shown in Figure 1.10.

The way this problem has been solved in the past, is to post-process the path to smooth out the turns as much as possible. This processing adds another layer of calculation which needs to be done.

This smoothing out has at least one major flaw. Even though Dijkstra is able to find the most optimal path in the node graph, smoothing it out means altering the path. This means that we can end up in a situation where the finished path is sub-optimal around obstacles, as shown in Figure 1.11.

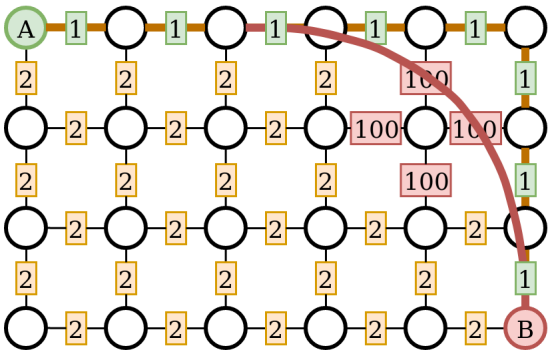


Figure 1.11.: Post-processing on the path making it not optimal any more.

If unlucky, the smoothing out of the turn may have the turn end up in an area with bad cost. And in the worst-case, smoothing out the turn may leave it overlapping the corner of a building

or something similar.

The task assigned to us by Kongsberg Defence & Aerospace, is to investigate machine learning as an alternative approach for calculating these flight paths. The hope is that we can output paths with smooth turns directly, much faster. In addition, they wanted a web service where one can play around with the algorithms and be able to compare our solutions to traditional algorithms.

1.4. Why Machine Learning

In traditional programming, programs are written by writing an explicit set of instructions, which the computer should follow one by one. Any meaningful program will take an input and create some kind of output. There is a problem however, when the task at hand is hard to define. For example, how does one write the instructions for recognising whether an object is a chair or not? Or given an image of handwriting, determine what digit is written?

In cases such as these, an alternative approach is needed. One way to approach this problem is to create an algorithm which can take some input, and adjust itself to «learn» what it should output. These algorithms are called machine learning algorithms. They are given an input which they use to create an output, which is then compared to the expected output, so that it can see what it did wrong and adjust itself to be a little more correct each time.

These algorithms are not only for when the task is hard to define though. In 1952, machine learning was used to play checkers . As computer resources were heavily limited at the time, iterating through the future rounds of the game was not an option. This meant that they had to use machine learning instead. By showing the examples of input where the correct output is known, which is called labelled data, it is able to learn and generalise what to output. In other words, it will learn how to give correct answers for both the data it was shown, but also for new inputs which it hasn't seen before.

Similarly, machine learning solutions have been studied as a method for designing interplanetary trajectories real-time on board spacecraft[1]. As spacecraft processing power is significantly slower than that of modern desktop computers, machine learning models are pre-trained to approximate these functions. Essentially the heavy processing is done beforehand, and the approximations themselves can then be calculated with significantly less processing power, as well as with more consistent calculation times. Therefore, if an approximation for pathfinding algorithms can be achieved with machine learning, their post-training calculation time will be significantly faster, and also more consistent.

A human shown a 3D-heightmap of an area would be able to tell where the lowest flight path should be. Machine learning is able to perform human tasks efficiently, and the thinking is that this could be true in this case as well.

Using machine learning to calculate the path, given the heightmap, would remove the need for iterating through the node graph. As well as if it was trained with the turn radius in mind,

it would no longer need the post-processing. Both these aspects would greatly reduce both processing time and resources needed for calculating a path, while maintaining accuracy. In addition, this would avoid the problem of the curve smoothing altering the most efficient path, which may even mean that it would be more accurate than using traditional pathfinding.

1.5. Service

The customer wanted a web service. It must allow a user to find paths from a start point to an end point easily. The service is broken down into 3 parts as shown in Figure 1.12. A pathfinding module is an implementation of a pathfinding algorithm.

Figure 1.12.: Simplified view of the software architecture

The service seeks to offer a more user friendly experience of the pathfinding modules. The user should be able to use and see the result of pathfinding module without needing to know how it internally works.

All modules should be compatible as long as they follow the LAPS pathfinding module protocol. For the user, this means that different modules will be controlled from the same interface, and removes the need to be familiar with multiple programming languages and interfaces.

This also removes the need the create a new UI for each new module, and as the result is expected to vary from module to module, having a universal UI will be beneficial. This will also allow external modules to be used.

Usability is one of the main reasons to create the service. A web-based solution would allow the service the be used without having to download anything. The service can also be accessed by mobile devices if needed.

The service can be divided into the frontend and backend. The frontend is what runs in the user's browser and interacts with. The frontend should be responsible for collecting user data, and present the results from the pathfinding modules.

The backend should be responsible for the communication between the pathfinding modules and the frontend. The backend runs on a separate server side, and from the users perspective will be interacted with through the frontend.

It should be also be able to run the pathfinding algorithms, store user maps, manage account and other things that might be required.

The backend must validate that the user's inputs are valid. If it is done by the front-end, it would be trivially easy for a malicious entity to crash the service or worse.

1.6. Division of Labour

One can see our assignment as a whole consisting of two assignments. The two tasks do not have much in common, but are connected. As a result, we thought it would be best to split up the group into two main areas: the service and the pathfinding.

As individuals, we have experience with different technologies. Henrik and Vetle have a keen interest in machine learning and experience working with it. The other half weren't as interested in working on the machine learning part of the task. Therefore it made sense to assign them to the machine learning part and the others to the service.

Håkon had earlier experience working on server-side applications and wanted to work on the backend side of the service. Even wanted to work on the frontend. With limited experience in that domain, the project would be a way to get familiar with web development as a whole for him.

In the end, each member of the group ended up working on something they are interested in or already have experience with. This way we all get to work on things we think are interesting.

From there the machine learning crew eventually started investigating different approaches in parallel, helping each other out when they got stuck. The service half ended up splitting the work between the server-side (backend) and the client-side (frontend).

One of the challenges with this approach is that we have to have a concrete plan for gluing everything together. If the entire group was working in close contact with each other, each part would be easier to keep integrated. This meant that we had to define some stable interface between each of the components that we could target. We elaborate on this later in the report.

1.7. The Group

Even T. Røraas
evenrr@gmail.com
Frontend, group leader

Henrik Thue Strocka
henrik.strocka@protonmail.com
Machine learning

Håkon Jordet
haakon.jordet@gmail.com
Backend

Vetle A. H. Neumann
vetle.neumann@protonmail.com
Machine learning

1.8. Other Parties

Internal Examiner

Karoline Moholth McClenaghan

The internal examiner is responsible for arranging the bachelor project. They will be one of three responsible for grading the project. The groups will mostly interact with the internal sensor through the three presentations.

Internal Adviser

Henning Gundersen

The internal adviser's task is to assist the group, mainly in bachelor process. The internal supervisor can also assist academically or help find people who can help the group.

External Examiner

Anders Ronningstad

The examiner that represent the business/customer. Will be one of three responsible for grading the project. The external sensor must be present at all of the groups presentations.

External Adviser

Jan Dyre Bjerknes

The external supervisor represent the business/customer. They are the group's main connection with the customer. The demands of the final product is provided through the external supervisor. It is also their role to make sure the group have the resources needed. Hardware, software or guidance should be provided by the supervisor if needed.

Part II.

The Process

2. Requirement analysis

We need a formal process to define tests, risks and requirements. We went with a pipeline-based approach. We start out with a user story, then gradually break it down.

2.1. User stories

User stories are a paragraph or two which explains some functionality the system has. They come in the form: «As an [actor], I want [feature] because [reason]». User stories can come from multiple sources, primarily directly from the customer. However, the group can also write one based on our own ideas, or based on something we discuss with the customer.

User stories are useful, because they give us a good description of something the system should do. They explain who the feature is for and why they want it. If the reasoning behind a feature is weak, we can reconsider whether it is worth it to implement the functionality at all.

User stories are assigned an incrementing ID prefixed with «U.». This gives us names like U.1 and U.5. The complete list of user stories can be found in Appendix G.

2.2. Features

Features are derived from user stories and contain one specific request the customer wants developed. Multiple features can come from a single user story and a feature can appear in multiple user stories. Once every requirement of a feature is achieved, the function is considered done. Each feature is to have a technical document in which the overall function is explained, as well as listing every requirement belonging to that feature.

User stories are broken down into features because as a pure software project, it makes documenting the requirements orderly. As all the heavily connected requirements and information regarding their implementation are collected in one place, it gives the developer a simple way of getting up to speed on all the relevant information they might need. As well as in the case that a specific requirement depends on other requirements being implemented beforehand, it is an easy place to check the status of these.

We knew from the beginning that several of the features were fairly ambitious. We didn't expect to finish all of our features, which is what ended up happening. We document all our

features in Chapter 10. Note that they are named the same as user stories, but the prefix is «F.» instead of «U.».

2.3. Requirements

Requirements are derived from features and is the smallest division of a feature within the scope of the project. Multiple requirements can come from a feature, and all requirements have to be categorised under a feature. When developing it is the requirements that will be worked on, but will be broken down into subtasks. More about that in Section 3.2.

Features in themselves do not have a priority, but requirements do. Not every requirement of a feature has to be developed sequentially, as it's more important to assess and develop the highest priority of requirements than to focus on fleshing out every feature.

User stories sometimes do not describe a whole feature, but may instead describe aspects of that feature. In this case, the description of the feature will be made into requirements and be put as requirements of the feature it is relevant to.

Because our features are broken down into requirements, each requirement is named after the feature it comes from. This is also why we have chosen to document them as part of the feature documentation in Chapter 10. Like the features, we also knew from the beginning that we likely wouldn't finish them all. For convenience, the full list of requirements can be found in Appendix J, and the full list of features in Appendix K.

2.4. Tests

To verify the system requirements, tests are needed. Ideally, there should be at least one test for every requirement, such that each requirement has defined methods which can be followed to ensure it is met. Each requirement can have multiple tests, and tests can be for multiple requirements. The tests written are to be documented and listed in the Tests table. Tests may become obsolete with time as the system develops.

A test can be an automated software test, or it can be a manual test which we have to do ourselves. Unsurprisingly, an automated test is preferable.

Test verification shall be done on all implemented requirements before they are shown to the customer for validation. The status of each performed test are to be noted, and every failed test must be properly assessed to ensure that the system is working as intended. Every test must be marked whether or not the test is still relevant, and if it has been performed or not, as well as whether the performed test passed or not.

The system tests are not the same as the software tests. Software tests that are implemented ensure that the code does what it is expected of it, whereas the system tests are to verify

the requirements. However, there may be some overlap: some system tests might consist of running software tests.

Tests are tied to our requirements. This means that in theory, we are meeting the requirement if all the associated tests pass. As a result, our tests are documented next to the requirements which they actually test, in Chapter 10.

2.5. Risks

Risks are uncertain events that might happen during the project. If an unforeseen event happens it might cause problems and force the group to deviate from the original plan. The group might lose time or material due to an unforeseen event, resulting in a having a negative impact on the final product. There is no such thing as a risk-free project. Risk management is about detecting potential risk, finding mitigations and creating solutions to mitigate damage. High impact risks should be prioritised before low impact risk, and likely risks will have an higher impact than an unlikely one.

To make sure risks are analysed and mitigations are planned, risks need to be visited regularly. Therefore, during each Functional Analysis meeting, risks related to functionality that is to be added to the system are identified and possible mitigations are discussed. The estimated impact of the risk is also discussed, by determining both its probability of happening and the impact it will have if it happens. New information discovered during development relevant to prior determined risks, is to be brought up so that these risks can be revisited to redetermine probability, impact or mitigation, if needed.

We tie risks to features. We document our risks in Appendix H.

3. The Project

3.1. Process Model

The process model chosen to develop and work on this project was Kanban, which was chosen with consideration to both the project and the team. Because the project involves investigating pathfinding without being able to know whether or not this was a viable solution to the problem, an agile model was chosen to adapt well to change if the approaches we implement don't work.

Kanban, Japanese for "signboard" or "billboard", is a project model with few rules or guidelines. Its core principles are: To visualise the workflow, and pulling work instead of pushing[2]. Developers of the team choose themselves what tasks to work on next, and visualising the workflow ensures that developers have the information they need when making that decision. Pulling work means to only accept more work once a task has been completed, which is a way of reducing work-in-progress meant to reduce lead time on tasks. As work-in-progress is limited and the progress is visualised, tasks stuck in the system are immediately made clear to the team and can be assessed early.

Since there are few guidelines, the execution of the Kanban process is tailored to suit the needs of the team. As workflow is visualised and time goes on, any bottlenecks in the process are made apparent. Together these two aspects encourage the final core principle of Kanban called Kaizen, meaning "continuous improvement". Teams using the model should make changes to the process as they go to remove bottlenecks in the process.

3.2. Task Pipeline

Due to the team and the project being monodisciplinary, there was less need for a multidisciplinary systems engineering approach, which meant we could tailor our Kanban process as a software development process. Work items go through the system one stage at a time, moving to the next stage whenever it is done in the current stage. The stages of the task pipeline are illustrated in Figure 3.1.

3.2.1. Backlog

The backlog consists of features to implement. Before something can be worked on, it needs to be split up into requirements and subtasks. These tasks can then be moved into *To-do*. During replenishment meetings, the team and the customer prioritise which features have the highest priority to implement.

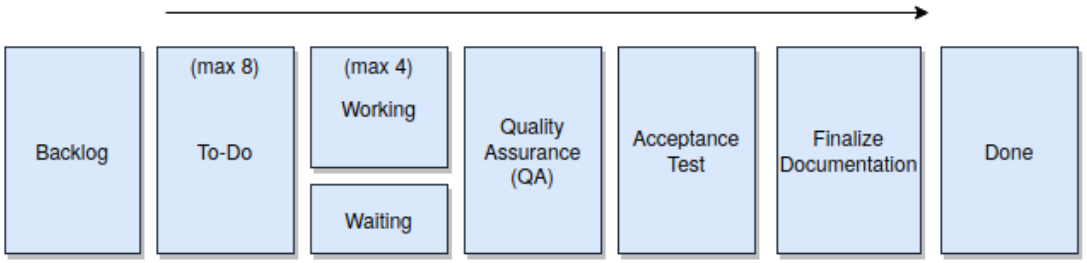


Figure 3.1.: The task process

3.2.2. To-do

To-do are tasks ready to worked on for implementation by a developer, and team members choose which task to work on from this stage once they are done with the task they were working on. Chosen tasks are moved to *Working* and split into one or more subtasks by the person who picked it. Each task inherits the priority of its parent.

3.2.3. Working

Each team member can have a limited number of items in *Working* at once and must judge if it is better to wait for code to be completed before moving the task. This is how we ensure that we follow this core principle of Kanban. The exception to the rule is when a subtask depends on a different subtask being completed. At that point, the developer can move on to a different task. The developer will move the blocked task into the *Waiting* category when this happens. The *Waiting* category should contain as few items as possible. Work should immediately resume on an item when it is unblocked.

3.2.4. Testing

Written code must be tested before the subtask is considered done in this stage. Code testing is done with both unit tests to ensure the code works by itself, and then with integration tests to ensure that the system is working as intended. The testing process will be performed via a Continuous Integration (CI) tool. This will also notify whether any newly added code breaks previously written functionality. In addition to writing the code and tests the subtask must

also be documented in the relevant features technical documentation. This documentation is then used in the code review process.

3.2.5. Quality Assurance

Once a subtask is considered done, before it can be merged with the system, it must go through *Quality Assurance* (QA) where the code undergoes Code Review by at least one other person other than the author of the code. This is to make sure the code is readable, potential defects or bugs can be noticed early, and helps share knowledge. When all subtasks for a task has passed QA they are merged back together and moved to *Finalise Documentation*.

We were initially quite zealous about reviewing each other's code, using our review process in Appendix A. However, thanks to the COVID-19 pandemic, it became harder to communicate and the reviews slowly died out. Because of the way the group is divided, working on different things, none of us really had much code in common in the first place. This made the issue worse, and at one point we had to give in and stop doing it, because they just didn't get done in the end.

3.2.6. Finalise Documentation

Before a task can be considered for *Acceptance Testing* all documentation must be both complete and fit together properly. In this stage the documentation is verified to be consistent and of high enough quality as a whole to be considered done. After this the task will be move to *Acceptance Testing*.

3.2.7. Acceptance testing

Implemented requirements that are merges and tested undergo an *Acceptance Test* during Acceptance Meetings. Here features of the system are verified that they comply with the requirements of the system, and validated with the customer that it is satisfactory and what they intended. Tasks which pass acceptance testing are considered done and will no longer be modified.

3.2.8. Finalisation

After a task has been verified to be completed its technical documentation must also be completed. Here the final parts of the technical documentation is written and the technical documentation as a whole is verified to be accurate.

3.2.9. Visualisation

Visualising the process pipeline and its tasks is done using a Kanban board. A Kanban board is a board divided into columns for each stage of the process pipeline, with an additional column for tasks that are done. Work items are represented by cards on the board that are moved as the task moves through the system. Each card must have enough information on it so that team members can make decisions on which tasks to work on next, this includes the date card was added, which feature it belongs to if it is a requirement, and who is assigned to the task.

3.3. Meetings

Regular meetings are an important part of our model. Without them, our setup would devolve into chaos pretty quickly. In this section we give an overview of every type of meeting we have.

3.3.1. Standup Meetings

Standup meetings held between the team every weekday with core-hours, at 09:15 while standing up. These shall last at most fifteen minutes. At the beginning of the meeting the team shall walk through cards that have been stuck in the system for multiple days, with the exception of backlog or done items, to discuss the cause and identify if it is a problem. After which the team states what they are working on and what they are planning on doing for during the day, stating any issues they are having with their current task if there is any, so that the other team members are aware of the issue and can discuss potential solutions after the meeting.

3.3.2. Replenishment Meetings

Held once a week with the team and customer, in which what tasks on the backlog are of the highest priority will be discussed and subsequently put on top of he backlog. The team shall have suggestions for what is highest priority to present to the customer. To-do is filled up with as many tasks as fit from the top of the backlog after prioritisation. This ensures the development direction of the product will never stray from what the customer wants, and the team is always working on the items of highest priority.

3.3.3. Acceptance Meetings

If any features are completed they will be shown to the customer during Acceptance Meetings which are held once a week. Once the customer has validated that it is what they wanted the feature is considered done. If it fails validation, we have to add new tasks to the system to fix the issues raised.

As both the Replenishment Meetings and Acceptance Meetings are held with the same participants and at the same frequency, they have been combined as two phases of one meeting.

3.3.4. Functional Analysis Meetings

At least once per week the team shall meet for a Functional Analysis meeting. Here, any new User Stories must be processed if there are any. They shall be discussed and analysed to be broken down into Features, Requirements and Tests, as well as Risks shall be evaluated. The team shall also inform the other members of information discovered that may change other requirements, or introduce new risks.

3.3.5. Retrospective Meetings

Held once every other week retrospective meetings is for summarising what has happened since the last, both good and bad. They will be held after the standup meeting at around 9:30 but no later than 12:00. There will be several questions asked to all group members related to the work done in the last two weeks. Summarised the talking points will be What we did right/wrong why it happened and what actions can be done to mitigate/further them.

3.4. Milestones

Because the Kanban process of development is an evolutionary process there should be milestones for the team so they have something to work towards. It also helps you plan ahead. They should be set before any work has begun and be given an estimated time-frame for completion, this way the team can estimate whether they are on or behind schedule. There should neither be too much nor too little development time between milestones. If they are too close they are no longer effective, if they are too far apart the process loses flexibility.

A big risk with the Kanban model chosen for this project is that the team starts falling behind on tasks. There is no way in the Kanban model to see if you are on track to finish the tasks you need. This is why milestones is included in the process model used, their purpose is to serve as guiding for what features and requirements are prioritised. If implemented properly the team will know if they are on track to finish on time and deliver a good product. They are different from sprints, in that multiple milestones can overlap and be worked on at the same time. This combines well with the split nature of our project.

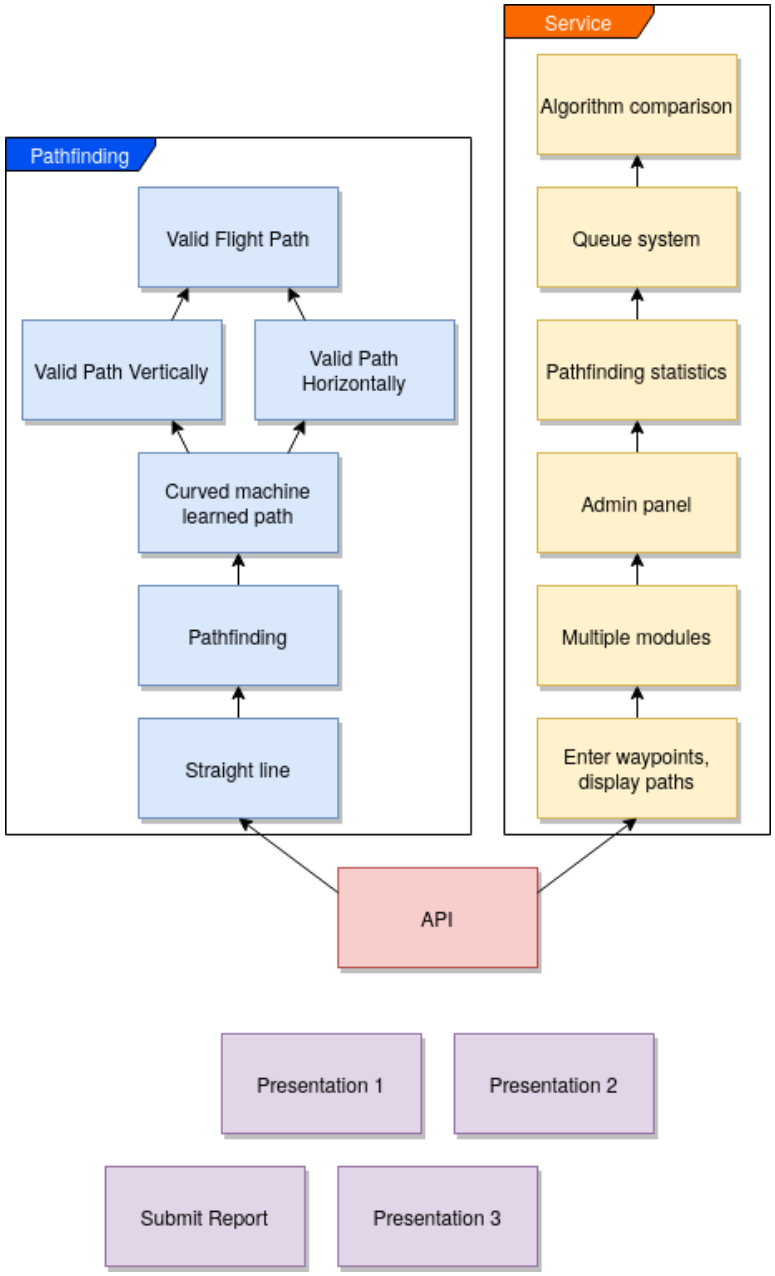


Figure 3.2.: LAPS milestones

3.4.1. Explanations

When coming up with what our milestones should be, we based them around what we wanted to come up in terms of the final product. The following is an explanation of all our milestones as seen in Figure 3.2.

Presentation 1/2/3

Each presentation is a milestone in and of itself as we have to prepare and execute them. They are major events in the timeline.

Submit Report

Perhaps the biggest milestone of all is when we submit this very report. It will be a major relief to all group members, and we are very much looking forward to it.

API

The first milestone is to define an API which the backend and pathfinding modules can communicate through. This has to be stable, and must allow the two subsystems to be developed independently of each other.

Service: Enter waypoints, display paths

This milestone is all about getting a very minimal version of the website working. It entails being able to insert a start and end point, submit this to the pathfinding subsystem, and display the path it returns.

Service: Multiple modules

Central to our service is the idea of comparing different versions of algorithms. Therefore, we need a good way to select between the different versions and algorithms which are available.

Service: Admin Panel

The main idea of the admin panel is to be able to add new modules to the running system. This is an important part of our requirements which we very much intend to add. It will require authentication.

Service: Pathfinding Statistics

In order to figure out whether or not an algorithm is better than a different version, we want to use statistics. The service should keep track of things like average run time, average cost of the paths calculated by the algorithm etc. We can then accumulate data over a long period of time, collecting a bit of data every time a module runs a job.

Service: Queue System

In order to compare algorithms later down the line, we need to have some sort of queuing system. This is because we likely want to run hundreds of iterations of each algorithm in order to get the best possible idea of how it works. The queue system is necessary to not overload our server, if we were to submit every job immediately, we may run out of memory, and throughput will grind to a halt. With a proper queue system it will be easier to share the load across a network of servers, rather than only having one.

Service: Algorithm Comparison

We want to be able to compare the different algorithms. We want to do this with both statistics and by displaying the result side-by-side. If we achieve this milestone, we can say that we have a fairly finished product.

Pathfinding: Straight Line

To know if we have a basic framework that works our first milestone is to create a straight line. This is because it is relatively simple to do and can easily be used to test the frontend as well.

Pathfinding

Pathfinding paths made with Dijkstra, A*, or any other pathfinding algorithm. This is so that we will be able to know the theoretical minimum altitude, and have something to compare our performance with.

Pathfinding: Curved Machine Learned path

In this milestone the machine learning algorithm can produce a path similar to and A* optimal path but it can also curve the path to fit the drones fly radius. This path has to be a flyable path but it does not have to be the optimal path.

Pathfinding: Valid Path Horizontally

Here the path has to be optimal and generated to fit the drones turning radius.

Pathfinding: Valid Path Vertically

The machine learning models are able to generate paths that are theoretically flyable, at least vertically.

Pathfinding: Valid Flight Path

If the path is valid both horizontally and vertically, then we have made quite the achievement on the pathfinding front.

3.5. Tools

We need to use collaboration tools in order to succeed in producing a product. They are essential to good cooperation within the team, which is what drives our success.

3.5.1. Electronic Kanban Board

For our Kanban Board needs, we use Trello. Trello is a general purpose management board program that can be customised to suit any project. It also has functionality like assigning users to cards, categories and more. We customised it to fit our model as described in Section 3.1. We use the assignment feature to keep track of who is doing what.

3.5.2. Git

Version control is an important aspect for any development team, especially since the problem is a purely software problem. We decided early on to use Git for this purpose. Git is a very powerful version control system, and is one we have a lot of experience using, making it the natural choice.

There are many Git services out there which are free to use. A Git service provides repository hosting, and normally an issue tracker and other features. GitHub serves the needs of our project very well, and we use its pull requests to good effect together with Travis CI.

We use Git to keep track of application source code, documentation source code, and our bibliography database.

3.5.3. Communication

For communications between members we use Discord. This is not used as a place to store important files, it is just used as a means to communicate. We also use it to host meetings between the group members.

We communicate with everyone else using e-mail, and hold meetings with people outside the group using Skype.

3.5.4. File Storage

For file storage, we use Google Drive. We use this for everything apart from source code and images etc we need in our report and documentation. We also store the PDFs of all of our documents there when we finish them and have no intention of going back and editing them.

3.5.5. Travis CI

Travis CI is a Continuous Integration tool which will automatically build and test our code for us. We use Travis to ensure that our code will build and passes the tests at all times. Every time we push a commit, Travis will automatically run the tests for us from a clean slate. It is important that it runs from a clean slate, because certain things in our development environment might suddenly be relied upon by our tests without us realising it.

Because the backend is the only part of the system which has tests, it is the only part which gets tested this way.

A branch is a different version of the code base which is separate from the rest. It is tracked independently of the other branches and we can choose which branch to push our code to. Travis treats our staging branch specially.

On this branch, we use continuous deployment (CD) to automatically deploy our code. We use this with the backend and the frontend. The process goes something like this:

- A new commit is pushed to the staging or production branches.
- Travis automatically runs every test.
- If the tests succeed, Travis builds our code in release mode, copies the files over to our server, and finally restarts the service.

This prevents us from doing a manual, error-prone deployment ourselves. It is not possible to do it incorrectly by accident, since the whole process is automated. All we have to do to deploy our website is push the code, and wait for it to complete. If we make a mistake and a test fails, our code will not get deployed.

3.5.6. Code Coverage

When writing automated tests, it is important that the tests cover as much of the code base as possible. To this end, we have setup code coverage tracking in the backend with `codecov.io`. When the tests succeed in CI, we use a tool called *tarpaulin*[1] to run them again. Tarpaulin will then output a report saying which source code lines were tested and which were ignored. This gives us an idea of how well-tested the backend is.

It is easy to get lost in a fruitless process of trying to get the coverage percentage as high as possible. This is mostly a waste of time which doesn't add much value. It is there to show which pieces of code are tested and not. Even with 100% coverage, we are not guaranteed that the application will produce the expected results in every possible case. Therefore we decided to only use the coverage reports to make sure that there aren't large stretches of untested code. By creating a report every time the source code is built, we can strive to keep the coverage high with every commit.

We do not use Test-Driven Development (TDD). While TDD has its benefits, it has a tendency to slow development. This is a problem for a bachelor's project such as ours starting from a blank slate. Furthermore if one has large quantities of tests for every little function the way one is supposed to, refactoring can slow down a lot. Some refactoring requires changes to every test because of an API change. We wanted to avoid this due to lack of time.

[1] `https://github.com/xd009642/tarpaulin`

Part III.

Implementation

4. Technologies

In this chapter we write about the technologies which make up our project, and justify why we chose them.

4.1. PyTorch

A framework developed by Facebook and has begun to see wide adoption within the research community is PyTorch. Among many reasons the one we considered most important was its ability for rapid prototyping and ease of use. To be able to create your own custom machine learning model is as simple as extending a class, and then mixing and matching many of its predefined models together.

This ability to use our own custom models further down the line in a new solution is what makes PyTorch excellent for our project for two reasons. The first is that it allows us to write small parts of our model as separate modules and only modify the part we want to investigate. Say the model contains a convolutional part and a fully connected neural network part, say we want to change our convolution part we can then replace it without having to worry about breaking the fully connected part.

Which brings us to the second and potentially biggest benefit, when you train a model you will get a matrix of weights for each module, therefore the old weights can be used in the new model because the last layer was never changed. This can save huge amounts of computing time by only training the part of the model that has actually been changed.

4.2. Jupyter Notebooks

Jupyter is a web-based method of running Python code. It provides a web interface for accessing the directory it is hosted in, as well accessing Notebook documents, both of which can be done remotely. These documents are divided into cells, which can be either of the type markdown or code. The markdown allows text, has support for text, code blocks, pictures, videos and LaTeX, which provides a way of explaining code in the same place it is written. In the code cells, declared variables are remembered between cells.

4.3. Docker

Docker is a service which allows one to run software in a containerised environment. A container is an environment which is separated from the host machine, and cannot interfere with it. In the case of Docker, a container needs a full installation of an operating system to operate, but it uses the host system's kernel. This essentially means that a Docker container is a lightweight virtual machine.

A Docker image is an immutable base one can create containers from. An image is created from a build file known as a Dockerfile. We can base new images on other images in the Dockerfile. The file itself is just a list of commands to run in the image to produce a new one. We can also perform other operations, such as copying files into the image and more. This means that one can put pretty much anything in an image quite easily.

When one wants to actually run a container, one creates one from an image. This copies the filesystem of the image into the container's directory, which allows multiple containers to be created from the same image. Each container is therefore also independent of both the host and each other. One can also mount shared volumes to share data between containers.

Multiple Containers from the same image may be run at the same time. This allows for easy scaling of servers, as they may be started and stopped depending on demand. Containers are only meant to run one process at a time, and one should instead run multiple of them with one process on each. Docker-Compose is a tool made to make both starting and stopping multiple containers as well as scaling easier. It is configured via a docker-compose .yaml file, in which the configurations regarding how to run and scale the Containers is stated.

4.4. Redis

Redis[1] is an in-memory, key-value database. It has many uses which make it a great choice for our use case. As described in Section 7.3, our architecture is spread out across multiple processes. Because of this, it's a good idea to have a central database which can be reached from every part of the system. Redis is a good choice for Inter-Process Communications(IPC) as well, which makes using it a great option as it can be used for several things at once.

4.4.1. Redis as a general-purpose database

Redis is primarily a key-value database. This means that we can assign values to keys, and index values based on keys. There are several data types which can be assigned to each key. The relevant ones for database usage are the string, hash set, list, set, and sorted set.

[1] `https://redis.io`

Strings

The string type is the simplest. It is simply a binary string which can contain arbitrary bytes, and which can be of any length.

Hash sets

The hash set is almost the same as a string, but it is in itself a key-value structure indexed by a key. In our application, we use a hash set to keep track of what map data is available. Every map has an ID. All map data is stored in a single hash set, and the map ID is used to index it. This is quite useful for us, because the only thing we have to do to see which maps are available, is to list the keys in this hash set.

Lists

The list is a potential building block for using Redis for Inter-process-communications(IPC). It is not possible to directly assign to a list. Instead, one can push elements onto the list, from either the left or the right side. Conversely, one can pop elements off the list from either side. This means that lists can be used as a stack, just by choosing the direction one pushes and pops from.

Sets and sorted sets

A set is a group of unordered, unique elements. As an example, we have to keep track of which pathfinding algorithms are available. By storing this in a set, we can guarantee that the set only contains one of each entry.

There are also sorted sets, which are just like sets, except each element has an associated score, which allows one to sort the set.

Persistence

While Redis is an in-memory database, it is suitable for persistent use case as well, provided your data fits in memory. Redis will automatically dump it's database to disk at a given interval if enough keys have changed. This dump can be read back from disk at a later time.

4.4.2. Redis for Inter-Process Communications

Using Redis lists, or streams, which are just lists with more advanced features, it is possible to use Redis for inter-process communications, or IPC for short. Its high performance makes Redis suitable for this purpose, which will scale to a large number of clients at once. In addition, by connecting Redis instances together, one can easily split the load across multiple machines. Redis also supports channels for passing messages.

Lists have blocking pops which make it easy to sleep until a message is received, but in other cases one might desire look for a message on a set interval instead.

To make lists work for communication, we just have to have one or more consumers push elements onto the list. It is then up to the consumer of that message to pop elements back off the list. Figure 4.1 is a visual representation of how this would work. In this case, the list works as a queue of messages, because we pop from one end and push from the other. If we push and pull from the same side, the list acts as a stack instead.

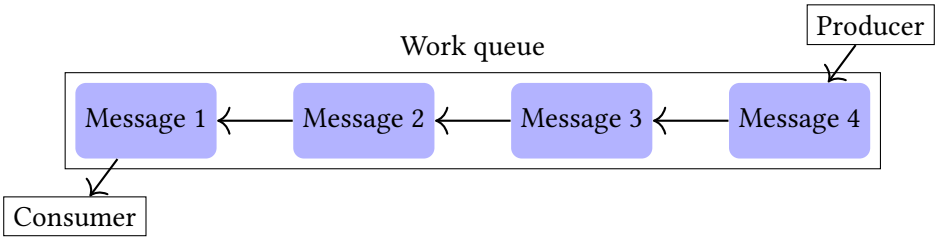


Figure 4.1.: Using Redis lists to communicate

If clients A and B both are blocking and waiting for an element, whichever client started listening first will get the element[3, commands/BLPOP]. This means that we can have any number of consumers when applying the pattern in Figure 4.1.

4.5. Rocket

Rocket[2] is an HTTP framework for Rust. Rocket requires very little boilerplate to work with web-development technologies such as cookies, dynamic routing and more. We want to use asynchronous I/O, so we have to use a development version of Rocket. Rocket has been around for a while, and was written before Rust had built-in support for asynchronous I/O, and it hasn't been fully updated to use the built-in support. We believe that the time we save working with Rocket instead of another framework we are not familiar with will far outweigh the time spent fixing API breakages.

We initially tried using a different framework, which had numerous issues and we didn't find it very nice to work with. There was a breaking point where we decided that it wasn't worth

[2]Link: `https://rocket.rs`

the extra effort, and so we went to Rocket, despite having to use development versions of both Rocket and Rust itself.

4.6. Vue.js

The frontend has to be written in JavaScript, because it has to run in the browser. We want a single-page application, and to that end it is extremely useful to use a proper framework for this. This allows us to use modern techniques to build our application and makes it easier.

To that end, we have chosen Vue.js. It is a JavaScript framework which works using components: Each component has one purpose and can be re-used multiple times. Each component can have it's own internal state which does not change anything else in the application. Most importantly, components allow us to split up our code into smaller, manageable chunks. These component files contain the HTML template to create them, the JavaScript code, and the style sheet to make it look right.

Vue is also reactive. That means that we bind the data and display together. When the data changes, the display is updated automatically. This means that the component code itself has zero relation to what actually appears on the site. It is only concerned with updating data.

4.7. JSON

Java**S**cript **O**bject **N**otation, or *JSON* for short, is a *serialisation format*. A serialisation format is a way to convert a data structure to a string of characters or bytes. The output can later be converted back into the corresponding data structure. This allows one to share data easily across multiple clients.

JSON was originally designed to be used with JavaScript[4], but has become one of the most common serialisation formats around. It allows one to serialise complex data structures. It's easy to read and understand for humans, which is very helpful during development and debugging.

Because of this, JSON is the only serialisation format used in LAPS. There is only one exception. JSON does not support comments, and isn't the best fit for configuration files. Therefore, we use TOML[3] for backend configuration files. See Listings 1 and 2 as examples and comparison between the two formats.

[3] url: `https://github.com/toml-lang/toml`

```
{
  "redis": {
    "address": "127.0.0.1:6379",
    "password": "some-password"
  }
}
```

Listing 1: A snippet from the backend configuration serialised as JSON. Note how JSON does not support comments.

```
[redis]
# The Redis server to use.
address = "127.0.0.1:6379"
password = "some-password"
```

Listing 2: The same data as in Listing 1 but in TOML format instead.

5. Algorithms

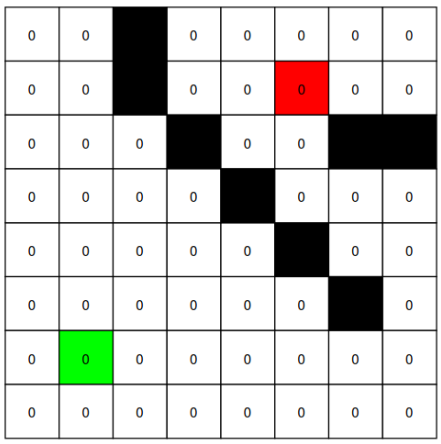
5.1. Dijkstra

The classic way to find a path between A and B with the lowest cost is by using a pathfinding algorithm, and there are two major pathfinding algorithms. A* and Dijkstra where A* is a modified version of dijkstra. This project will be using the dijkstra algorithm because it has the potential of being modified more easily, this is explored more in chapter Section 6.7.1.

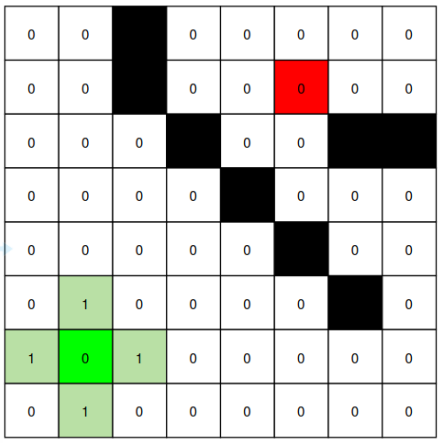
When the dijkstra algorithm start it creates two sets of nodes, one called open-set contains all nodes on the grid except the starting node. Then you have the closed-set, this is all the nodes the algorithm has previously visited. This set is empty at the beginning. The last initialisation step Dijkstra does is to set the start node as the current node it is investigating. Figure 5.1a shows the state at the beginning of the algorithm.

When it is done with the initialisation step the algorithm will start by going through all the neighbouring nodes that are in the open set and calculate their cost. This is done by taking the cost of the current node and adding that to the travel cost from itself to the neighbour node. Figure 5.1b shows Dijkstra after it has calculated the cost value of the first set of neighbours shown in light green.

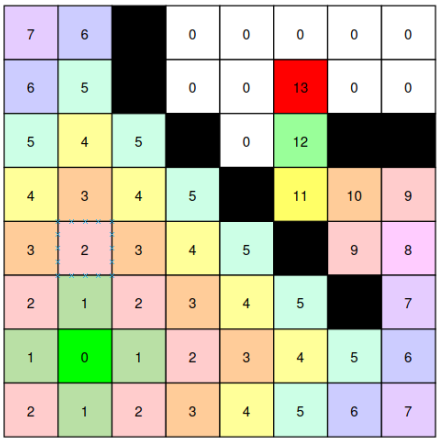
This step is then repeated until the current node is the stop node. When selecting the next current node djikstra will always choose the node in the open set with the least cost. The grid with all the cost values for each node can be seen in Figure 5.2a. Then to get the actual path it is only a matter of tracing the lowest cost neighbour node for each node starting at the end, as shown in Figure 5.2b.



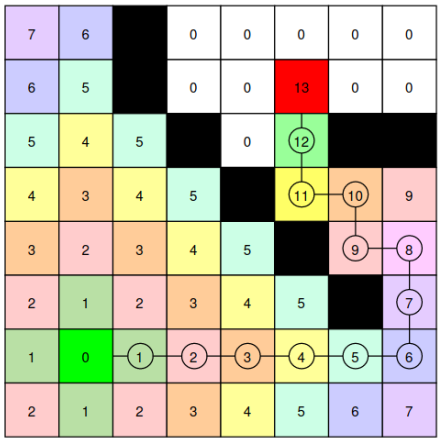
(a) First stage of Dijkstra's algorithm. Green is the start node and red is the end node.



(b) Second stage of Dijkstra's algorithm when it has investigated the neighbours of the start node.



(a) Dijkstra finished with calculating a path with each cost value has a separate color.



(b) Dijkstra finished with calculating all needed cost values, the optimal path is highlighted with black circles.

5.2. Machine Learning

The task itself requires the use a machine learning algorithm to approximate a path with the least height difference from A to B. But machine learning algorithms are a broad field with a multitude of different approaches and solutions for all kinds of problems.

5.2.1. Neural Networks

Currently the most popular machine learning architecture is neural networks. At its core a Neural Network is a $\mathbb{R}^n \longrightarrow \mathbb{R}^m$ dimensional function that is organised into layers of nodes as seen in Equation (5.1). The first layer is called the input layer and is the function input parameters. The last layer is called the output layer and represents the values that the network will output. The layers in-between are called hidden layers and are only there to give the network more weights to tune.

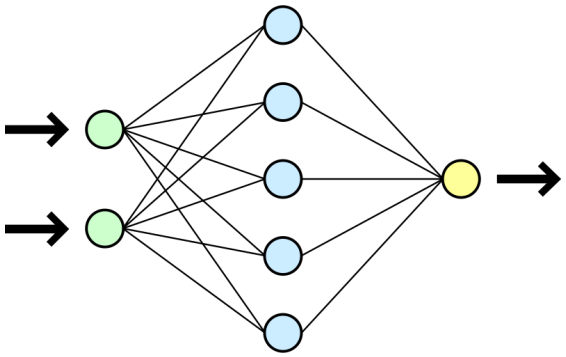


Figure 5.3.: Example Neural Network Structure [5]

The weights of a neuron are an individual scalar for each connection the neuron has. Each neuron is connected to all neurons in its previous layer, as represented by a black line in Figure 5.3. So the value of each neuron in a neural network is a weighted sum of all nodes in its previous layer as shown in Equation (5.1). In addition to this the output of the weighted sum is usually passed trough a function called a activation function. This is done for a multitude of reasons but the most important is to stop the values of the network from skyrocketing. Equation (5.2) is a common activation function called Sigmoid.

$$\sum_{i=0}^{n} w_i \cdot n_i \tag{5.1}$$

$$f(x) = \frac{1}{e + e^{-x}} \tag{5.2}$$

Forward Propagation

Forward propagation is a term given to the action of passing some input values through a neural network. As the values of each layer are calculated sequentially, the data propagates forwards. This term leads into how a network 'learns' by using an algorithm called backpropagation.

5.2.2. Backpropagation

The key to how a neural network approximates a function is by adjusting each weight for each connection in the network. It does this is in the form of an algorithm called backpropagation, if the name does not give it away it works in the reverse order from forward propagation. By starting at the output nodes and calculating the error for each.

An error function is a function that describes how wrong an output is. For a function to be usable as an error function it has to be continuous and differentiable. This means you can extract from the error function how to change the output to minimise it, making the output more 'right'.

To change the output of each node without controlling the inputs, you adjust the incoming weights for each input. To decide how much you should adjust each weight you can use the chain rule to figure out how 'important' each input in the sum is for the final output of the node. The process is repeated for each node in each layer, starting at the final layer moving backwards, hence the name.

5.2.3. Convolutional Neural Networks

A problem neural networks ran into early with image classification was the problem of extracting locally specific features. It was easy to flatten an image into a list of colour values, but in this format the network was very bad at extracting features from an image and using those to classify it.

To solve this problem neural networks can employ special convolution layers to do the feature extraction. What convolution layers to is to reduce the size of the feature space by moving an $N \times M$ filter matrix of weights across the image. So each new pixel will be a sum of the dot-product of all pixels from that filter, as described in Equation (5.3).

$$\sum_{i=0}^{N} w_i \cdot x_i \tag{5.3}$$

The example shown in Figure 5.4 shows a 4×4 image being convoluted by a 3×3 filter. Here the output would be a 2x2 image also called a feature map. The reason this can extract features is because usually pixels next to each other is part of a local feature, and these features are usually part of a bigger feature. For example how a finger is a part of a hand, a hand a part

of an arm etc. So by stacking the convolution layers they are able to extract more and more complex features from a given image.

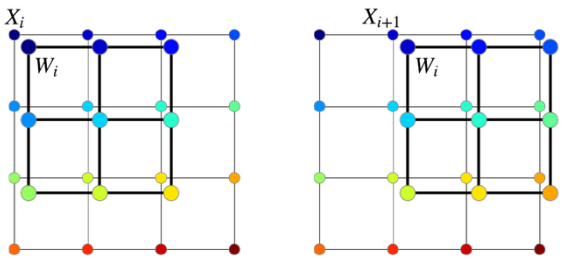


Figure 5.4.: Convolution filter example on a 4x4 image.[6]

These layers are trained much the same way a normal neural network layer are trained via backpropagation, but now the weights are inside a filter and not directly connected to the previous layer.

5.2.4. Reinforcement Learning

Reinforcement learning problems consists of two parts, an environment and an agent. The environment is the problem domain, consisting of states with information describing the current state of the domain, and actions which affects the state and moves the environment into a new state. An example of an environment is Pong, where the states are the position of the paddles and the ball, and the states are either moving the paddle up or down.

The other part of a reinforcement learning problem is the agent, which uses a policy for picking actions depending on what the current state is. Upon choosing an action the agent is given feedback - called the reward - to tell it how good the selected action was so that it may adjust its policy. The agents job is to adjust its policy to pick the actions that maximise total reward. 5.5 illustrates the relation between the agent and environment.

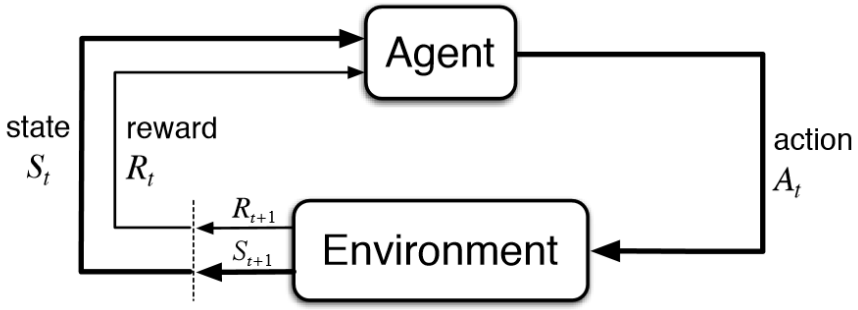


Figure 5.5.: Reinforcement Learning environment and agent[7]

To maximise total reward the agent has to decide how it will weight immediate rewards In comparison to long-term reward. Though immediate rewards are good, if an action gives lower

reward now but results in significant reward in ten steps, that action would be better. This also leads to exploitation vs exploration. When the agent is learning, if it only performs actions which give the highest reward it may be missing actions which would lead to higher rewards. Therefore the agent has to make a balance in its policy between exploitation and exploration of actions[7].

5.2.5. Graph Neural Networks

Graph neural networks are an emerging subfield within the broader machine learning field. Graph Neural Networks differ in the way they operate on data. Normally a neural network operate on euclidean data, but it is hard to represent some data in this format. Often it makes more sense to structure data in the form of non-euclidean graphs.

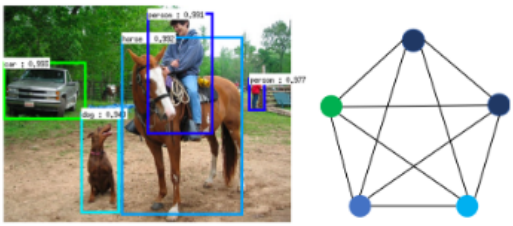


Figure 5.6.: Example of object detection data in euclidean and non-euclidean space.[8]

This has some advantages, mainly that it is often easier to model data and data relations in terms of graphs. Even for some data we normally does not see as graphs it can make sense to structure them as graphs, for example images where each node is a pixel with connections to each node besides it.

GNNs originates from image processing with convolution, more specifically its ability to extract spatial localised features. GNNs aim to be able to do the same but more generally on all graph structures.The way a normal GNN work is by using feature information about a node and its neighbours to estimate a D dimensional vector that represents the information of that neighbourhood using the following formula:

$$h_v = f(x_v, x_{co[v]}, h_{ne[v]}, x_{ne[v]})$$

Where $x_{co[v]}$ is the feature of the connection with node v. $h_{ne[v]}$ is the D dimensional vector that represents node v. $x_{ne[v]}$ is the features of neighbour node v. Here f can be seen as a standard neural network that can be optimised.

6. Pathfinding

When approaching a machine learning problem, the first step is to identify what inputs the model will have, and what outputs would be achievable. At its core, machine learning is just approximation if functions. The tasks machine learning excels at are tasks that humans perform well. This is important when deciding on outputs.

Once we have an idea of what the inputs and outputs should be, the category and type of machine learning should be decided. There are three primary categories: supervised-, unsupervised- and reinforcement-learning. The categories are not hard boundaries and there are types which are hybrids between these categories. The category determines the way the model learns.

To investigate pathfinding machine learning solutions on heightmaps, a cost function has to be designed. The best path to generate is the path with the least cost. Because of the limited time on the project and to mitigate for RI.3.3[1], it was important that we focused on the problem with the as little complexity as possible. Therefore we initially chose the simplest cost regarding heightmaps: altitude at each node. The cheapest path is the path with the least changes in altitude across it. Similarly, the outputs we focused on would need to be the ones with lowest complexity, while still giving us useful insight to the problem.

6.1. Development Environment

The chosen tool for developing the pathfinding solutions was Jupyter Notebooks. Hosting a Notebook Server allows us to access the Jupyter Notebook remotely, which has the advantage of allowing development on a different computer than the one running the code. This is critical, because this way we can use very powerful servers to train the models. Training takes a lot of computing time, and reducing the training time allows us to try out more models in less time. Each person developing on the pathfinding was given access to their own Jupyter Notebook on a training server. This was to mitigate RI.3.8, which states that working in the same directory on the same computer may cause conflicts.

For hosting the development environment, it was decided to use Docker[2] as this would give us a consistent environment in which we could develop and run the pathfinding modules.

If we end up with a machine that is not powerful enough as per RI.1.1, Docker allows us to move the development environment. We could very easily deploy the same Docker image from the

[1]The model does not converge on a solution to the problem.
[2]See Section 4.3

old machine to the new one. Docker is widely used in software deployment exactly because of this, and understanding it would be beneficial if we decided on using it in that use case as well.

A reverse proxy places multiple HTTP servers behind one port, and will redirect traffic to the right server based on the URL or a header. This allowed us to have different notebooks on different subdomains. This helped keep them separate such that the machine learning developers did not interfere with each other.

6.1.1. USN Machine Learning Server

For the first part of the project we did not have access to a powerful server for machine learning. This limited us quite a bit up until we were given access to a server at USN. We did not have it set up and running until after the second presentation. Before then, we used Docker containers to set up a development environment on a group member's gaming computer. Moving to the new server was a cinch thanks to the containers.

Then when the server had been configured properly some time was spent configuring our models and training data to work on the GPU. After everything was configured, the models could be trained approximately 100 times faster.

The servers had four GPUs. After a while, it became apparent that memory management on the GPUs was a problem. When both machine learning developers were training a model at the same time, the memory would often fill up. This caused the programs to either crash or slow down significantly. To solve this problem, we allocated two GPUs to each developer.

6.2. PyTorch

This ability to use our own custom models further down the line in a new solution is what makes PyTorch excellent for our project for two reasons. The first is that it allows us to write small parts of our model as separate modules and only modify the part we want to investigate. Say the model contains a convolutional part and a fully connected neural network part, say we want to change our convolutional part we can then replace it without having to worry about breaking the fully connected part.

Which brings us to the second and potentially biggest benefit, when you train a model you will get a matrix of weights for each module, therefore the old weights can be used in the new model because the last layer was never changed. This can save huge amounts of computing time by only training the part of the model that has actually been changed.

6.3. Problem Analysis

When investigating a problem, it is often smart to start off with a simplified version of the problem and tackle it one challenge at a time. Our task of developing a machine learning based solution for pathfinding is a complex task. We can break it down into many smaller parts which allow us to lay the groundwork, one step at a time. As our time on this project is limited, it is important to have some simpler goals, in case the more complex goals are hard to achieve.

The output of a neural network will always be a list of numbers. It is how you interpret these numbers that enable the bridge between the output and the path.

Currently the problem is quite complex with many parts that add complexity and are candidates to be simplified. The first simplification that was done was to change the environment from 3D to 2D with the value at each point being the height of the ground. This means that the algorithm only has to deal with two dimensions and each point has a defined value.

Further these values can be used to define a good cost function. Here, we can create quite a complex function can take a large amount of information into account. In the interest of keeping the problem simple we chose to use the total height of all the nodes in the path as the cost. This mean that the optimal path in this simplified case is the path with the lowest total height.

Next was the problem of turn radius, in the real world few vehicles has a perfect turn radius of 90 degrees so this has to be taken into consideration in the final product. However for a simplified problem assuming a perfect 90 degree turn radius is okay. Together all these simplifications allow us to use traditional pathfinding algorithms to generate a “optimal” path within the constraints of the simplifications.

Because of this “optimal” path that is now available, it can be used to verify and guide the machine learning algorithms. With the hopes that once the machine learning results are accurate enough the simplifications can be removed on layer at a time.

Before an implementation can be made there is one more aspect to decide: How will the machine learning algorithm output a path that is of any use to the user? There are many ways to do this and each implementation will discuss how it is done in that particular case, but they all lie on a spectrum depending on how much processing has to be done before the path is usable.

On one hand there are the machine learning algorithms that directly outputs a path that is usable without any modification to it. On the other there are algorithms that require intense processing before the path is usable. Algorithms that produce a path immediately may be harder to train because some of the work is offloaded to the post-processing stage.

The last thing to take into consideration for a machine learning based solution is that it is inherently an approximation of the problem domain. So even if the algorithm outputs directly usable path it is still only an approximation.

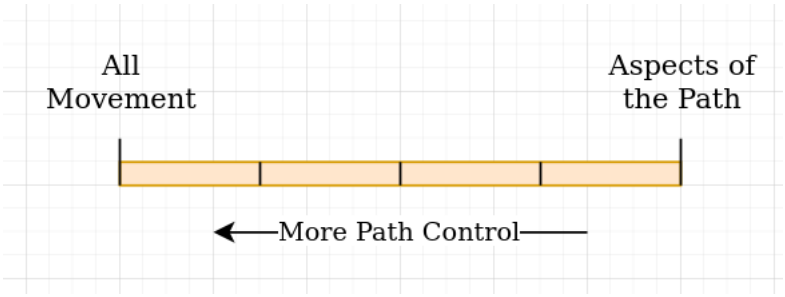


Figure 6.1.: Machine learning output control over path.

6.4. Convolutional Neural Networks

As specified in Section 5.2.3 one of the main benefits of CNN's is their ability extract local features from an image. In many ways similar to how a human could look at an image of an object and immediately know what the object is. This property emerges in a neural network when convolution is used. The hope was to utilise this property when generating the path from an image.

6.4.1. DeepStar Structure

Convolution Layers

As described in Section 5.2.1 any input for a neural network is simply a series of numbers. This is no different for convolutional neural network, the numbers are just expressed a bit differently. Instead of being a series of input numbers the image is expressed as a matrix vectors. So each pixel get a cell in the matrix, and each cell contains a vector with one dimension for each colour channel used.

As the image is run through the convolution layers of it is converted to a feature by the filters described in Section 5.2.3. This feature map is on the same matrix format as the input image and is an representation of the features the convolution layers has learned. However these convolution layers are not able to make the predictions that we need.

Classification Layers

This is where a normal neural network as explained in Section 5.2.1 is used. This network is put in between the output of the convolution layers and the output of DeepStar, and is responsible for taking the feature map and using it to make a prediction. In this case it has to predict the path between the start and stop point in the input image, Figure 6.4 shows the classification layer of DeepStar.

Flattening

However to do this the matrix formatted output of the convolution layers has to be converted to a series of numbers that the network can read. This is done by going trough every cell in the matrix and stacking every vector on top of each other. So by the end you have a series of numbers where every set of three values represent one cell or pixel in an image. This process is called flattening.

Loss function

As explained in Section 5.2.2 backpropagation is the way a neural network can train and learn how to approximate a solution. An an essential part in this process is to determine how wrong an output is. This is done by a loss function, or sometimes called an error function.

How this is done varies widely depending on the use case, but for DeepStar a simple loss function called MSELoss can be used. This loss function measures the mean squared error for output as explained by Equation (6.1) where x is the actual output of the network and y is the expected output of the network. It is this loss function output that is the accuracy of a network.

$$\sum_{n=0}^{N}(x_n - y_n)^2 \tag{6.1}$$

Normally the accuracy of a network is hard to conceptualise, this is because by definition the accuracy is just a number assigned by the loss function that describes how wrong the output of a network is. As the complexity of a loss function increases so does the understandability of the resulting accuracy of a network.

This however is usually not a problem because as long as the same loss function is used a lower score is always better. So an loss of 0.15 is always better than a loss of 0.2 unless they use a different loss function. This means that unless two networks use the same loss function their accuracy values cannot be compared.

Pooling

6.4.2. Version 1

To keep the input simple for the first iteration a three channel (RGB) input image was used. The red channel represented the heightmap, and green and blue channels represented the start and stop point respectively. Because no proper heightdata was available yet it was substituted with a simplex noise function instead. See Figure 6.2 for an example.

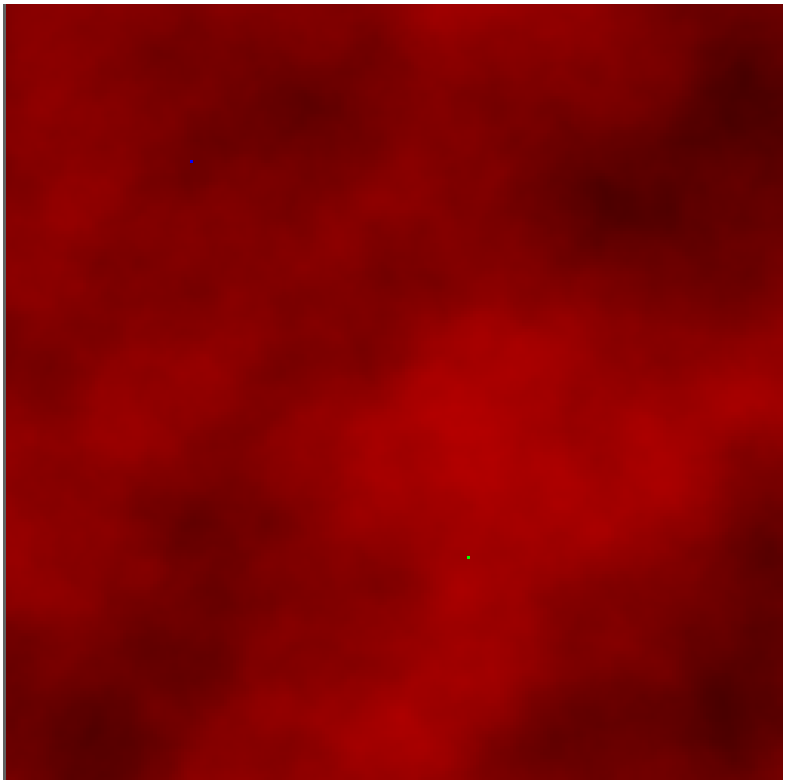


Figure 6.2.: Example noise map input data. The red channel contains the noise, and its hard to see but there is a green and a blue dot for the start and stop point.

A noise function is similar to a random number generator but instead of producing truly random numbers it produces random numbers that are dependent on its neighbours. This means that the noise map substitution will have a more smooth transition between values than if a random number generator was used.

Output

Next the format of the output had to be determined. Keeping it simple for the first version it was decided to have the network output two values. Each value would be interpreted as X and Y respectively and be within the range $[0, 1]$. Meaning an output of $(0, 0)$ being top left and $(1, 1)$ bottom right of the image. This kept the output simple and independent of image size.

Combining the two outputs of the network it would only be able to output a single point on the image. It was decided this point would be the median point of the dijkstra path between the start and stop point. The idea being that this network could be run recursively to produce a more fine grained path.

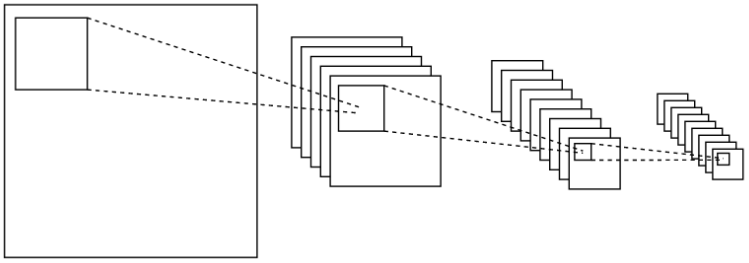


Figure 6.3.: Convolution layers of deepstar v1

Network Structure

When the input and output for this version had been defined an internal structure that would translate the input to the output had to be defined. This is where the less scientific nature of machine learning rears its ugly head. Because how do you go about creating this structure, you know there will be at least a convolution part and a classification part. Bu other than that it is mostly guesswork and gut feeling that guides this process.

Keeping with the ethos of keeping it simple the first attempts had two convolutional layers followed by a two layer deep classification neural network. As many first attempts do it did not work very well, but after a lot of trial and error the structure shown in Figure 6.3 and Figure 6.4 was arrived at.

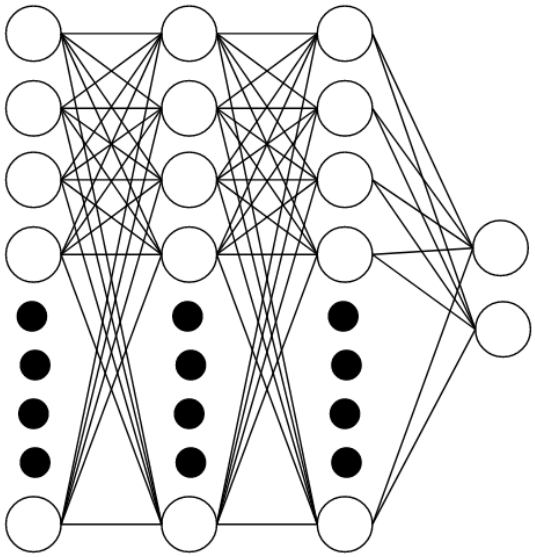


Figure 6.4.: Classification layers of deepstar v1

Results

It was able to achieve an average accuracy of around 16 pixels. At first glance this seems like a really good result. It turned out that when applying the algorithm recursively the error kept accumulating. The end result was a path that is not much of use. However this result did prove that the network was converging on something.

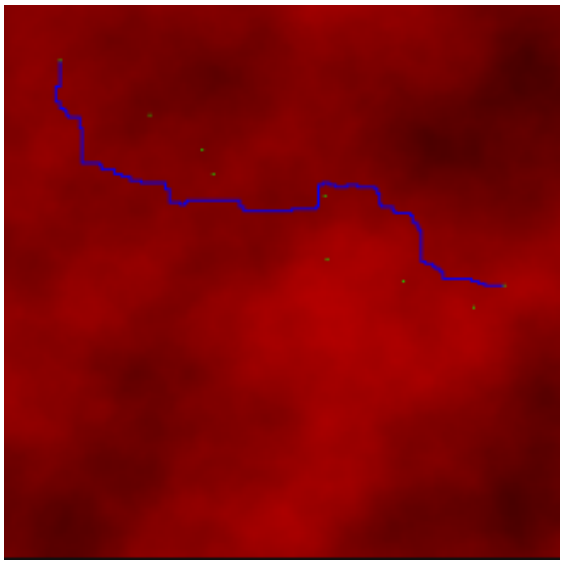


Figure 6.5.: Dijkstra optimal path overlaid in blue. Green dots are where the network predicts the path to go.

As stated in Section 6.4.1 normally the accuracy of a network is hard to conceptualise. However because of the way this network has been structured it has a quite elegant interpretation.

Because of the MSE loss function used the accuracy is defined as the mean squared error of the two outputs. Therefore because it can be defined as $x_e^2 + y_e^2$ where x and y is the two outputs. This can then further be put into the circle formula which gives.

$$R^2 = x_e^2 + y_e^2 \tag{6.2}$$

This results in the accuracy being the squared distance between the outputted point and the expected point.

6.4.3. Version 2

Because machine learning is more of an art than a science, gut feeling is on of the most important prats of deciding how to move forward. In the case of DeepStar there was also the fact

that the schools computing server was still unavailable. So for the second version the goal was to try and reduce the computing overhead and maintain or increase the network accuracy.

By version two some heightmap data had been created and from here on DeepStar would use actual height data from the Kongsberg region. This was done by taking a 1km by 1km slice of height data, the slicing that image again into many smaller $256x265$ images to use as training data.

Input

The first and most obvious optimisation that could be done was to reduce the amount of colour channels in the input image. In version one the G and B channel where mostly empty except for a single pixel at the start and stop point. This would mean the network wold take a single channel heightmap as an input in addition to two points for the start and stop point.

Structure

This was achieved by saving the height map as a single channel greyscale image, and then feeding that into the convolution layers. The convolution layers was kept the same except they took one channel as input instead of three.

The start and stop point was instead fed directly into the classification layer as shown in Figure 6.8. Four inputs was added on top of the flattened feature map input, these four inputs would represent the X and Y for the start and stop point. The input would be normalised to UV coordinates in the range $[0, 1]$, same as the output.

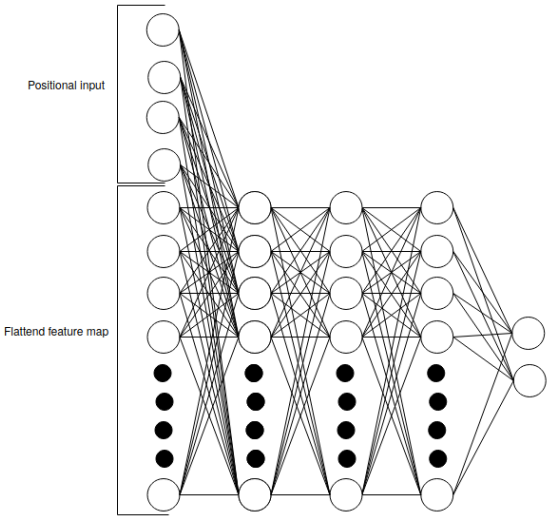


Figure 6.6.: Classification layers of deepstar v2

Results

There are many ways to tweak a neural network and it is not immediately obvious how those changes will affect the network. There is a umbrella term for all those parameters you can change on a network, they are called hyper-parameters. Examples of these hyper parameters is all the way from the learning rate used by the optimisation algorithm, to the structure of the network.

It is these hyper-parameters that are tweaked to see how well a version preforms. There is no guarantee we have stumbled upon the most optimal values, but because of the finite time at some point you have to cut your losses av move on.

After much tweaking with the hyper parameters the accuracy of the network never went above the previous seven pixels. So in the end, the second attempt ended up being more of a sidestep in terms of performance than a forward step. However, we decided to keep this data structure for future versions because it fulfilled the primary requirement of reducing the size and computing overhead of the network.

6.4.4. Version 3

Now that the network was computationally less expensive and would train quicker it was time to focus on how to improve its accuracy. To avoid going in blind and just trying random things, it was decided to lend some tricks from other fields within machine learning. Specifically face keypoint detection[9].

In face keypoint detection the aim of the network is to detect key-points on any given face. These points can the be used further to determine features about the face or classify it. An example of face keypoint detection can be seen in Figure 6.7.

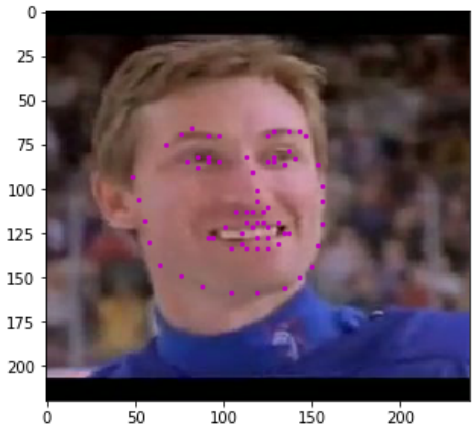


Figure 6.7.: Example of face key point detection[10].

Output

More specific the idea is to borrow the way it outputs its keypoints. It is done by splitting the image into chunks. So if the image is 256x256 pixels it could be split into a 64x64 chunks with each chunk being 4x4 pixels. Then each chunk is given an individual output that represents that chunk.

The output of the network is now 128 unique values where each value represents a unique label. The output went from two output nodes as seen in Figure 6.4 to 128 output nodes. The first 64 outputs belong to the first axis and the last 64 outputs belong to the second axis. Here each row and each column is given a unique output called a label and by selecting one of each the network can specify a chunk.

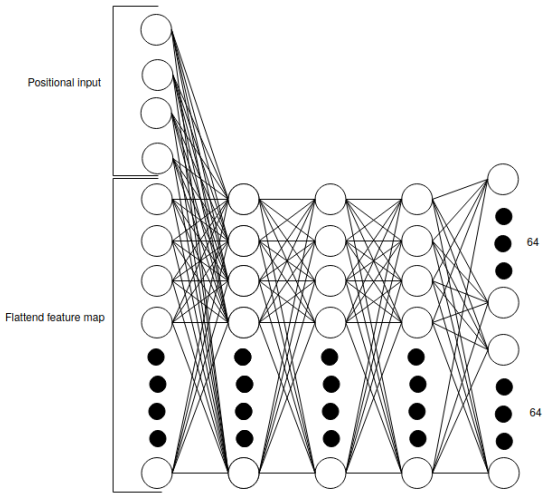


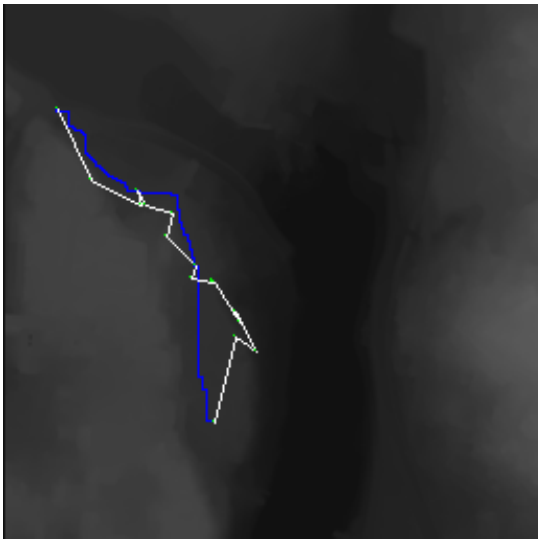
Figure 6.8.: Classification layers of deepstar v3

Results

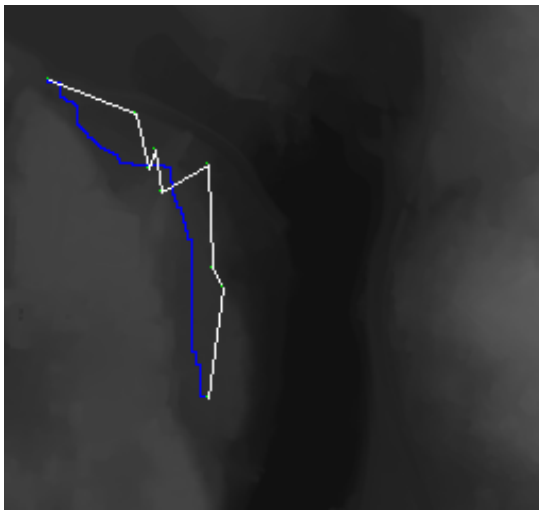
The rest of the network structure and input was kept the same, including the UV normalised input positions. After some days of trial and error with tuning the networks hyper-parameters it was able to achieve a good accuracy.

This is where a problem arose, because now even though the loss function was not changed the output was completely different. This mean that the Equation (6.2) was no longer fulfilled. This mean that there was no scientific way to compare version 2 and 3, so to judge this it was a matter of maximising the networks accuracy and visually compare the resulting paths as seen in Section 6.4.4.

As seen here the paths seems to have about the same accuracy but it is hard to tell. At the very least it does provide any significant improvement over the last solution.



(a) DeepStar V2 Output



(b) DeepStar V3 output.

6.4.5. Auto Encoder

Auto Encoders are a special type of machine learning algorithms. As seen in Figure 6.10 they take the input data and compress it down into a smaller vector. How small this vector are is dependent on how the network is setup. After the data has been compressed it is run through the reverse process, where a separate network takes that compressed data and tries to recreate the original data.

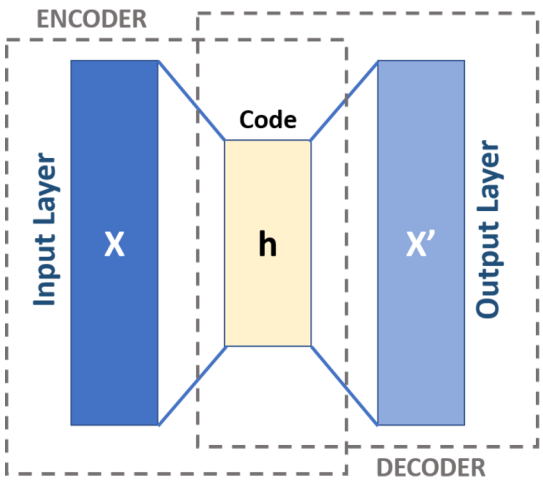


Figure 6.10.: Illustration of an auto-encoder[11].

This is useful for us because if we use convolution layers the compressed vector that is used to recreate the image becomes the same feature map that is used by the previous versions

of DeepStar. The idea is that after the auto encoder has been trained the decoding part that produces the feature map can be taken from the auto encoder and replace the convolution layers of DeepStar

This concept has a name and is called transfer learning. Because now we are separately training the convolution layers of deepstar and the classification layers. As long as the parameters of the convolution layers does not change it does not have to be retrained and the training process can be sped up.

Results

After about a day of hyper-parameter tuning for the auto encoder it was able to reproduce the images with a pretty good accuracy as seen in . It was now time to insert it into DeepStar and see if it will increase the accuracy. As you can see in the comparison in the it did not make a significant improvement.

6.5. Graph Neural Network

After trying many different ways to improve the performance of the CNN, we felt it was time to investigate something a bit different. During our research on improvements we discovered a new type of neural network called graph neural network. As the name suggest this type of network uses graphs like in Figure 6.11, instead of normal euclidean data structures.

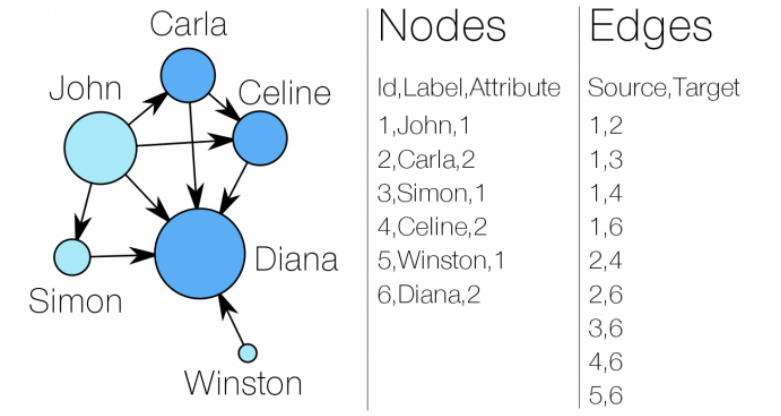


Figure 6.11.: Example of a graph with nodes and edges from[12].

The reason we felt this was a natural technology to investigate is because our problem already works on a graph and it felt natural to apply this method to our problem.

The first problem was how to represent our data as a graph. Even though our problem lends itself to a graph representation, it was at this time presented as an image. We ended up converting the heightmap to a graph node with each pixel being a node with edges connecting

it to its four closest neighbours. The value of each node was the height at that point and the value of the edge was the travel distance between the two nodes.

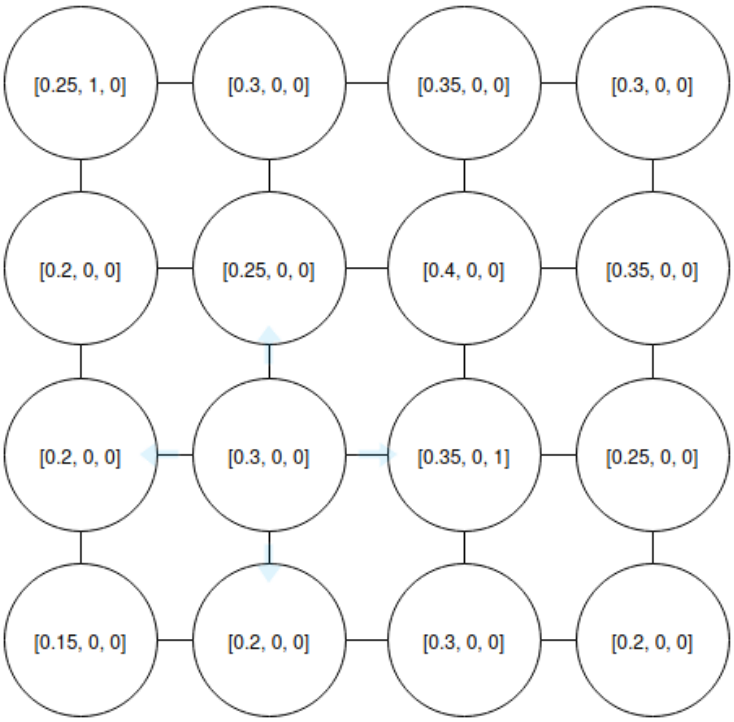


Figure 6.12.: Example of a image converted to a graph. With a start point of $(0, 0)$ and stop point of $(2, 2)$.

With the input of the network defined, the hard part remained. How do we output a path from a graph network? One limitation of graph networks is that the output will always be on graph form with the exact same shape as the input graph. The only parameter you can change is the dimensionality of each node.

In the end, we narrowed it down to two possible approaches for the output. The first was to have the network learn an edge value. The value should let one start at the beginning and always follow the edge with the highest value in a depth first search.

The result would then be the path. The second and the one we went for was to have each node output 1 if the path was on that node and 0 if it was not.

With the input and output defined it was time to investigate what structure the network should have. To keep it simple in the beginning, we decided on a two layered edge convolution solution using a network called spline-conv. This network utilised pseudo-coordinates for each node and a spline to smooth the result. The hope here was that in the future we could extract this line and have a path that is independent of the underlying graph.

6.5.1. Version 1

As this was towards the end of our project, we only had time to make one version of this network. The results were underwhelming.

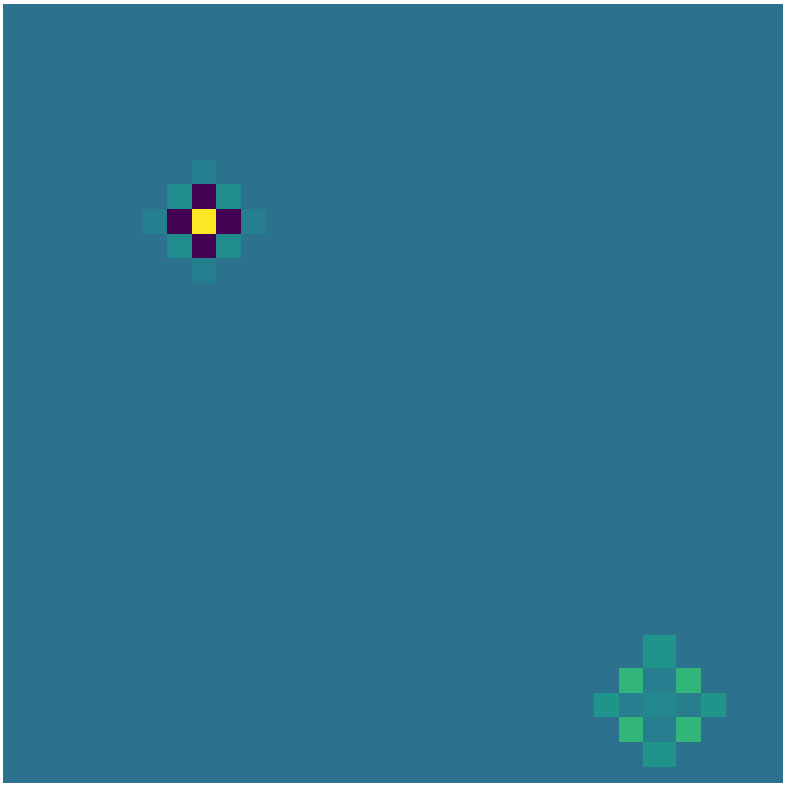


Figure 6.13.: Output of GraphStar v1 converted back into image form.

As you one can see in Figure 6.13 it outputted some data at the start and stop points, but failed to do anything more. We believe this is because the network had a hard time propagating the data throughout itself. In an edge convolution, the output of a node can only be affected by its neighbouring nodes and itself.

6.6. Reinforcement Learning

Note: As to keep the documentation about the machine learning related choices and the implementation details separate, documentation on reinforcement learning code is located in Section 10.12.

The movement limitations of a drone are very strict, with a maximum turn radius and the fact that it cannot change its velocity instantly. After having looked at supervised learning for placing every pixel of the path, there seemed to be some pixel inaccuracies. Because of this it

did not seem like this approach would be suitable for generating a manoeuvrable path pixel for pixel, at least without additional processing.

Instead, having the machine learning output which pixel to move to from the current one until the goal is reached seemed like a good idea. Convolutional networks are designed to analyse pixels in close proximity to each other, which seemed like it would complement this approach well.

As we did not have access to any algorithm which could produce the most optimal manoeuvrable paths, we had no data for training a supervised learning network to do this. So the alternative was to look at this problem as a reinforcement learning5.2.4 problem.

To give a quick recap on reinforcement learning: In normal supervised learning the algorithm's output is compared to the correct output. In a reinforcement learning approach, the problem is divided into steps, where a machine learning algorithm has to pick which action to take at each step, and is given a scalar reward to signify how well it did after each step. The part of the reinforcement learning that selects actions each state is referred to as the agent, and it trains itself via interacting with the environment and learning from the rewards, to always pick the options which in the long run gives the highest rewards.

Because of this approach being divided into steps as, it seemed like a possible solution to our problem. In addition, since the actions available to the agent are designed freely, we could design the actions to take considerations to the movement restrictions of the drone, so that no matter how badly it chose it would still create a flyable path.

6.6.1. Designing an Environment

Training a reinforcement learning agent to solve your problem, first requires having an environment for it to interact with and learn from. Which is why the first step is designing and implementing this. In this section the general design principles behind an environment is discussed, as multiple environments were created for this project.

There are three main parts of an environment, that has to be designed:

- States - the information determining the current state, consisting of all the information that will be available to the agent.
- Actions - which actions are available at every step and how they affect the state, and when the episode ends.
- Rewards - the reward system for every action picked.

States

The state and what makes up the state has to be considered. The state is all the information that the agent should have access to which it will use in its policy to determine action.

It is also important that the information in a state fulfil the Markov Property, which is that future states shall only depend on the current state and not any other information[13]. This is so that the agent acting on the environment has all the information it needs to learn how to maximise the reward.

Actions

An environment is divided into episodes, and each episode is divided into steps. A step is made up of an action, and the processing of that action and how it affects the environment.

Actions are the agent's available methods for interacting with the environment. The agent will have access to the same actions every step. If the environment was an implementation of pong for example, the actions would be to move either up or down (possibly also staying still).

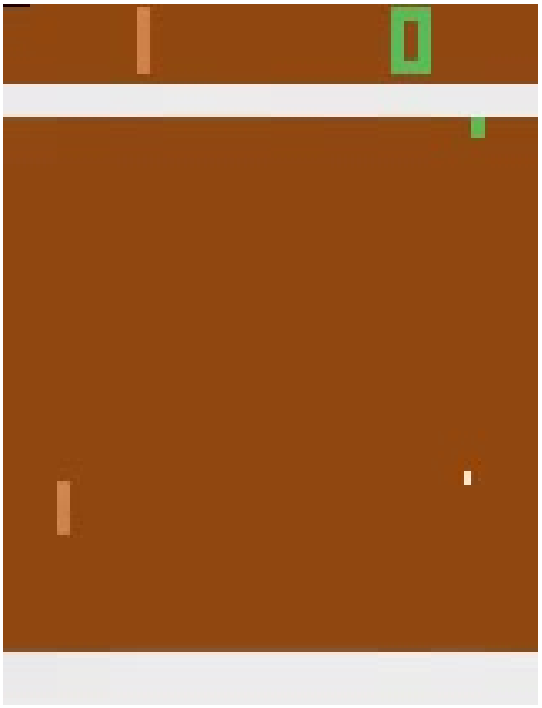
The episode is at some point considered done. Often either because the agent has «lost» the game, too much time has passed, or the agent has «won». Once it is considered done, there is no point in choosing additional actions, and the environment is reset.

Rewards

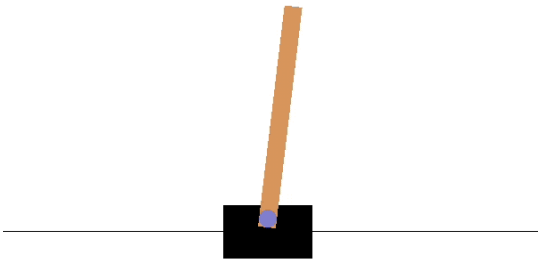
As mentioned before, the reward in reinforcement learning is the scalar value that is associated with each action for each state. The Reinforcement Learning algorithms are developed such that the agent is able to maximise the total reward it gets each episode.

Environments require a form of reward system, to determine the reward for each action at each state, which will in turn tell let the agent now how well the chosen action was. One example of a reward system is OpenAI's cartpole environment[14]. The goal of the environment is controlling a cart either left or right to keep a pole balanced, and the episode ends if the pole goes outside the playing area or falls over. The reward the agent gets at every step is simply one, regardless of the state of the pole (as long as it is upright), which means maximising the reward effectively means balancing the pole as long as possible.

As reinforcement learning algorithms take the reward of future states into consideration when selecting the action on the current state, it will view the actions leading to states with reward as higher reward than those which do not. In the cartpole example this means that actions which keep the pole balanced more rewarding than those which do not, even if both actions have the same reward in this step.



(a) OpenAI's Atari Pong environment



(b) OpenAI's cartpole environment.

Reward functions should not tell the agent how to perform the task, only assess the immediate result of the chosen action. The reward of the action that ends the episode (if there is one) should be zero[7].

An interesting result of the algorithms aiming to maximise reward, is that it will avoid actions that result in negative rewards. This means that instead of focusing on giving positive rewards for good actions, one can focus on punishing the bad actions instead, and it will have the same results.

There are two main types of defining rewards:

- Discrete Rewards - giving rewards for events that happen.
- Continuous Rewards - having a function which continuously give rewards depending on how good the current state is.

Discrete rewards work based on the fact that reinforcement learning algorithms try to maximise reward in the long run, so if you only give reward one time the expected rewards will propagate backwards through the environment.

Continuous rewards make training easier and convergence faster. There are benefits of discrete rewards as well though, such as the agent should avoid specific areas or stay inside others. Mixed rewards are also a possibility, by doing both[15].

6.6.2. Learning Algorithms

As with supervised learning, there is a range of algorithms within the reinforcement learning category. The simplest of which is Q-learning, which is keeping a table of Q-values, estimated reward for an action with future states taken into consideration, and updating the values of the table each time they are chosen to make the estimate closer to the real value. Theoretically with enough episodes and training, the Q-table will eventually be correct and following it will grant the maximum reward of the environment. The problem with this approach is that the Q-table consists of a row for every state, and a column for every action, meaning as the information in the state grows the number of Q-table cells increase exponentially. As the values are updated individually this also increases the training time exponentially.

Deep Q-learning and Double Deep Q-learning aim to solve this, by instead of keeping a table of the values estimating the values using neural networks. The result is that the Q-table doesn't have to be kept in memory no matter the size, and the agent is able to generalise patterns in the states, so that not every cell of the table has to be reached multiple times to be learned. As the states of our environments contain the whole map, using this method or a similar one is necessary, as the amount of potential states increase exponentially with the size of the map. Because of this, Double Deep Q-learning was the first algorithm investigated as a potential solution.

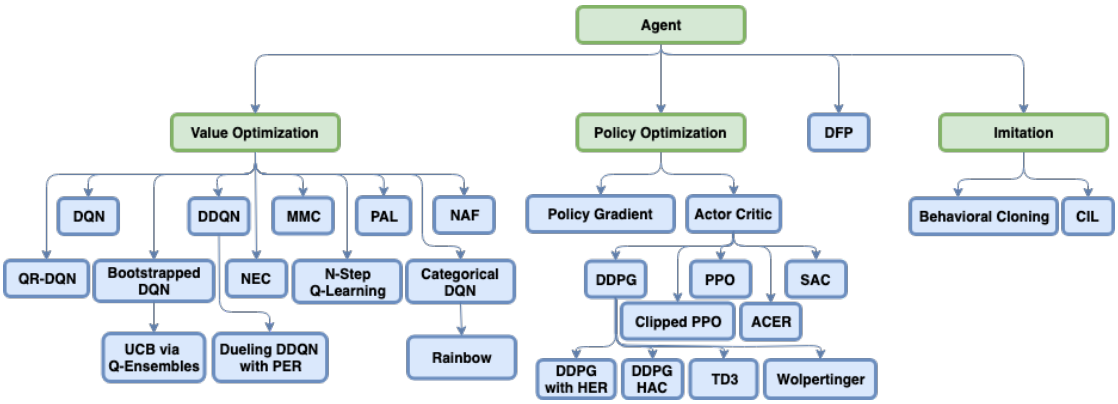


Figure 6.15.: There are a plethora of learning algorithms to choose from[16].

Q-Learning

Specifically, Q-learning works by initialising a Q-table with a row for every state, and a column for every action available. Each cell in the table has a Q-value, which is an estimation for how good an action is when in that state. If the Q-table were perfect, then one would achieve the highest rewards by always selecting the action with the highest Q-value. The trained Q-table is an approximation of the perfect theoretical Q-table, and is adjusted gradually step by step, approaching it.

Each time a step is taken, the Q-value for the action selected is adjusted. It is adjusted based on the reward it just gained by performing that action. But an action is not only as good as the immediate reward it gains, but also how good the future states it leads to are.

It takes into account the value of future states, by taking the Q-values of the actions in the state it arrives at into account. This way, if a Q-value is adjusted based on a high reward it got, the next time that state is reached this Q-value is taken into consideration by the action that lead to it.

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \tag{6.3}$$

Equation (6.3) is the specific equation used to adjust the Q-values at each step[7].

Initialized

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	327	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	499	0	0	0	0	0	0

Training

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

Figure 6.16.: Q-table with all values initialized to zero, with adjusted values after training[17].

By only selecting the actions with the highest Q-value, the algorithm will never learn. It will keep selecting the action it tried before, and not explore new possibilities. This is a problem in scenarios where gets bad rewards this step, but great rewards further down the line.

Because of this, we should force the agent making the selections to explore more possibilities. This way, cells with a high reward may still be discovered, and can be taken into consideration when doing further training. This is done by having a chance ϵ to select a random cell each time it is going to perform an action.

Finding a balance between exploration and exploitation can be done by tweaking ϵ. If one only explores, the agent will never exploit the good actions, and will produce inaccurate Q-Values. Therefore, a common way to balance the parameter is to start with a high ϵ value, and slowly reduce it. This way it will do a lot of exploring initially, but more exploiting later on.

Deep Q-Learning

The main parts of the Deep Q-Learning algorithms are:

- Policy and target neural network
- Replay buffer
- Training loop

The policy and target networks are the neural networks essentially doing the same job as the Q-table; estimating the Q-value of actions based on the current state. They are identical in structure, and can be in the form of any neural network, for example convolutional.

Training neural networks requires a lot of data and many batches of learning. It would be hard for a neural network to learn how to accurately estimate the Q-value of a state it has only seen a few times. The replay buffer aims to solve this by keeping track of a large number of the previously seen transitions, so that they can be trained on multiple times. A transition is simply the collection of information returned when taking a step, consisting of:

- State before step
- Action chosen
- Reward gained
- Whether or not this step ended the episode
- The new state (assuming the episode is not over)

Together these aspects are used in the training loop, which is the part of the code responsible for training the code.

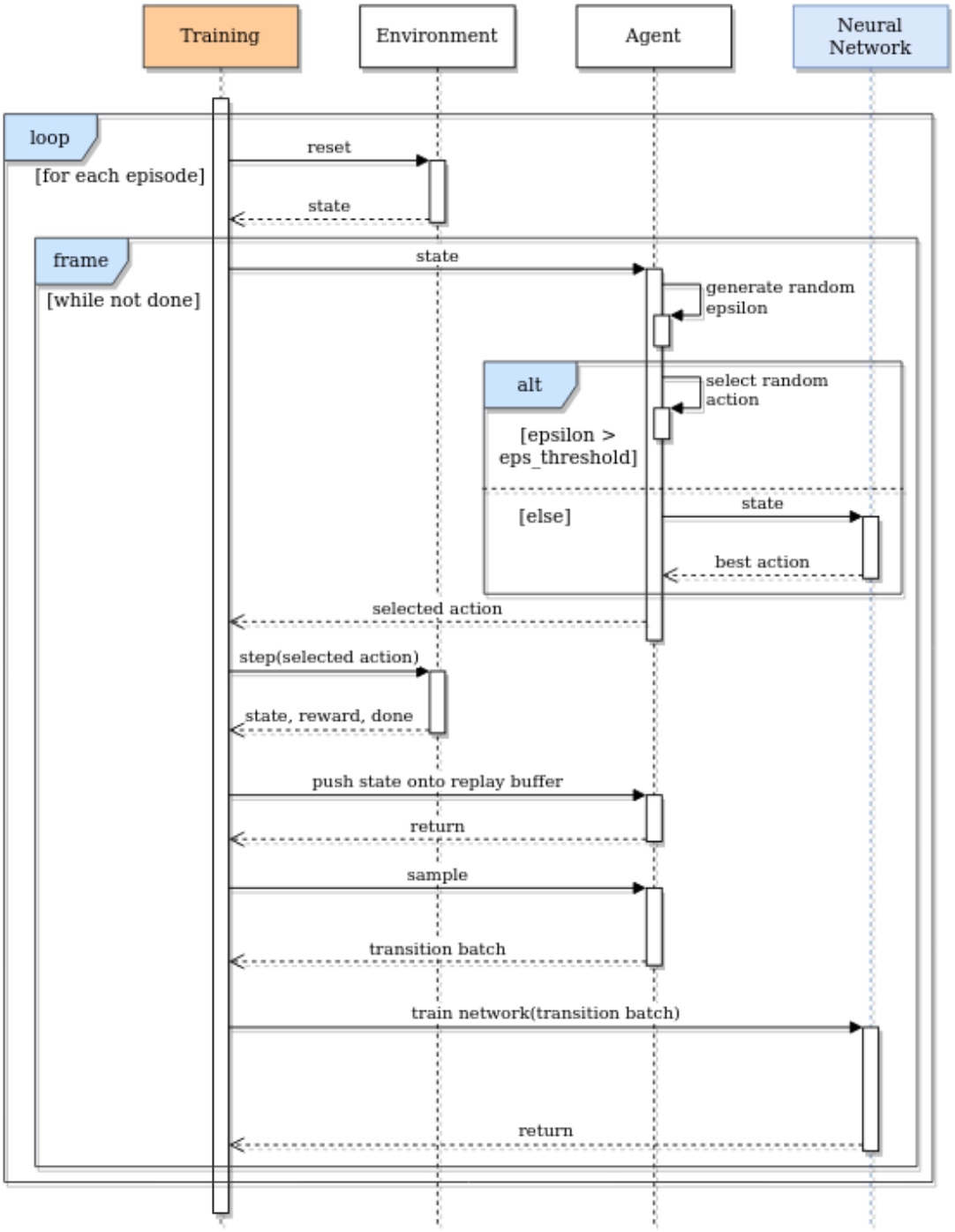


Figure 6.17.: A sequence diagram of the training loop.

6.6.3. Cardinal Directions Environment

Because of the time-related machine learning risks, RI.3.3 and RI.3.5, and as was stated earlier, it is important that we focus on simple representations of the problem before we increase the complexity. So instead of focusing on drones and their movement with turn radius, the simplest movement would be only moving in cardinal directions in a grid. The states and reward function was based off of this core idea of the environment.

The states chosen for this environment is initially to only have the start and goal position be the state with a static map, so that we are able to know whether or not the agent is working, instead of it being hard/impossible for the agent to learn the environment. Then as the environment uses different maps on each episode the map must be a part of the state, so that the agent is able to take this into its decision. As the movement in this environment is simply cardinal directions with no consideration for turn radius or velocity, only the position (and eventually the map) as part of the state is required to determine what the future state will be, meaning it fulfils the Markov Property.

The static map that was chosen as a test map, was one which the upper left corner has zero in height value, whereas the rest of the cells has Manhattan distance in value. Manhattan distance simply means the how many horizontal and vertical cells they are apart, in other words the sum of the absolute difference between the horizontal and vertical positions of the cells. This method for generating the static map was chosen because it is easily implemented, cheap to compute, can be scaled to any size, and provides a map which shouldn't be too hard to learn.

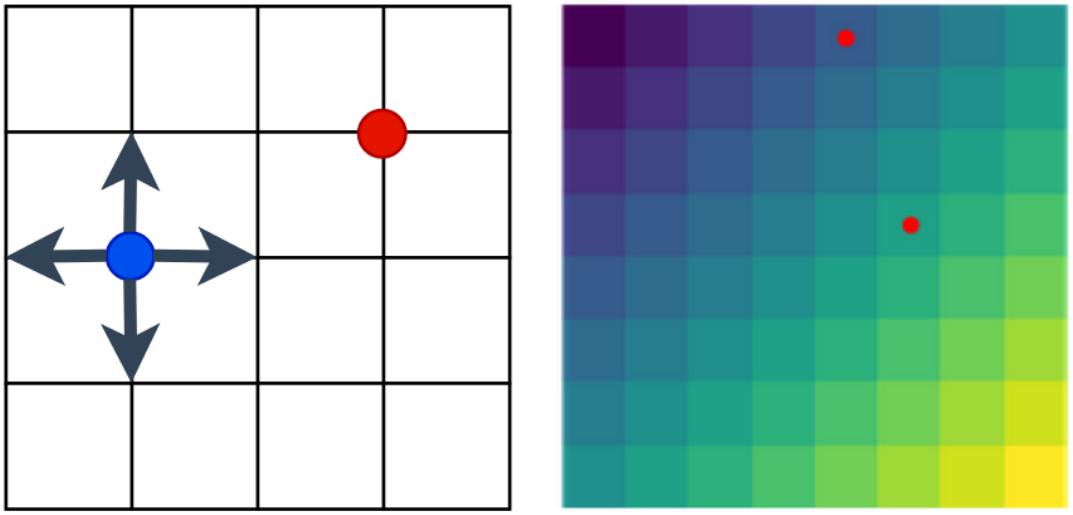
Because the map allowed to scale to any size, it allowed for resizing the grid. The impact of this is that the more grid cells the more possible states, and each state is something that the network has to learn and generalise.

For the episode to end, the drone would have to reach the goal. Because this could end up in the episode lasting infinitely, an additional done condition was added, which was that if the episode lasts too long it will be stopped early.

The initial reward function that was used for this environment, was to give a reward at each step, that has the negative value of the elevation of the new grid cell. The idea was then that the agent would learn that each move it did would punish it, and the only way to stop this would be to reach the goal. This way the path with the highest reward, would be a path that both reaches the goal and has the lowest total height.

Implementing the Learning Algorithm

To create an agent for this environment, as mentioned earlier, Double Deep Q-Network was chosen as the learning algorithm to use. Because there is a number of concepts involved in the algorithm, it was initially chosen to be implemented from scratch, as it would help with understanding how it works.



(a) Movement in the cardinal directions environment

(b) The static map, with dark blue low value, and yellow high

Figure 6.18.: The cardinal directions environment

After having spent some time implementing it, it failed to converge. To verify that it was working properly, and that it was not the environment at fault, it was configured to train on OpenAI's cartpole environment, but did not converge on this either. With machine learning if there is a single mistake it is not unusual for the code to not converge, and it is incredibly hard to pinpoint what is at fault.

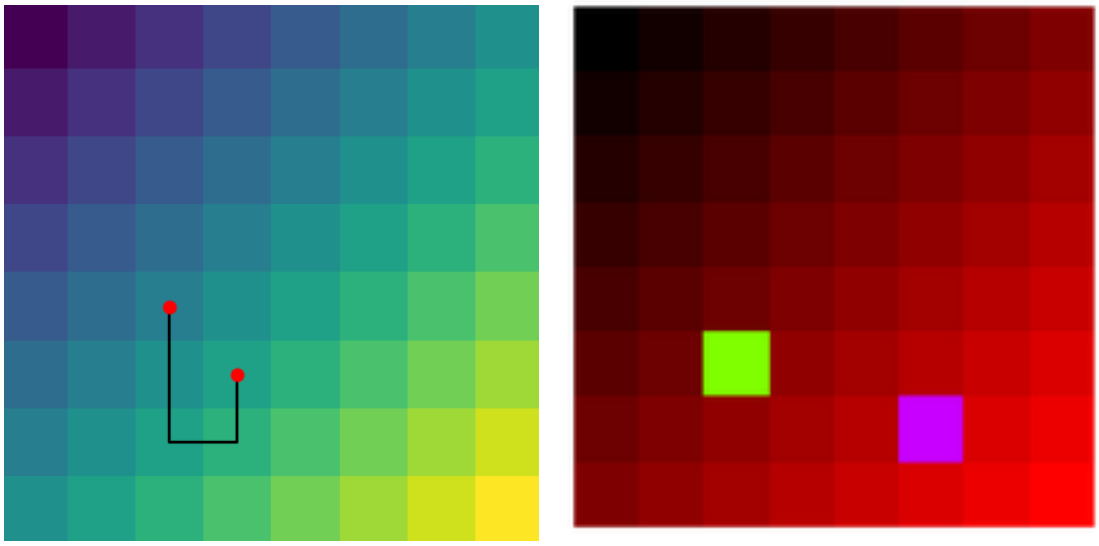
Instead of spending unnecessary time troubleshooting this Double Deep Q-Learning implementation, a pre-made implementation was pulled from a PyTorch tutorial about the algorithm[3]. The time spent was not wasted time though, as it significantly helped in understanding the core ideas behind the algorithm, which is a pretty central algorithm within reinforcement learning.

Rendering

To use convolutional neural network with reinforcement learning, the input of the network has to be in the form of an image. This means that the current state of the environment, along with all the information in the state needs to somehow be conveyed with an image. As usual with neural networks though, there is no definite way of doing this, so it is just a matter of trying what you think will work.

Since we only care about the contents of the image, and it is made on the spot, kept in memory and not stored as a file, these images will be referred to as RGB-arrays. These images are

[3]https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html



(a) Notebook (b) RGB-array (with heightmap in red channel)

Figure 6.19.: Render modes of environments

implementation-wise just arrays with three channels for red, green and blue, where every array has an index for each pixel, and each pixel can have a value from 0 to 255.

The process of creating the images, will be referred to as rendering.

As stated before, the state of this environment was only comprised of the current position of the drone, as well as the goal position, with heightmap planned to be added as well. Because there is only three different properties, with there being three channels in an RGB image, it was well suited to put one of the properties in each.

Since the previous positions of the drone was not a part of the state, because they do not affect the current choice, they were not represented in the image. Somehow displaying the different previous positions the drone has been in is useful though, as it is hard to determine anything about the drone's behaviour without this information.

Therefore it made sense to create a separate rendering method, which displayed this additional information. The Notebook rendering method was created, which displays the start and stop position of the drone, as well as the heightmap, and the path the drone has taken. Both of these rendering methods are shown in Figure 10.2.

Recall that an episode is made up of episodes, and steps that make up those episodes, where there is a reward for each step. And because the point of the reinforcement learning is to maximise the reward gained, it is helpful to have some graphs displaying these properties as well.

Therefore, two useful graphs are also added to the display of the training:

- Episode durations - a graph displaying each episode, and how long that episode lasted. For this environment, reaching the goal ends the episode, so the shorter the duration the better.
- Episode reward - the reward gained each episode. More reward is of course better. Because reward can vary wildly between episode, a mean line is also plotted, to show how it is progressing over the episodes.

These three panels - the two graphs and the notebook rendering - was helpful in showing the progress of the learning. They are shown in Figure 6.20.

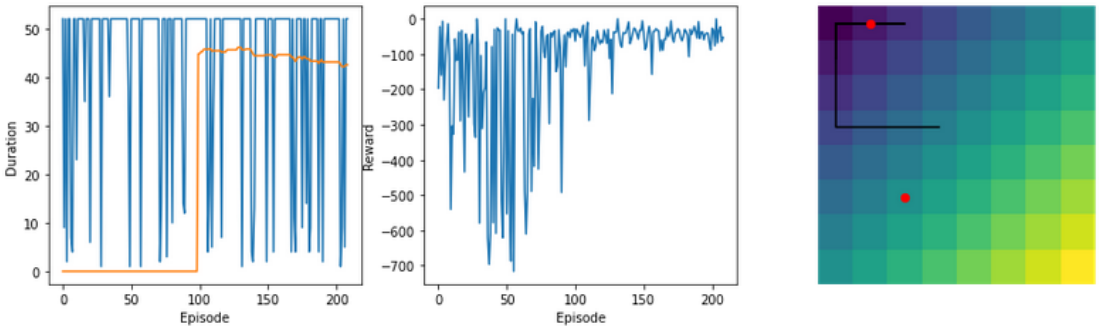


Figure 6.20.: Episode Performance Plotting

6.6.4. Training on the Cardinal Directions Environment

After training for several hours, the graphs showing the agent's performance showed no improvement. In other words the model failed to converge. As usual, there was no easy way to pinpoint what was at fault. There were several things that could be causing this:

- Mistake from when the learning algorithm was changed to work with our environment.
- Parameters of the learning algorithm and/or network, causing the converging to be slow.
- Bad reward function, resulting in the environment to be hard to learn.

To solve this, process of elimination was used to remove one by one until the problem was solved. As the easiest one of the three to troubleshoot was the reward function, this was simplified into an even easier reward function so that the learning function would guaranteed be able to learn it.

Instead of taking the height into consideration, the reward function was made into only giving positive reward for moving in the right direction, and a negative reward for moving in the wrong direction. This reward system is visualised in Figure 6.21.

Even with height removed from the reward function, it did still not converge. With there being many parameters to tweak, and changing them showed no obvious change in performance, it was hard to tell whether or not this was at fault.

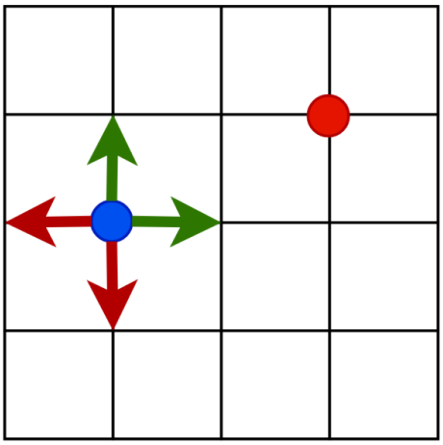


Figure 6.21.: Positive reward when moving in direction of green arrows, and negative for moving in direction of red arrows.

To test whether or not the learning algorithm was the problem, a traditional Q-learning implementation was made. This Q-Learning implementation turned out to be a lot quicker, because the algorithm adjusting the model was a lot faster than the one using a neural network.

This let us tweak the parameters, and see the results of them significantly faster than with the Deep Q-Learning implementation. It allowed us to tweak the parameters and the environment in such a way that it would finally converge.

Specifically, it was noted that a too low epsilon would end up making the path only go in one direction. As well as adjusting the rate at which it took into consideration the Q-Values higher helped it converge.

In addition, it was helpful to add a better way of determining whether or not to end the episode early. By adding a variable counting the number of mistakes made, or bad moves, and ending the episode early if there were too many. This also helped keeping more important states in the replay buffer. For example, if the agent runs into a wall two hundred times in a row, these new states may push out other states it could learn more from.

Training With Height Consideration

As discussed earlier, the chosen planned reward function for this environment was to take the negative value of the height of the cell moved to. This way, since the episode only ends when the goal is reached, it would constantly be punished for not having reached the goal. To maximise reward, it would have to minimise the total punishment, which could be done by reaching the goal and minimising total height.

This was implemented in both the reward function, and the state information. The rendering method was changed to convey the new state information, by displaying the height values of

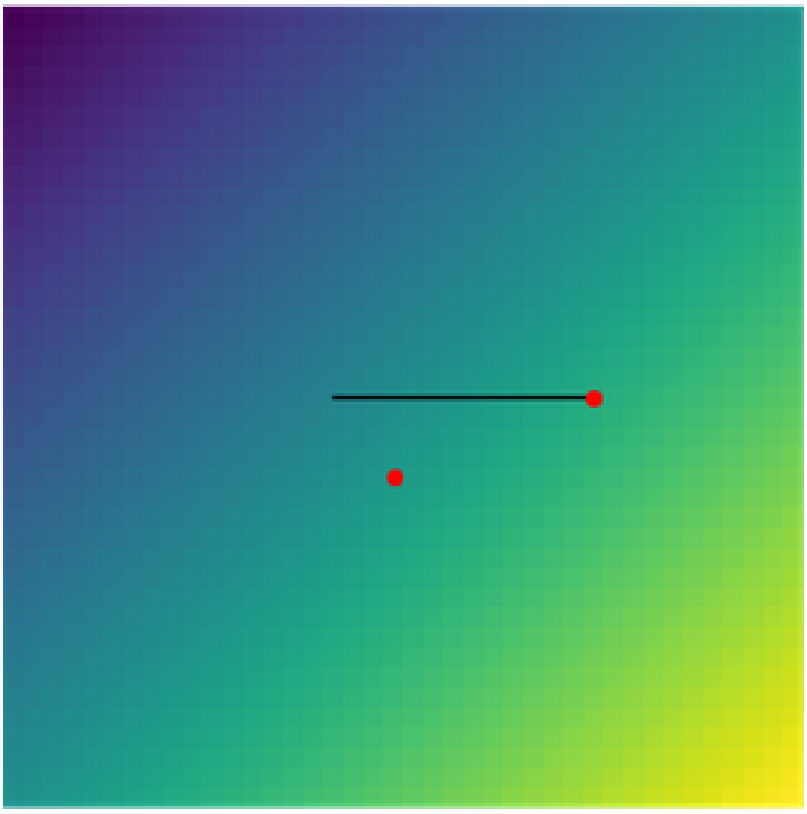


Figure 6.22.: Path as a result of too low epsilon.

the grid in each pixel of the red channel in the RGB-Array.

After training it failed to converge. But because the height values was a part of the state information now, implementing this via a normal Q-Learning algorithm to try another learning agent, converged fine. Though because of the static map, there would theoretically be more possible states. But this does not affect the Q-Learning, as because of the static map only a limited amount of states would be visited. Therefore the additional possible states do not make any difference.

To reduce the amount of parameters, as well as trying an alternative machine learning architecture to see how it would fare; The structure of the network was changed to a deep feed forward structure, which is when there is no convolutional layers, and every neuron is connected.

A deep feed forward structure takes scalar numbers as input, so rendering the network was not necessary, and it could instead take the values of the state information, instead of being conveyed as an image.

This network model converged rather quickly, but would scale badly when increasing the size of the grid. Though it would converge at a grid size of 5x5, it failed for larger sizes and showed

little to no improvement at 10x10.

Because it was seemingly the convolutional aspect of the network structure which kept it from converging, it was suspected that the method of rendering was not well suited for the converging part.

In hopes of improving the converging speed, several options for the rendering of the RGB-Array was implemented:

- Blurring points - Blurring both start and stop a certain size. In other words the centre would be 255 in colour, and it would linearly decrease proportionally with the Manhattan distance, until it was zero.

  The idea behind this is that convolutional neural networks only activate the surrounding neurons in the next convolutional layer, that colouring a single dot would have little impact after these layers. Therefore the blurring, so that both the start and the stop would activate more neurons, while still accurately conveying the positions.

- Centring the drone position - Looking at implementations for reinforcement learning agents which learned the cartpole environment mentioned earlier, one of the rendering tricks used was to crop the image around the cartpole. This way the cartpole environment would keep the images more static even with different states, and it in theory help generalising the information.

  This was implemented in our rendering methods, by padding the sides and rendering the drone position in the middle of the image. The padding was done so that no matter the position of the drone, no pixels would need to be sacrificed or would be out of the frame of the render.

  In theory this would hopefully allow the model to only generalise better the position of the drone and how it affects the environment, and be one less thing for the agent to learn.

These approaches made seemingly little to no difference in performance.

While taking a step back to analyse and think about why the convolutional aspect of the model made it not converge, we found . It noted that when having pooling enabled when training on position related data, it would converge worse.

Similarly, it was noticed that during the process of rendering the RGB-Array, the arrays were resized to fit the structure of the convolutional neural network. This resizing had by default enabled interpolation of the images. Interpolating an image means using an algorithm to try to make scaling an image more smoother and less noticeable. This often means smoothing out the image to make the stretching out less jarring. Interpolation caused the image to be harder to generalise for the agent, as it would blur the position of the drone inconsistently.

With these changes made to both the agent and the rendering method, the model finally converged. But because of a bad reward system, its behaviour was well rewarded, but did not perform as intended.

Modifying the Reward Function

Recall that the current system is to reward with the negative of the height of the grid in the heightmap. With the only ways of ending the episode to reach the end, or taking too many turns. The static map currently in use is upper left corner is zero in height, with all other cells having height value of Manhattan distance from this cell.

With the model converging with this reward function, it would simply move to the upper left corner and stay there as much as possible. It failed to learn that moving to the goal is worse reward this step, but much better in the long run, as it would end the episode earlier.

It would also do a great job of detecting ways to exploit the reward function. On moving, if the drone tried moving to a position outside of the grid, it would get moved back and given zero reward. It quickly learnt that by repeatedly doing this it would get no negative reward, in contrast to moving around on the grid. This resulted in it spending all its turns moving into the wall.

This is not uncommon in the field of reinforcement learning. Exploiting design flaws in the reward function not intended by the developer, is called *reward gaming*[18]. In general, if there is something potentially exploitable, reinforcement learning tends to find it. It was for example able to discover a new bug in the 1983 Atari game Q-Bert, which was not known[19].

Most likely because it has two learn two policies which it tries to maximise, both staying in the low height zones, as well as reaching the end, it fails to balance them. Instead it ends up only maximising reward from one of them.

Attempting to fix the problem of it not reaching the end by adjusting the reward function, the idea was that combining the reward function with Dijkstra's algorithm would make it possible to learn it what the correct way is.

By calculating the cheapest cost for getting to the goal position using Dijkstra's algorithm, for all the choices available for the agent at that state, we can compare whether this move was the most optimal or not.

The different modifications on reward functions that were tested, using Dijkstra's Algorithm were as follows (best means best cost, current means current cost):

- 0 reward when hitting wall, as it would not move - It learnt that walls were safe to bump into, so would rush to a wall and stay there.
- $best/current$ otherwise. Would travel infinitely, refusing to go to goal as it would keep gaining reward if it did not.
- 1 if optimal, and $-\frac{best}{current}$ otherwise. As the reward for going in correct direction were greater than going wrong, it would get close to the goal, then further away and repeat this to farm points.
- 1 if optimal, and $-\left(\frac{best}{current}\right)$ otherwise. Failed to generalise the path in any meaningful way, just bad paths.

None of the attempted reward functions utilising Dijkstra's algorithm showed promising results. This method brings with it the problem of how do you define a sensible reward function from the results of a pathfinding algorithm. We found no obvious answers that gave intended results.

In addition to the problem with the cost function, as shown in Figure 6.23 even with converged paths, it provides sudden unexpected movement. This indicates that if this approach is to take consideration for drone manoeuvrability its actions would either have to be verified to confirm its flyable, or change the environment such that only the allowed movement are selectable actions.

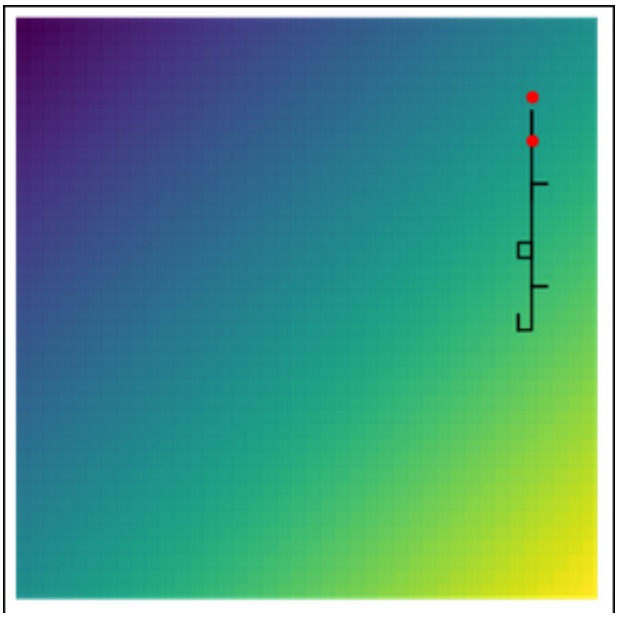


Figure 6.23.: Shaky reinforcement learning path.

6.7. Movement Restrictions

After having investigated the problem giving the algorithm full control over the path, with the intention of looking at taking movement restrictions into consideration at a later stage, we did not get to a point where it would be sensible to introduce that additional complexity into the data. But as we wanted to investigate how well machine learning dealt with movement restrictions, we looked at the problem using another approach.

Whereas our first approach was to look at machine learning solutions where the machine learning is responsible for controlling the path in its entirety, therefore choosing all the points of the path. Instead the approach we wanted to investigate was to let the majority of the path creation be handled by an algorithm which could produce a path, and have the machine learning only control aspects of it.

To give an example of this idea, consider the scenario where the path is composed of two straight lines. The points of the path are then the points along the straight lines, and the normal evaluation method of minimising the sum of elevation of all the points making up the path still applies. The inputs for the machine learning is the heightmap, as well as the start and stop point.

Would the machine learning in this scenario, be able to determine where to position the midpoint in such a way as to minimise the total elevation of the path?

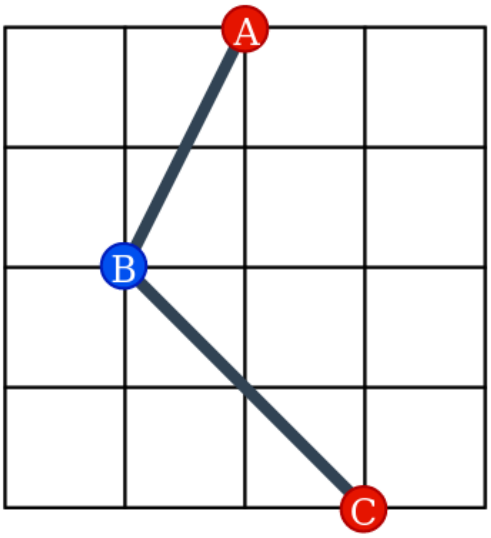


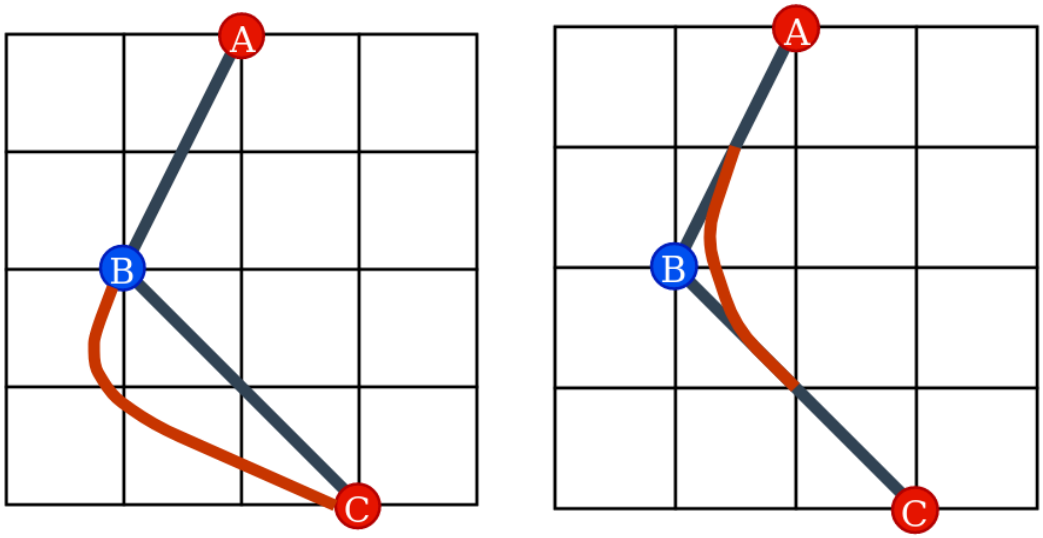
Figure 6.24.: A path consisting of two straight lines.

By taking this approach we are reducing the control the machine learning has over the path, to instead try to make it learn how to utilise the algorithm to create the path with the least total elevation.

The main benefits of it is that the decrease in control means less for it to learn, although possibly at the expense of this output being more complex and harder to generalise. In addition to this, we are able to use path generating algorithms which ensure that the resulting path is manoeuvrable by a drone.

As our main motivation for using this approach was that we could see how well machine learning fared with movement restrictions in place, we wanted an algorithm which could produce a simple manoeuvrable path. The easiest such a path would be one like the example, with the machine learning outputting the midpoint. But because of drone turn radius, for such a path to be flyable the midpoint angle would had to be smoothed out.

To smooth out the turn, there are multiple alternatives. One can either assume that the curve is taken as soon as or after the turning point is reached, a post-turn. Or one can assume that the turn is begun before the turning point, a pre-turn.



(a) Post-turn (b) Pre-turn

Figure 6.25.: Types of Turns

Both the alternatives were discussed but the latter of which, the pre-turn, was chosen because it has the benefits:

- As the heightmaps when used in the modules are in the form of arrays, taking a post-turn might have the turn end up outside the grid, for which there are no defined height values in the array.

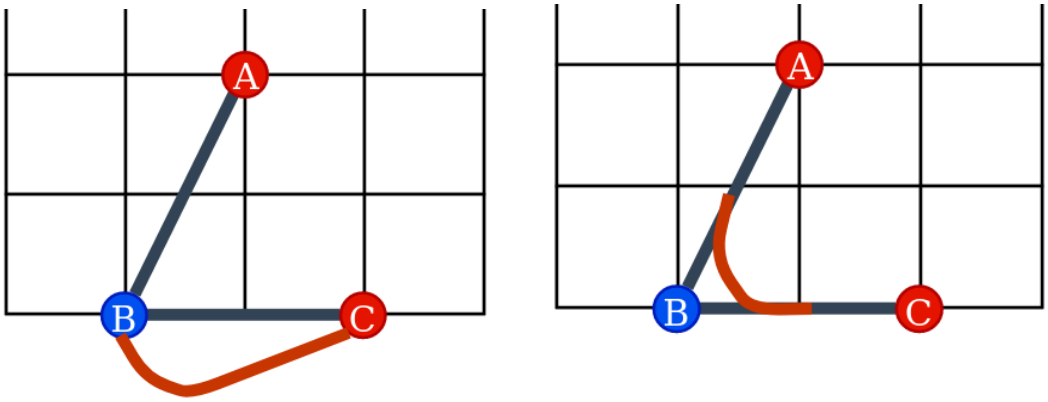
  Doing the turns in the pre-turn method, ensures that the turns end up inside the grid, assuming all the three points are inside.

- Imagine a turn on the points A, B, and C, where B is the turning point. The three points will make up the two vectors AB, and AC. Doing a pre-turn would maintain the BC angle, when it arrives at C. Whereas with the post-turn method, this angle is not maintained, making this angle vary more and harder to find.

  This may only seem like a minor point, but when considering paths consisting of two turns or more, meaning four or more points, knowing this arrival angle is necessary for calculating the rest of the path.

When the path is expressed as a geometric set as lines and curves like this, using an alternative method of getting the total elevation of the path is possible. When taking the sum of the heights of the cells visited during the path as the total elevation, straighter paths will inherently be better, as seen in Figure 6.27a and Figure 6.27b. This is because of the imperfections that are brought into the scenario when bringing a diagonal path into a square grid.

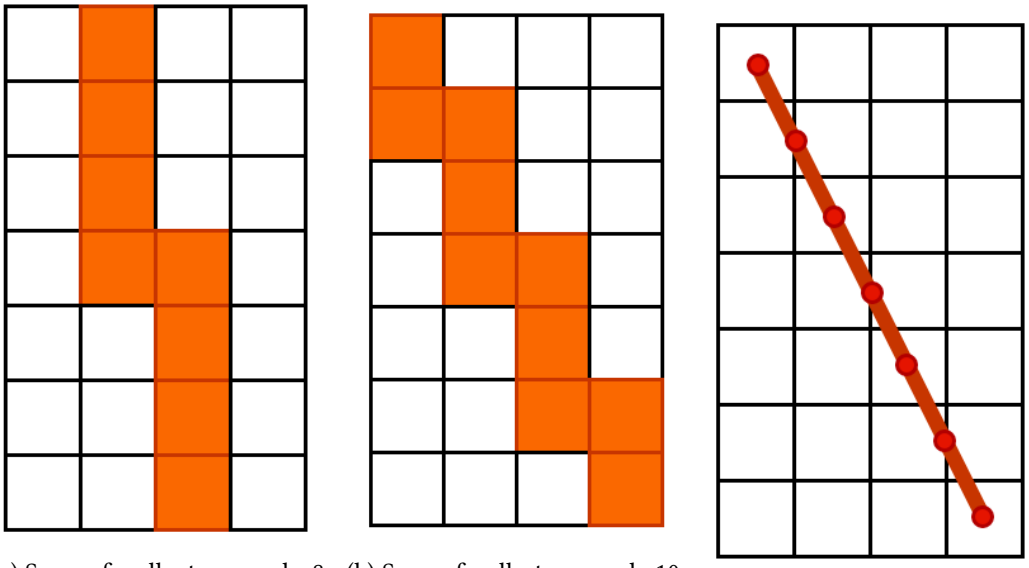
The alternative method of getting the total elevation, is to follow the line/curve, and once you've moved a specific distance along the line/curve take the height at that point. This is



(a) Post-turn near edge

(b) Pre-turn near edge

Figure 6.26.: Near-edge turn comparison



(a) Sum of cells traversed, 8 cells

(b) Sum of cells traversed, 10 cells

(c) Sampling height along line

Figure 6.27.: Methods of finding total elevation

essentially sampling the height, and avoids this diagonal issue. This sampling method is illustrated in Figure 6.27c.

This method of doing turns was implemented as a Python module, as a part of the *gym-drone* package on GitHub at LAPS-Group/gym-drone. Because it starts the turn "short" of the turning point, we refer to it as the turn short algorithm, and the module is called the *turn_short* module.

Implementation

The Python module consists of three public functions:

- flight_possible
- shortest_turn
- get_turn_height

Some geometric information was inferred by looking at the problem, which was used to solve how to implement this:

- The turn will be between points A, B and C. A and C being the Start and Stopping points, B being the turn point. Forming vectors AB and BC. The angle of ABC is α.
- The turn of a drone can be interpreted as a circle arc, or a circle.
- It is assumed that the turn will be be symmetrical, in relation to B. This means that the centre of the turning circle, will lie on $\frac{\alpha}{2}$.
- The point at which the turn begins/ends, will be a tangent to the circle. This means that the angle between B, this point, and the circle centre will be 90. Forms a right triangle. On Figure 6.28 this is the triangle formed by point a, B and the centre.

As there is a right triangle, we are able to use sin, sine, cosine and tangent functions on it. Function *flight_possible* uses this to calculate the radius of the circle for the point closest to the turning point. This tells us the maximum turn radius which can make the turn at that point. If the turn radius of our drone is less than this, then it is able to make the turn.

Similarly *shortest_turn* uses the almost same approach. But by knowing the turn radius of the drone, we can instead calculate the starting end ending waypoints of the turn, and the circle centre position.

Using the notations used in Figure 6.28, the equations would be:

$$radius\ of\ turn = \left|\vec{Ba}\right| \cdot \tan\frac{\alpha}{2} \tag{6.4}$$

$$\left|\vec{Ba}\right| = \frac{turnradius}{\sin\frac{\alpha}{2}} \tag{6.5}$$

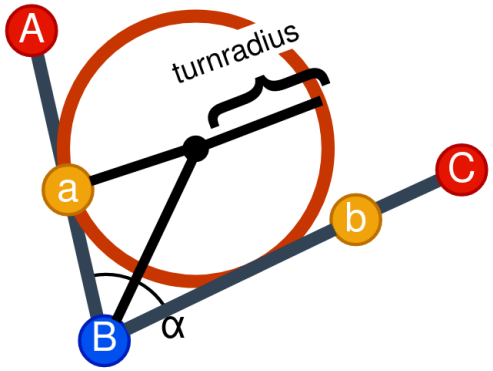


Figure 6.28.: Geometric representation of the turn short.

Lastly, *get_turn_height* utilises these functions to find the two straight lines, as well as the circle arc forming the turn. Then sampling is done as explained previously.

6.7.1. Dijkstra with movement restrictions

Because graph start version one was not showing much promise of future prospects ,and it was only two weeks left before the group would start focusing only on documentation in preparation for the hand in. It was decided that we would stop working on graph start and focus all of our efforts on reinforcement learning. Because at the time it showed more promise.

With one person freed up the first thing that was done was to integrate the turn short algorithm into Dijkstra. The hope here was that we could then generate the most optimal flyable path using the Dijkstra algorithm.

To enable Dijkstra to use any custom movement algorithm the idea was to generate new edges for each step Dijkstra does. So instead of using the graph edges you give the graph into a custom function that then outputs all edges Dijkstra can use. In this case that algorithm is the turn short algorithm as explained aboveSection 6.7.

By doing it this way it meant that Dijkstra could support any custom movement algorithm with minimal modifications. But it did also introduce some potential problems. Because the turn short algorithm needs two points to determine if it can swing to a destination, this implicitly means that Dijkstra's available edges is not just dependent on the node its currently on but also the previous. And because Dijkstra considers a node closed when it has been visited once it means in some very specific cases it can be unable to find a path when one is available.

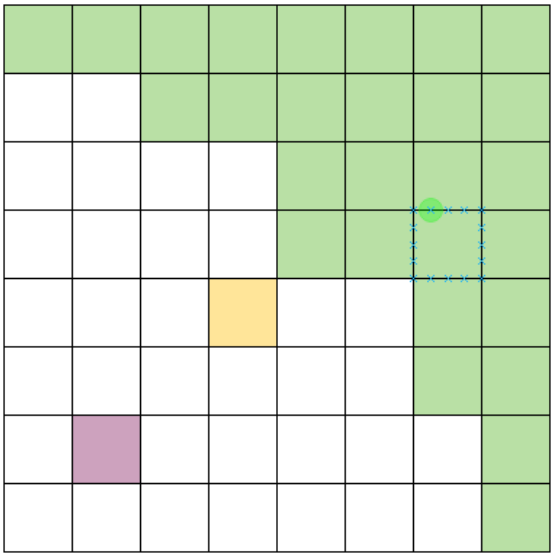


Figure 6.29.: Example of dijkstra with the turns short algorithm. Purple and Yellow is the previous and current point, and green is the available edges.

However this is an extremely rare case when using real world heightmaps and the situation will probably never arise with our data. And as you can see in Figure 6.30b it has the option of using the edges beyond the intersection so it would still find a path. Therefore it was not a priority fixing this, but it is still important to be aware of it.

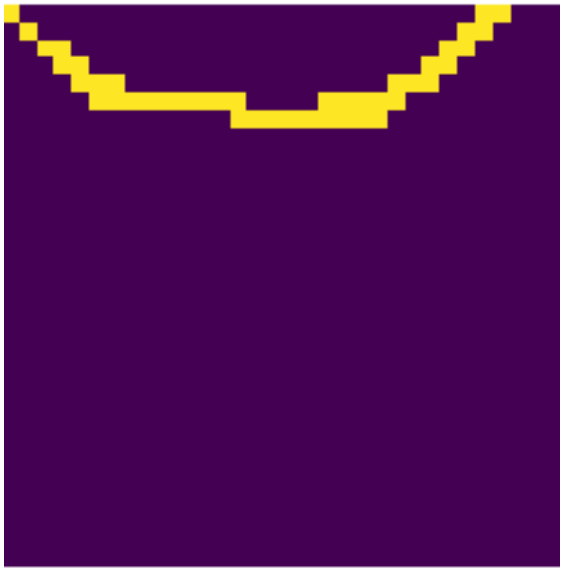
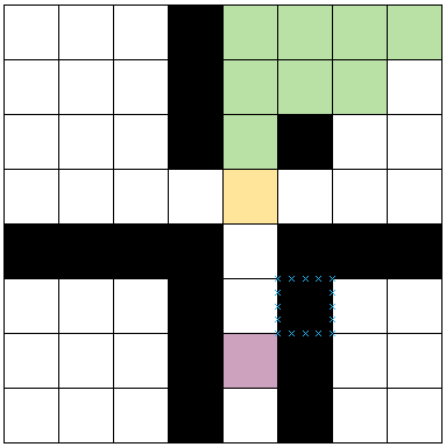
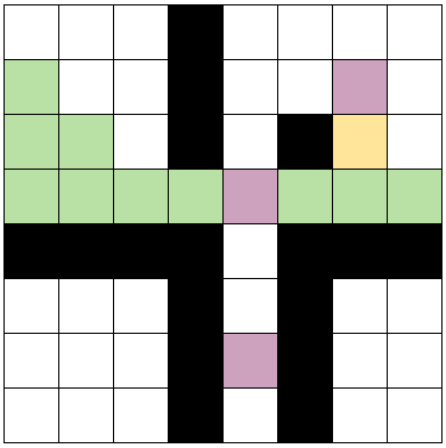


Figure 6.31.: Example of Dijkstra with turn short enabled.



(a) Dijkstra uses node at intersection to get into top quadrant.



(b) Dijkstra cannot now use the node at the intersection.

The last thing to consider with this new implementation, we have already discussed how computing time grows exponentially with the size of the map for the normal dijkstra. This becomes even worse with the turn short version, the main time sink is the recomputing of edges for each step the algorithm has to take. On average it is about 5x slower.

6.8. DeepStar Version

To round off the investigation into supervised learning with CNNs we decided to generate some training data using the new and modified turn short Dijkstra. This is to see if the network could converge on a more flyable path. As show in Section 6.4.2 does converge at least somewhat. The hope is that we would get about the same results just with more flyable paths.

6.8.1. Version 4

In addition this version would generate an image of the path instead. To do this it would use an autoencoder as described in Section 6.4.5, with one modification. Instead of recreating the image from the feature map the decoder would generate a image of the path instead.

To keep things simple the Dijkstra algorithm was restricted to one turn short point. This mean that it could essentially place one point to turn with and then go to the destination. This also meant that the problem of it being slower than normal Dijkstra is somewhat alleviated, on a 32x32 pixel image it went from about one path per second to four paths per second when generating data.

Results

This approach at first seemed very promising reaching a very low loss. However it ended up being a cautionary tale of why you need to understand every aspect of your code. Because the loss function that was used is the MSELoss function, this loss function worked fine for all the other version however it had an subtle flaw that made it inadequate for this use case.

Because in the image that the autoencoder targeted most pixels would just be empty, This was because the goal was for the output image to just contain a pixel path. With most pixels being empty when you took the average error of all those pixels, just outputting a blank image gave it really high accuracy. This was not technically wrong but it is very unproductive for this case and made the loss look really good while in reality it meant nothing.

The fix for this problem was to change to a loss function that is more adapted to the situation. That loss function is called Binary Cross Entropy Loss, this loss function is similar to the one used in DeepStar version 3 and can be read more about here `https://pytorch.org/docs/stable/nn.html#bceloss`.

6.9. Reinforcement Learning

We wanted to see if reinforcement learning would be able to learn how to utilise the turn short algorithm, and how it would perform. By creating an environment in which the actions select the positioning of the turn short points, it could possibly learn how to use it to select waypoints resulting in good paths.

The least complex way of using the turn short would be to create a path with only one turn. This way the first and last points of the turn short would be the starting and stopping points of the turn, and the turning point would be the only point left to choose.

Each time the environment is reset, a random start and stop point is chosen, and the agent is tasked at selecting where it thinks the most optimal turning point will be located.

This same approach could be done using supervised learning, by using a dataset which has a heightmap, with start and stop indicated, and the best turning point labelled. However a problem arises when considering multiple turns.

If for example considering a 20x20 heightmap, with one turn there is approximately 20^2 points where the most optimal turn point may be located. But when you are considering two turns, you have to find the most optimal position for both the first and the second turning point. This exponentially increases the complexity, and for only two turns there would be 20^4 solutions.

Generating such a dataset, by iterating through all the possibilities, would be infeasible. Especially when considering that a dataset may require tens of thousands of entries for a network to generalize it. Thus reinforcement learning was instead used, as it would circumvent the need for training data.

6.9.1. Implementation Details

Using the same method that was used in DeepStar Version 3, instead of using a value in the range [0, 1] to indicate position, having individual labels indicating chunks of the grid. Each chunk represents a possible position on the heightmap the turning point may be.

To translate this into an reinforcement learning environment, the state is comprised of the heightmap, as well as a start and stop position. The actions available to the agent, is selecting which chunk to place the turning point in. Instead of using coordinates for selecting the chunks, there is one action available for each chunk. Since there is only one turn, there will only be one turning point to place. This means there will only be one step until the episode is considered done.

Visualised, the start and stop points are points A and C respectively, and B is the selected turning point in the grid in Figure 6.25b and Figure 6.26b.

Because the grid is divided into chunks to choose from, instead of selecting an individual pixel on the grid, it reduces the amount of possibilities which can be chosen between. The size of the chunks can also be increased, resulting in less chunks.

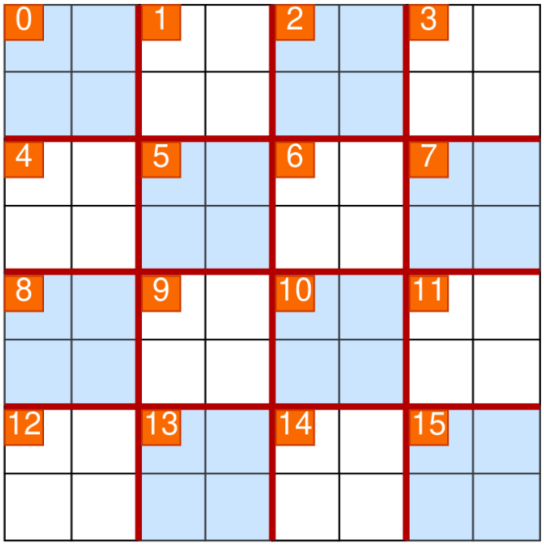


Figure 6.32.: An 8x8 grid, divided into a 4x4 chunk grid. Each chunk is indexed.

The reward system for the environment is similar to the reward system used in the previous environment, where the reward for each step is the negative of its total height. Because this is using the turn short approach, the total height is calculated using the sampling method, though this should not make any difference.

As not all turns are possible because they require turning angles too steep for the drone's turn radius, these are an exception in the reward system. If an action representing a chunk which creates an impossible turn, then the reward is instead an arbitrarily chosen negative number, with higher magnitude than most badly selected paths.

Instead of using the static heightmap for this approach, as two different types of heightmaps were used:

- Height maps from Norwegian Mapping (NMA) Institute in the Kongsberg area, made for and previously used in DeepStar, 10,000 images.
- Artificial height maps, provided by project's external supervisor. A total of 100,000 images in 31x31 resolution.

Before using the heightmaps they were both normalised. Normalising data before using it in training means pre-processing it, so that the values in the data are within in a common range. It is an important step of machine learning which may increase stability of the model as well as learning rates. Implementation and explanation for the normalisation process is in Section 10.12.

In this context, normalising the heightmap images will affect the best path in most scenarios. This is because the sum of the values of the pixels (representing the height) is used to find the most optimal path. We chose to ignore this consequence of the normalisation though in favour

of the increased learning rates, until we received enough progress where we could investigate those consequences.

Rendering was done like last reinforcement learning environment; red channel for indicating height, green and blue for start and stop positions respectively, both blurred.

6.9.2. Training Results

Training on the artificial images provided faster converging rates, and as such, we chose to use these artificial images for training. This could be due to multiple factors, among them:

- Data quality - the images in the heightmap were of varying quality. Some had spots with zero height data, and some of the images were blank.
- Quantity - the number of images may have helped generalize the learning better.
- Data more similar - the artificial height data only had smooth height bumps. Whereas the NMA heightmaps had a wide of features, such as trees, cliffs, and buildings, causing abrupt height differences in the images.

Training was first done with a single convolutional neural network layer. As mentioned earlier, starting with a simpler network structure increases the converging rate, but limits how well it performs. In this case it converged after approximately a day, and learned how to get better rewards. As predicted, it did not perform well. The points in every step were seemingly arbitrarily placed, and frequently attempt to make impossible turns.

The same training was attempted on a more complex network, with three convolutional layers and no pooling. It kept improving after training it for several days. Unfortunately, each episode kept taking more time, most likely because of some sort of a memory leak.

Training Result Discussion

As the convergence was not reached in the model, it is hard to tell how good it is able to do reinforcement learning on this training data. Nonetheless, it shows that a convolutional network like this can converge on smooth heightmaps without some of the more complex features of a real heightmap.

Network structures used in this investigation were low in complexity, with only three convolutional layers. Even with this low complexity we did not have time to manage to see how well it performed after having converged, as we got to this point exactly at the end of the project and were out of time.

7. Architecture

In order to achieve our requirements, in particular those derived from U.1[1] like F.2, we need to have a solid architecture to build upon. The architecture describes how the different components in the system fit together. This is very important in our case, because our project is split in two parts; pathfinding implementation and a web service.

7.1. Considerations

These are the considerations we had in mind while designing the architecture:

- There must be a separation of frontend and backend.
- The user must be able to select between different pathfinding implementations.
- Pathfinding implementations must be able to be added and removed while the system is running.
- Height data must be readily available across the whole system.
- It should be as simple as possible to implement.
- The architecture must be robust enough to allow the system to run for weeks continuously.

7.2. Development

We spent a lot of time in the first few weeks discussing how the architecture should be. Once we had decided on something, we stayed true to our design to the end for the most part. Some changes had to be made further down the line as we added more features, leading to previously unforeseen situations which had to be handled.

The initial architecture design was very much a team effort, involving everyone in the group. After that, Vetle and Håkon discussed the amendments to the design, in particular how module management should be handled.

[1] See Appendix G

7.3. Design

We decided that the best course of action would be to have an architecture which is split up into multiple parts. We wanted to have separation of concerns for the pathfinding implementations and the rest of the system. We refer to one of these implementations as a module.

Each module is it's own independent program, taking the form of a Python script[2]. We chose Python because it is a very commonly used language in the machine-learning world, and is the language the machine learning team wanted to work with. This approach has several benefits:

- Simplicity of development: the algorithm can be developed and tested completely independently of the system if one wants to. It can then integrated later on without much effort. The actual system integration can consist of a simple library.
- Ease of implementation: Because each module is independent, we don't need to have some other program which forwards messages to the right module, reducing complexity.
- Containerisation: Each module can run in a containerised environment. This means that it is almost impossible for modules to interfere with each other or the system accidentally[3]. We use Docker to do this in the system.
- If a module fails or crashes, it cannot take the whole system with it.
- All we have to do to register a new module is to launch it in a new process.

Figure 7.1 shows our software architecture. As one can see, it is based around Redis, which we use to pass messages and as a general purpose database. See Section 4.4, and in particular Section 4.4.2 for an explanation of how this works.

We base our architecture around Redis for a few reasons:

- Because we can use it as both a database and a message broker, there's no need for another database, reducing complexity.
- Our data is non-relational in nature. We would not really see much benefit from a relational database.
- It is highly performant, so using it as a middleman will not affect the speed that much.
- Using some kind of message broker makes communications a lot easier.
- Simplicity. Redis is very simple and easy to use. This helped make development easier and the final product less complex.

[2] If we wanted to, we could support different programming languages as well, since the modules are independent programs which follow a protocol.

[3] It is possible for a module to interfere with the database as they must have access to it to function. Very recently, Redis 6.0.0 was released, which added support for restricting access using a user system. We could generate a user for each module, and only allow them to touch the parts they need to work.

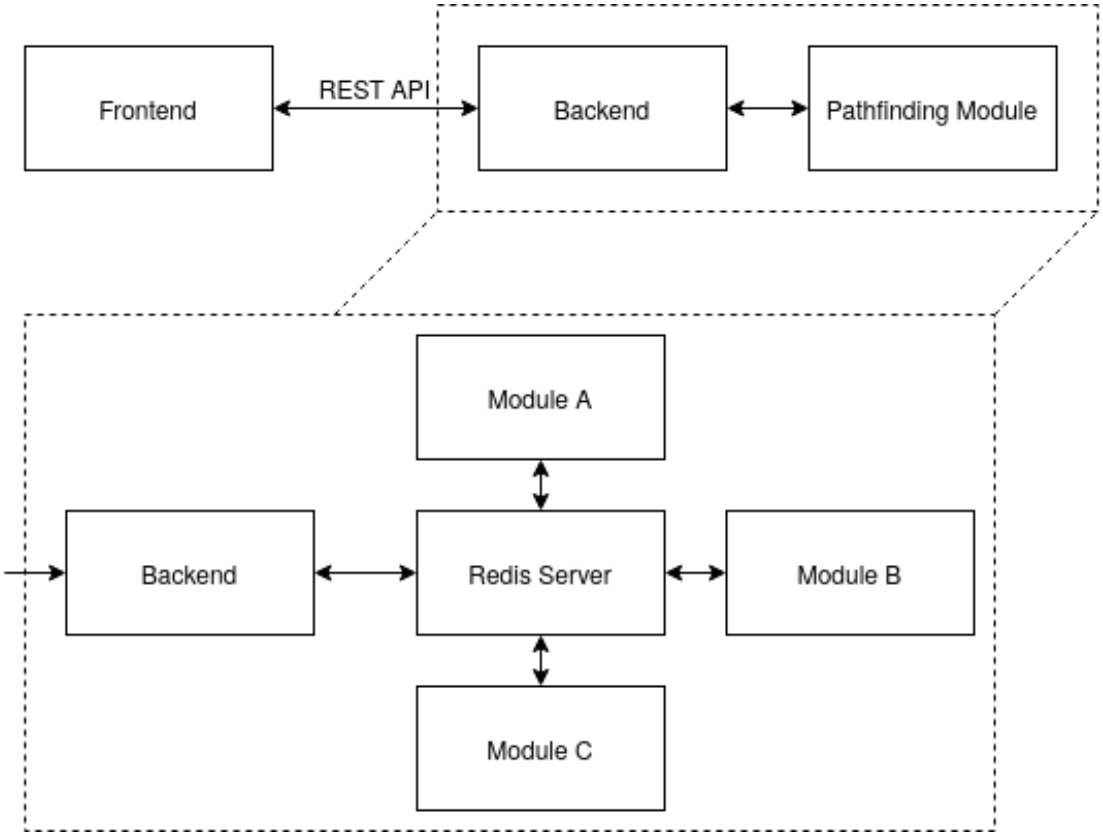


Figure 7.1.: The LAPS software architecture

7.4. Backend communications

Like we said, we use Redis to pass messages. This is a different approach from what we might have otherwise written where we would have to manage connections to each module. Instead, we only have to manage our connection to Redis. If we were to restart the backend, it wouldn't cause all pathfinding modules to fail because the connection was closed. Instead, they do not know that it happened at all, and everything will starts working again once the backend starts up.

The messaging specifications can be found in the Appendix E. Below, we give a short explanation of what the communications between the backend and the pathfinding modules. Note that every message is serialised as JSON[4].

[4]See Section 4.7 for details.

7.4.1. Startup and Shutdown

When a pathfinding module starts up, it has to tell the backend that it exists. It does this by sending the registration message containing it's name and version. This is used to keep track of which modules are available in the system. The backend listens for registration messages on a single Redis key.

Conversely, we have a message to do the opposite: the shutdown message. It has the exact same contents as the registration message, but removes the module from the list of registered modules. This keeps the list of registered modules up to date without zombies. Just like the startup message, the shutdown messages are also sent to a common Redis key.

7.4.2. Accepting Pathfinding Jobs

When a module is registered, it is ready to receive pathfinding jobs. It does this by receiving a job request from the backend and returning the path. The job request contains everything that the module needs to calculate the path. This includes, but is not limited to: Map ID, start point, end point and job ID. The job ID is a simple number which must be contained in the response to the backend. The backend uses this number internally to tell jobs apart.

Each module has it's own work queue that it listens for jobs at. This is the best way to find out which jobs it should do, because it allows us to use blocking list operations in Redis, and we know that the job ends up in the right spot.

7.4.3. Returning the Result

After receiving a job, the pathfinding module does it's thing and produces a result. The result is the path represented as a list of points. After calculating the result, they are sent to a job results list. This list is not unique to each module, and instead is where every single module puts its results. This is where the job ID comes in, to determine which job we are getting the results to.

The jobs can have three different outcomes: success, failure and cancelled. If no errors occur the outcome is a success, and the resulting path is returned. A failed or cancelled job result is used to give the user feedback that something has happened. This is a much better situation than waiting indefinitely for a result that will never appear.

7.4.4. Logging

Sometimes we want to know what a module is doing and what events lead it up to that state. We reach for logging in this case. So as an attempt to integrate logging into the system, we decided that the modules should be able to send their own log messages to the system. These

are picked up by the backend which stores them. In our architecture implementation, we also make the module output log messages to standard output.

This works similarly to returning job results, except that we send the module name and version instead of some arbitrary ID. These messages contain the log level and a UNIX timestamp of when the log was recorded.

7.4.5. A Note on Parallelisation

By using a work queue, we can fairly easily have several workers completing jobs in parallel. This is a fairly easy way to run several jobs in parallel. No changes are required by the module developer. The downside is that it does not help with delay, only throughput. A job will still take 20 seconds to complete, but multiple jobs can be executed at the same time.

We decided to embrace this. The only tweak we had to make to the architecture was to add a worker number field to the log messages to tell the workers apart. This helps us filter out which message came from what worker, which is helpful when trying to understand what went wrong in the module.

7.5. Backend API

While we can use Redis to communicate in the backend, we also have to send and receive commands to the website. This is done using a REST API as shown in Figure 7.1. Our API is described in detail in Appendix F.

7.6. Frontend

To communicate between the frontend and the backend, we are using a normal REST API. Unlike the software architecture for pathfinding modules, we came up with it as we went along. We had no way of knowing how everything would be implemented, nor what features we would end up actually having the time to implement, and to some extent the API needed to reflect this. We wanted to have a single-page application as much as possible, which is why we needed an API rather than using server-side template rendering. However, it was important that whatever API we came up with was well-documented, which we feel we have done.

7.7. Map data

When we export height data from the NMA, we receive this as a GeoTiff file. This is an Tiff image file with additional geographical data, such as projection, coordinates and resolution. This file format is not very widely supported outside of the geographical world, and it would

be much easier to convert into a more wide-spread format. GeoTiff format files should in theory be readable by any image reader compatible with plain Tiff files, but most of the readers we tried spat out an error message when it reached the geographical tags.

Therefore, it was decided that we should convert the data into a much more common format to make handling the map data easier. We decided to go with PNG, as it is a lossless compressed format which is widely supported. Because PNG does not support geographical data like GeoTiff does, we extract metadata when doing this conversion as well, and make it available through the REST API and in the database.

This converted image is greyscale and normalised, i.e the lowest height point is set to be completely black, and the highest point is made completely white. This has the effect of making all heights relative to each other, which means the algorithms perform the same no matter the general elevation of the area one is in.

We store the data in two hash sets in Redis: one for the actual map data, and one for the metadata. The image data is stored at `laps.mapdata.image` and the metadata at `laps.mapdata.meta`. They both indexed by the map id. By storing the mapdata in Redis, we ensure that they are easily accessible by not just the backend, but also by each of the pathfinding modules.

8. Frontend

8.1. Purpose

The frontend serves to present the product and be the user. The frontend should present what a user, not a developer, needs to use the product. It should be as easy to use as possible, with a self-describing interface.

8.2. Software

All websites use HTML to describe its structure. A pure HTML website is static, it cannot change. This would not suffice for this project. CSS and JavaScript are needed to give websites their appearance and interactivity respectively.

Cascading **S**tyle **S**heets, or CSS for short, allows us to style websites. It can be used to change where HTML elements are placed, fonts, colours, sizes of elements and much more. JavaScript is a scripting language which can modify the contents and style of a website dynamically. It is a full-blown programming language which also sees use outside of the browser.

8.2.1. Vue

VueJS[20] or simply Vue, is a JavaScript framework. It makes it much easier to make user interfaces, because pure JavaScript can be quite unwieldy.

The main thing that sets Vue apart is it's reactive components. Components allow one to break up frontend code into chunks, which are responsible for one task each. Components consist of the HTML, CSS and JavaScript needed to create and run that component in the browser. For example, a component could be a login form. The HTML part would contain the form elements, the style part the CSS, and the script the code to send the login form to the server.

Reactive components take it a step further. It allows binding data to components. When the data changes, the website is automatically updated to match the new data. With pure JavaScript, we would have had to tell it how to update the page. With Vue, this is not necessary.

8.2.2. Axios

Axios[21] is a library that helps us make HTTP requests. We could have used the built-in JavaScript API's for this, but Axios is much easier to use. Because we have to make a lot of API calls to the backend, this makes the code easier to write.

8.3. Sending Job Requests

One of the first features added was for the user to find paths between two points. This is a basic feature that should be implemented early. The initial implementation was simple, as the backend was also in early development and only needed coordinates to calculate a path. The coordinates had to be typed into text boxes, and the frontend would send the appropriate message to the backend.

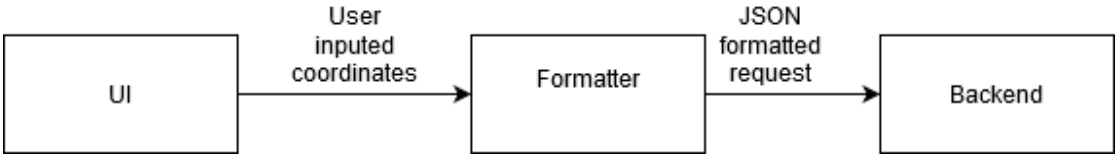


Figure 8.1.: Data flow of early job request feature

To implement this, we used Vue's reactive feature. We bind an input field to a variable, which will be automatically updated whenever a user types something in the field. In the case of Listing 3, it is bound to coordinates.start.x which is the start point's x-coordinate. We can also change the variable in code and have the change be reflected immediately.

```
<input
  v-model="coordinates.start.x"
  @change="fieldUpdated"
/>
```

Listing 3: An input element bound to a variable, which calls the `fieldUpdated` method when the data changes.

There are four of these input fields, which are used to input the x- and y-coordinates of both the starting point and the end point. The `fieldUpdated` method is run when any of the fields change. We will come back to this function later.

8.3.1. Selecting the Map to Use

As the service grew, more features were added. Eventually, we added the ability to select the map to use when calculating the path. This required a way to select the map in the frontend.

There was an added map ID field in the API call to find the path. We added a dropdown menu to the frontend to allow the user to select the map. The dropdown menu gets its elements from the backend through an API call. The updated flow of data can be seen in Figure 8.2.

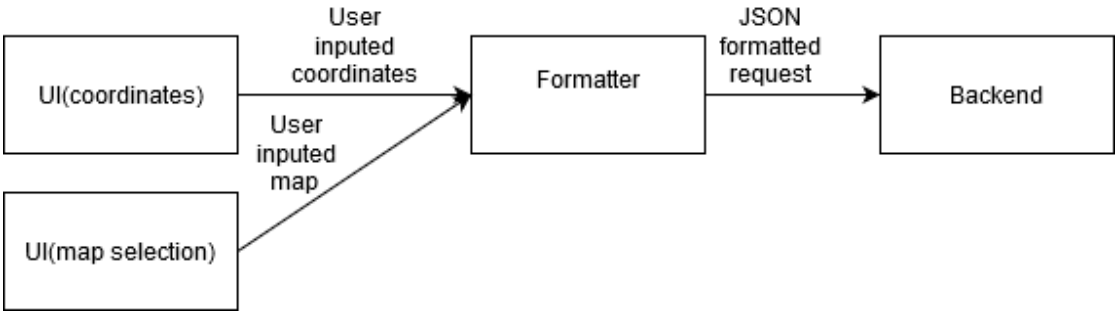


Figure 8.2.: Frontend flow diagram with selection of maps added.

8.3.2. Selecting a Pathfinding Algorithm

The last element added to the request is the user's choice of algorithm. This was also implemented with a dropdown menu. The flow diagram in Figure 8.3 illustrates this.

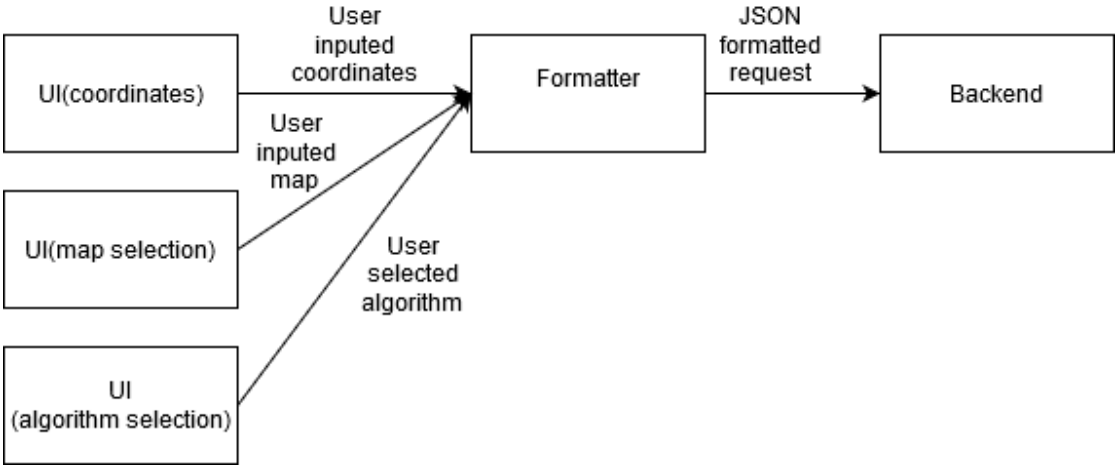


Figure 8.3.: added map input

8.3.3. Collecting the Data

With the coordinates, map id, algorithm and algorithm version, we have everything we need to find the path. Most of the user data is collected from different elements of the UI, which are controlled from different components. This creates the need to enable communication between the different components.

```
{
  "start": { "x": 0, "y": 0 },
  "stop": { "x": 1, "y": 1 },
  "map_id": 1,
  "algorithm": {
    "name": "sample-algorithm",
    "version": "1.0.0"
  }
}
```

Listing 4: Example data the backend needs to run a pathfinding job. Taken from Appendix F

The store serves as a centralised location to store any JavaScript object. This is useful as it allows us to collect all the user input in one place. We will describe the store a little later.

After collecting all the data, we can craft the request that the backend needs to calculate a path. The body of an example request can be seen in Listing 4.

We fetch the coordinates from the store and convert them from strings to integers:

```
this.coordinates.start.x = parseInt(store.markers[0].x);
this.coordinates.start.y = parseInt(store.markers[0].y);
this.coordinates.stop.x = parseInt(store.markers[1].x);
this.coordinates.stop.y = parseInt(store.markers[1].y);
```

Then we get the selected map id from the store:

```
this.coordinates.map_id = store.map_id;
```

Finally, we fill in the algorithm name and version:

```
this.coordinates.algorithm.name =
  store.selected_algorithms.name;
this.coordinates.algorithm.version =
  store.selected_algorithms.version;
```

Sending the Job to the Backend

We take the object containing the user data and convert it into JSON as a string. The backend requires that the content type HTTP header is set to JSON, so we have to do that too. The request is made asynchronously which means the code should not block and wait for it to complete, but instead run other code until an answer is returned. When that happens, we can continue and use the result. This function will only request a job with the specified coordinates, map and algorithm. The backend returns a job token is given which then can be used to request the results of the job. The code can be seen in Listing 5.

```
//convert coordinates to JSON
let message = JSON.stringify(this.coordinates);

//Start the job based on sent information and
//returns id to fetch result when done
let res = await axios.post(getRoute("/job"), message, {
    headers: {
      "Content-Type": "application/json",
    }
});
```

Listing 5: The code that sends the job request to the backend

8.4. Getting the Job Results

After requesting a job we do not get the result of the job, but instead we get a token that represents the job. This is because jobs are expected not be finished instantly. We use a technique known as long-polling to get the results of the job. By using a token system, job results can be reused, without having to rerun the job.

First we make a request for the job result. The function will wait until a result is given. If the result is not ready, the backend will return a code indicating that we should poll again. The backend does not send any status updates. The code can be found in Listing 6.

Note the error handling code. The backend returns the error code 504 when the result not ready yet. If this happens, we just have to try again and hope we get the result back this time.

If the request is successful we should have received an array of coordinates. This list of coordinates can be followed to get from the start point to the end point. We send the returned array to store, so it can accessed elsewhere in the code.

```
getJobResult: async function () {
  try {
    const c = await axios.get(getRoute("/job/" + this.job_token));

    mutations.setrecivedCoordinates(c.data);

  } catch (error) {
    console.log(error);
    //If the error is a time out send a new request
    if (error == 504) {
      console.log("504:timed out sending new request");
      this.getJobResult();
    }
    else if (error != 504)
    {console.log("something went wrong", error)}
  }
}
```

Listing 6: The code which polls for job results.

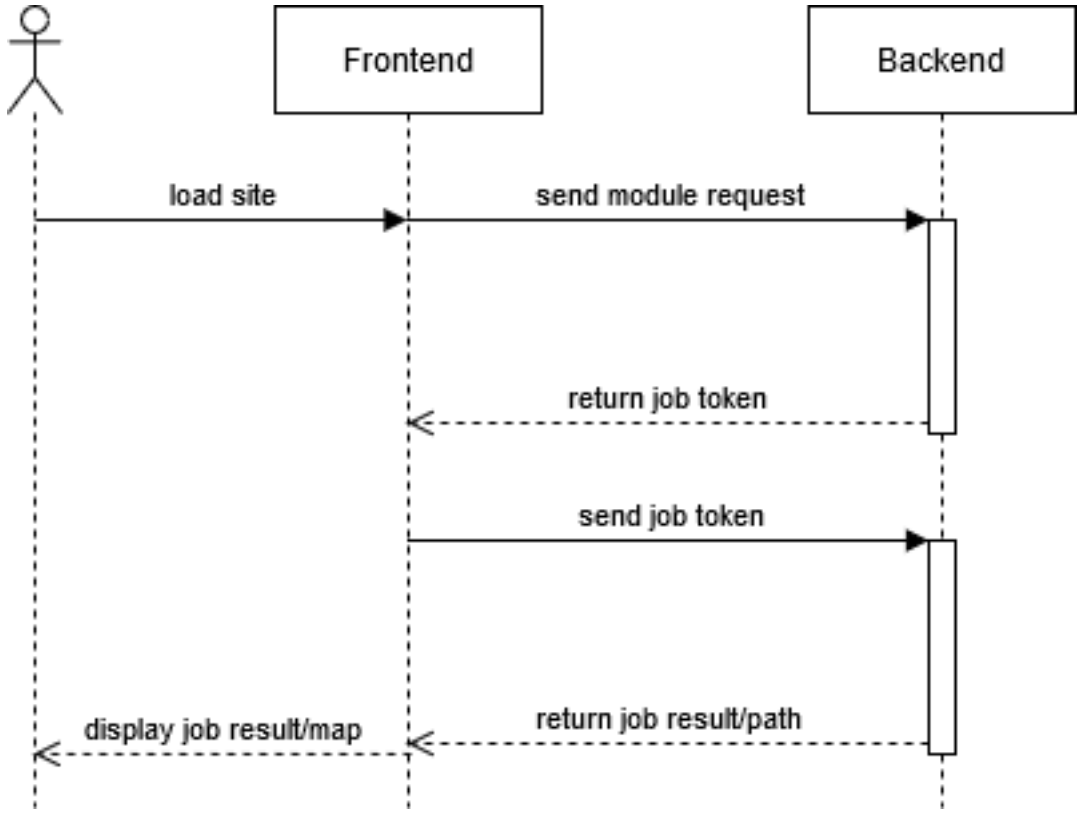


Figure 8.4.: Diagram of job result

8.5. The store

The service is split over many components. The store is a place where data can be stored by any of the components. This is helpful if we have a piece of data that needs to be used in multiple places, especially if the data is needed in another component than where it is from. This allows for centralised data storage.

However, because the store can also be used by all components, we should be careful of what data we store in it. If a variable is used in multiple places, it can be overwritten somewhere by accident which causes problems. Troubleshooting can also be painful as it is not obvious where the store has been updated. Therefore the store should only be used where necessary. Writing data should have a higher cost than reading data. Something like the users selected start and stop point make sense to put in the store as it is expected to be used by multiple components, but only written from one place.

If multiple functions rely on a piece of data, it is easier to manage the many relationship to the store, that many relationships to the source. As long as the source is in sync with the store, all the other dependencies should also be provided with the correct data. If there is a lot of traffic between two components and only between them, a special relationship should be made, like a child/parent relationship.

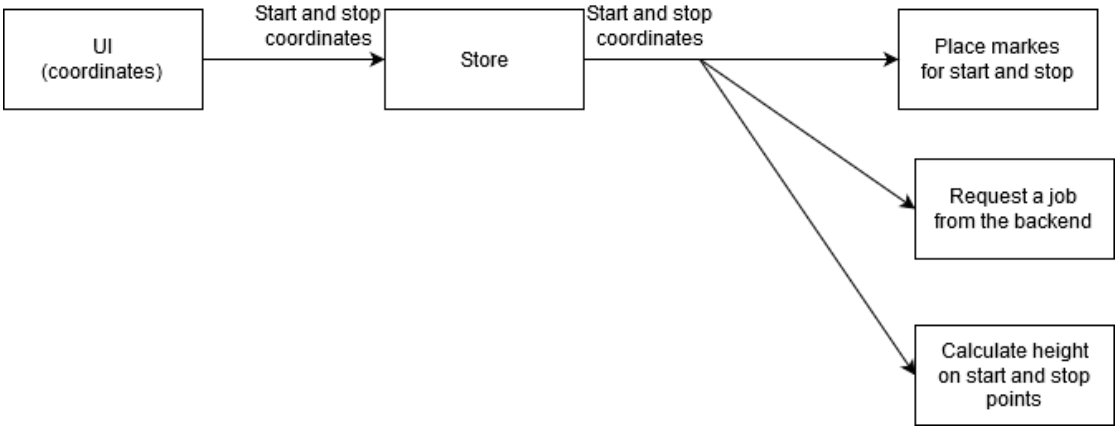


Figure 8.5.: Example of how multiple function uses the same data sent through the store

This function is run after the input field for coordinates changes. The coordinates are updated in the store and from there can be accessed by other parts of the frontend.

```
fieldUpdated() {
  mutations.setMarker(
    this.coordinates.start.x,
    this.coordinates.start.y,
    0
  );
}
```

The function that is called in the store

```
setMarker(x, y, markerNumber) {
  store.markers[markerNumber].x = x;
  store.markers[markerNumber].y = y;
}
```

Then we can call the variable wherever we needed it. Here we get the currently selected map id from the store.

```
this.coordinates.map_id = store.map_id;
```

8.6. Select algorithm

When the website loads a request is made to the backend to list all available algorithms. These are in return displayed in the form of a dropdown menu.

```
this.algorithms_arr = await axios.get(getRoute("/algorithms"));
let i = 0;
for (i = 0; i < this.algorithms_arr.data.length; i++) {
  let alg =
    this.algorithms_arr.data[i].name +
    " " +
    this.algorithms_arr.data[i].version;
  this.options.push({ text: alg, value: i });

  console.log(JSON.stringify(this.options[i]));
}
```

The returned data is put into a for loop. The for loop is simple. It runs through the length of the returned algorithm data. It assigns a value which is the same as the spot in the array. Because of Vue we can't simply select a spot in the array based on the index. This is because Vue has to add a new variables to its reactive system. Therefore if options[3] is not defined during render, even if we declare options as an array, options[3] would be undefined. The easiest way to do

this with array is simply to use push to add a new value, as this also adds the new value to vue render. Using push to add a new value isn't exactly uncommon, but if we don't know why push works, but directly setting the value doesn't can cause some issues. If an value is already declares either at render time or through adding later the value can be changed directly.

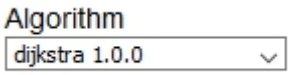


Figure 8.6.: Dropdown Menu

After we have a selectable, formatted list of the algorithms, we must display them. We bind the select HTML element with the new array with all the options we just made. Option.text is the what the users will see and option.value is the value that will be used when the user select that option.

```
<select v-model="selected" @change="onChange($event)"
  class="drop-down">
    <option v-for="option in options"
      v-bind:value="option.value">
      {{ option.text }}
    </option>
</select>
```

After the user selects an option, the onChange function will be run. When an algorithm is selected it get updated in the store so that other functions can use it. This where we get the selected algorithm from, when we send a job request.

```
onChange(event) {
  mutations.setselected_algorithms(
    this.algorithms_arr.data[this.selected]
  );
}
```

8.7. Select map

Both maps and algorithms are present the user in a dropdown menu. The difference is that the map menu displays all available maps and the algorithm menu display algorithms. As how the dropdown menu is explained under the select algorithm section, there is no need to reexplain it.

When a map is selected a link that is used to fetch a map from the backend is generated.

All the maps are stored in the backend. Based on the selected map we generate a path to the selected map. This can be used to call the map where we need it.

```
onChange(event) {
  //takes the selected option and generate the request path
  this.map_link = getRoute(
    this.map_path + this.mapList.data.maps[this.selected]
  );
}
```

8.8. Map

8.8.1. Display map

We wanted be able to interact with the map directly. We wanted to be able to click on it to place objects, and display the paths. An HTML canvas is natively supported way of creating a element that can be changed by code. Unfortunately there is currently no direct support for a canvas alternative in Vue, and therefore we must use HTML and edit the element through JavaScript directly.

We start by declaring the canvas element:

```
<canvas id="c" height="200" width="500" v-on:click="placeMarker">
</canvas>
```

For now the element is empty there is no problem declaring from the start, if the element used by Vue, declaring it before we where ready to input data could cause issues. We will also add more or less everything we need through JavaScript.

We want the map to be displayed right after we select it in the menu. So the first thing is to use the selected maps id and request it from the backend. Create a new image object and set the source to the selected map.

```
var base_image = new Image();
base_image.src = this.map_link;
```

We can now take the width and height of the picture and use it to scale the canvas size. Now with the size of the canvas set correctly we can add the image to the canvas. If we set the canvas too small, parts of the image will not be displayed and if set it to large, the user would be able to interact with the outside of the map. The last part is especially important as a if the user selected a coordinate outside of the map, the backend will return an error.

```
base_image.onload = function () {
  let pictureSize = document.getElementById("c");
  var height = base_image.height;
  var width = base_image.width;
```

```
  pictureSize.width = width;
  pictureSize.height = height;
  map.drawImage(base_image, 0, 0);
};
```

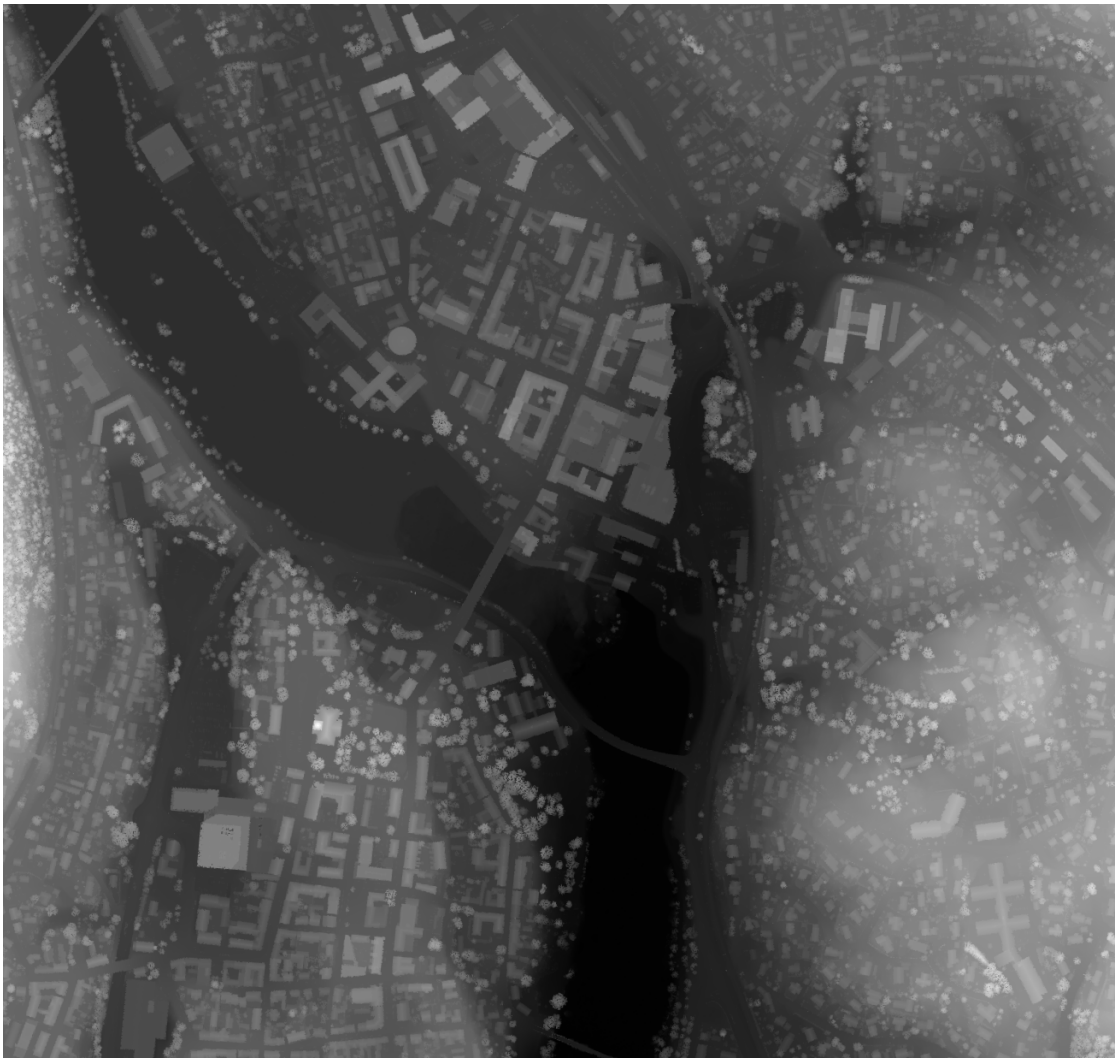


Figure 8.7.: Example of a displayed map

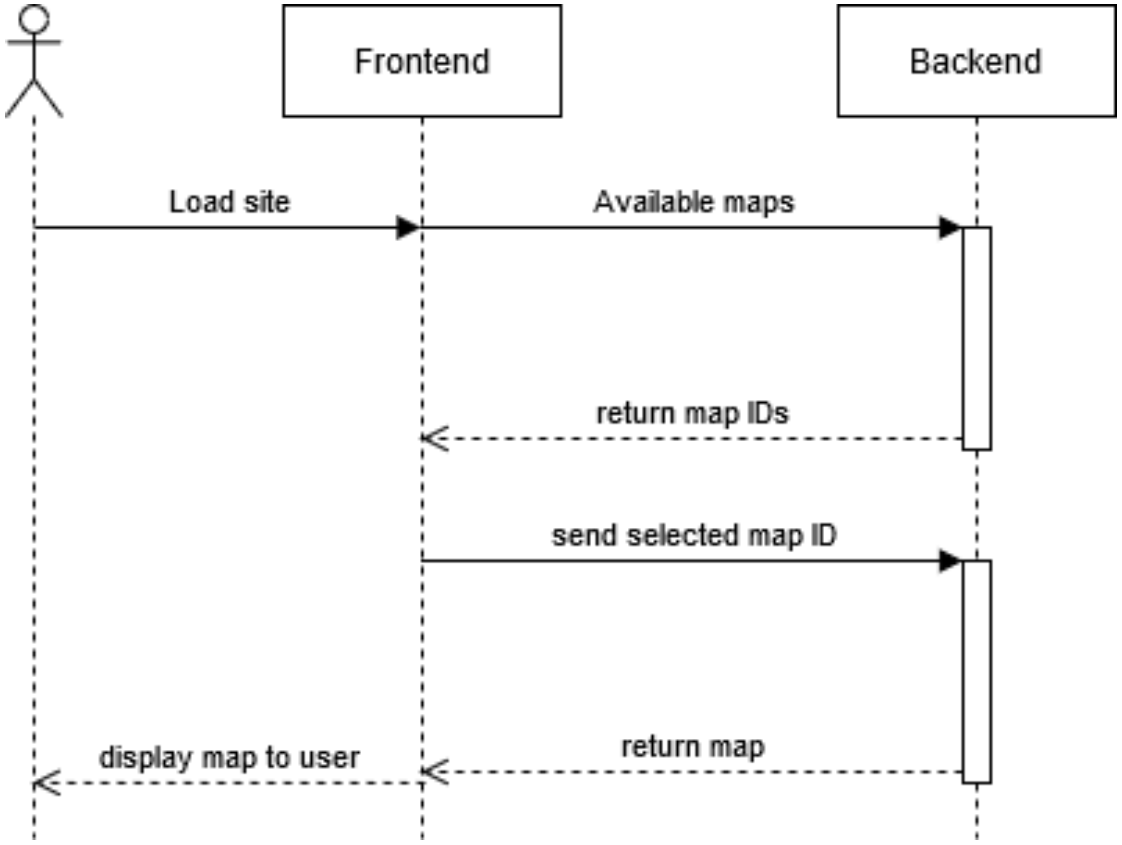


Figure 8.8.: Diagram for every step to display a map

8.8.2. Display returned path

We now have placed the map and have the result from our selected pathfinding module. We can fetch the coordinates from the store and use them. The way the results are given is that each coordinates represent a pixel on the map, with an X-axis pointing down. The origin is in the top-left corner.

```
<div id="draw-cordinates">
<div v-for="(point, index) in recivedCoordinates.points" :key="index">
  <canvas
    width="2"
    height="2"
    v-bind:style="{
      left: recivedCoordinates.points[index].x + mapOffSetX  + 'px',
      top: recivedCoordinates.points[index].y + mapOffSetY + 'px',
      backgroundColor: colour,
      Zindex: 1,
```

```
    }"
  >
  </canvas>
</div>
```

We loop through all returned coordinates, and place red dots at the corresponding location on the map. The red dots are 2 pixels wide and tall, as smaller dots are hard to see. The path is meant to be visual, not to be perfectly accurate. The coordinates themselves should be used for an accurate depiction of the path if that is needed.

To make sure the points are aligned with the map we need the offset of the map. getBoundingClientRect gives the position av the element from the corner of the current window. For the x axis, this is all we need to do. For the y axis we need to take into account the if window has been scrolled. Simply adding the scroll value should be enough. We then update it in the store.

```
var e = document.getElementById("c");
      var rect = e.getBoundingClientRect();
      mutations.setmapOffSetX(rect.x);
      mutations.setmapOffSetY(rect.y + window.scrollY);
```

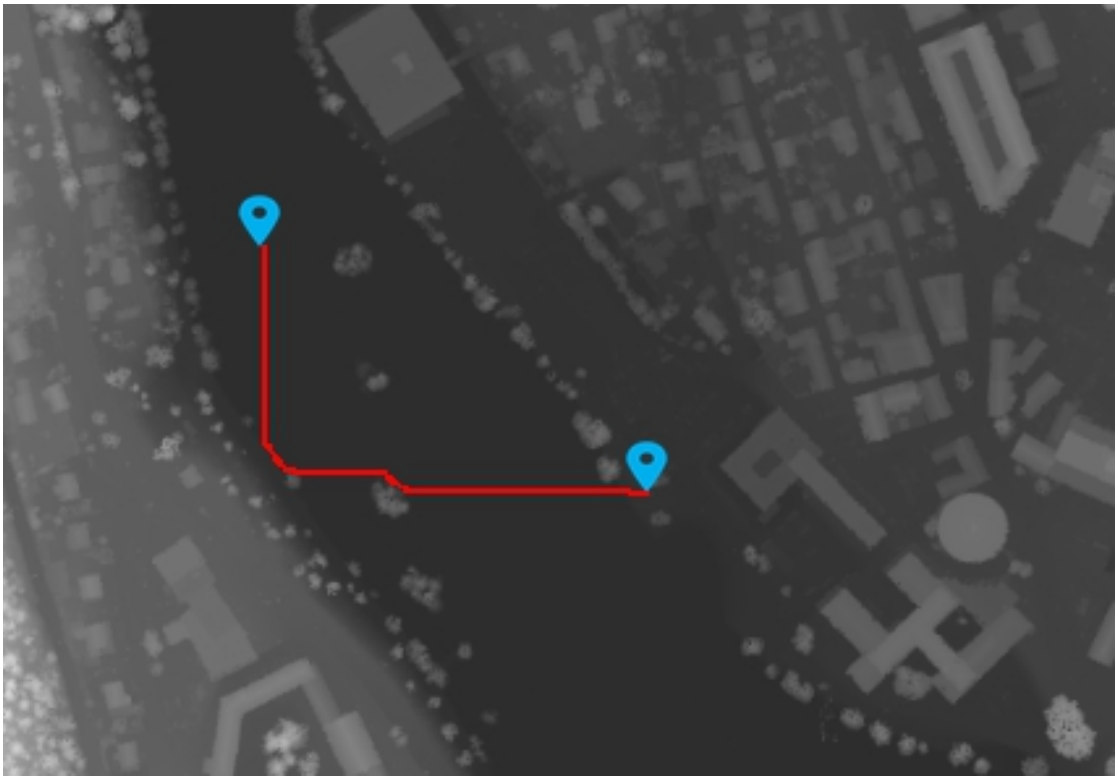


Figure 8.9.: Result of a path

8.8.3. Placing Markers

We wanted our users to be able to select start and stop by clicking on the map. The template element is used to toggle on or off the elements inside it. We only want the marker to appear after it has been placed. The markers position is controlled by using Vue to change the CSS. What we are seeing here is just the visual display showed to the user.

```
<template v-if="displayM1 == true">
  <img
    :src="images.marker1"
    style="width: 1%; height: 5%; position: absolute; z-index=2"
    v-bind:style="{
      left: x1 - 12 + 'px',
      top: y1 - 24 + 'px',
      Zindex: 2,
    }"
  />
</template>
```

The map is inside a canvas, therefore we can add a function that will be called when the canvas element is clicked.

```
<canvas id="c" height="200" width="500" v-on:click="placeMarker">
```

We can get some information about what happen on the click, luckily this includes where on the screen the click was. We subtract the displacement of the map and we can get the coordinates. We also set the render to true, so that the marker actually appears on the map. The selected marker flip between 1 and 0 so that the markers will be placed in turn. When x1 and y1 is updated the, CSS that controls the markers position is also updated. The getPointHeight function is to measure the height at the location the marker was placed. At the end the markers position is updated in the store.

```
if (this.selectedMarker == 0) {
  this.x1 = event.clientX;
  this.y1 = event.clientY + window.scrollY;
  this.selectedMarker = 1;

  this.displayM1 = true;

  let cordX1 = Math.round(event.clientX - rect.x);
  let cordY1 = Math.round(event.clientY - rect.y);
  this.marker1Height =
  Math.round(this.getPointHeight(this.x1, this.y1));
```

```
  mutations.setMarker(cordX1, cordY1, 0);
}
```

8.9. getRoute

Most of the calls made to the backend will look something like this:

```
this.algorithms_arr = await axios.get(getRoute("/algorithms"));
```

The only odd piece of code is the getRoute function. For a while the frontend was developed on a backend hosted locally on the development computer. After we get our server up and running we could host the backend there. The path to a server hosted backend is different than a locally hosted backend. So by running the path through getRoute we can get the correct path.

When we build the frontend it checks if we are running production(locally) or not (server side).

```
  if (env.production) {
    console.log("Running in production mode");
    routeAlias = "./frontend/route_production.js";
  } else {
    console.log("Not using production");
    routeAlias = "./frontend/route.js";
  }
```

If we are running production getRoute it returns the path as-is:

```
export function getRoute(path) {
  return path;
}
```

If we are not running production, all the paths are modified to the server version:

```
export function getRoute(path) {
  return "https://staging.laps.website" + path;
}
```

This makes development of the frontend easier as there is no need to run the backend locally anymore.

8.10. Admin panel

There are certain features that we would like users to have accesses to. Most of these features are already available through the backend. But as the users cannot access the backend, creating a frontend implementation is needed. To make sure these features are not abused, they are all locked behind an admin login.

8.10.1. Upload/delete map

Upload map

First we create a input that excepts tiff files which what our maps uses.

```
<label
    >GeoTiff file:
    <input
      type="file"
      ref="file"
      accept="image/tiff"
      v-on:change="handleFileUpload()"
  /></label>
  <button v-on:click="submit()">Submit</button>
```

When the submit button is pressed the submit function is run. The function creates a form with the uploaded map file, sends it to the backend.

```
submit: async function () {
  if (this.file == null) {
    alert("Please select a file!");
    return;
  }
  let url = getRoute("/map");
  let formData = new FormData();
  formData.append("data", this.file);
  try {
    await axios.post(url, formData, {
      headers: {
        "Content-Type": "multipart/form-data",
      },
      withCredentials: true,
    });
    await this.refreshMaps();
```

```
  } catch (err) {
    console.log(err);
    alert("Failed to upload map: " + err.response);
  }
}
```

Delete map

The maps currently available is presented in a list and in the list a button is generated, when pressed runs the deletemap function with the given map id.

```
<li v-for="map in maps">
  ID: {{ map }} <button v-on:click="deleteMap(map)">Delete</button>
</li>
```

The delete map function sends a delete request to the backend.

```
deleteMap: async function (map) {
    let url = getRoute("/map/" + map);
    try {
      await axios.delete(url, {
        withCredentials: true,
      });
      await this.refreshMaps();
    } catch (err) {
      console.log(err);
      alert("Failed to delete map: " + err.response.data);
    }
  }
```

8.10.2. Upload module

Uploading a module works similar to uploading a map. Rather than looking for tiff file we look for a tape archive.

```
<div id="moduleUploader">
  <h2>Upload new module</h2>
  <label
    >Module tape archive:
    <input
      type="file"
      ref="file"
```

```
      accept="application/x-tar"
      v-on:change="handleFileUpload()" /></label
  ><br />
  <label>Name: <input type="text"
                name="name"
                v-model="input.name" /></label
  ><br />
  <label
    >Version:
    <input type="text" name="version"
    v-model="input.version" /></label
  ><br />
  <button v-on:click="submit()">Submit</button>
</div>
```

When the submit button is pressed the submit function is run. We then check that all fields are filled in. Add the file to a form, and attempt to send it to the backend.

```
submit: async function () {
  if (this.input.name == "" || this.input.version == "") {
    alert("Please input name and version");
    return;
  }
  if (this.file == null) {
    alert("Please select a file!");
    return;
  }

  let url = getRoute("/module");
  let formData = new FormData();
  formData.append("name", this.input.name);
  formData.append("version", this.input.version);

  //Need to set the content-type header on the module field,
  //so recreate the file:
  console.log(this.file);
  let file = new File([this.file.slice()], "module.tar", {
    type: "application/x-tar",
  });
  formData.append("module", file);
  axios
    .post(url, formData, {
      headers: {
        "Content-Type": "multipart/form-data",
```

```
      },
      withCredentials: true,
    })
    .then(function (data) {
      alert("Successfully uploaded module!");
    })
    .catch(function (err) {
      alert("Failed to upload module: " + err.data);
    });
}
```

8.10.3. Running modules

From the admin panel we can see which modules that are running, we can stop them and restart them. In this case restarting also works as the start function. We create a list that displays all available modules.

```
<li v-for="module in modules">
  {{ module.name }} {{ module.version }} State: {{ module.state }},
  <a v-bind:href="moduleRoute(module, 'logs')">Logs</a>
  <button v-on:click="restartModule(module)">Restart</button
  ><button v-on:click="stopModule(module)">Stop</button>
</li>
```

We also create two buttons for each function. They will run either the restart function or stop function, with the associated module. For stop function the only thing we need to do is send a request to the backend with the correct id and the backend will do the rest.

```
stopModule: function (module) {
  let url = this.moduleRoute(module, "stop");
  axios.post(url, { withCredentials: true }).catch(function (err) {
    alert("Failed to stop module: " + err);
  });
}
```

It works more less the same for restarting a module.

```
restartModule: function (module) {
  let url = this.moduleRoute(module, "restart");
  axios.post(url, { withCredentials: true }).catch(function (err) {
    alert("Failed to restart module: " + err);
  });
}
```

8.11. Map data

Map data is in greyscale. The higher the point, the more white is in the pixel.. Therefore we can use this to get some information about the paths that is sent from the modules. The backend also have stored the lowest point in the map and the heights map. So this map for example have its lowest point of 142 meters and high point of 216 meters.

8.11.1. Using Colour to get Height

Because we used a canvas element the present the map, we can get data about what is in the element. This allows us the select a pixel in the canvas and get the colour of that pixel. By doing some math, we can calculate the height in this point. First take the colour code from between 0 to 255, where 0 is the lowest point(black) and highest point is 255(white). We take the point percentage from the highest point, so $p \div 255$ to get the percentage. The maps are *normalised*, so the tallest point on the map always has the value of 255, and the lowest point is always 0. The percentage we found earlier can therefore be used to find the height. Then add back the lowest point.

```
getPointHeight: function (x, y) {
  //get the RBG code from pixel with coords
  //x and y with size 1 by 1 pixel
  let pointRBG = this.mapCanvas.getImageData(x, y, 1, 1);
  //Find the percentage of maximum value
  let precetangeOfHeight = pointRBG.data[0] / 255;
  // multiple the percentage with difference between
  //the highest and lowest point
  let pointHeight = precetangeOfHeight * this.mapTotalHeightDiff;
  //We add back the lowest to point to get the accurate
  // height or the height over the sea
  let pointHeightfromSea = pointHeight + this.mapData.min_height;

  return pointHeightfromSea;
}
```

8.11.2. Useful data

With the ability to find the height of a specific pixel, we can get some useful data out of it.

- Lowest point
- Average height for the map
- Highest point

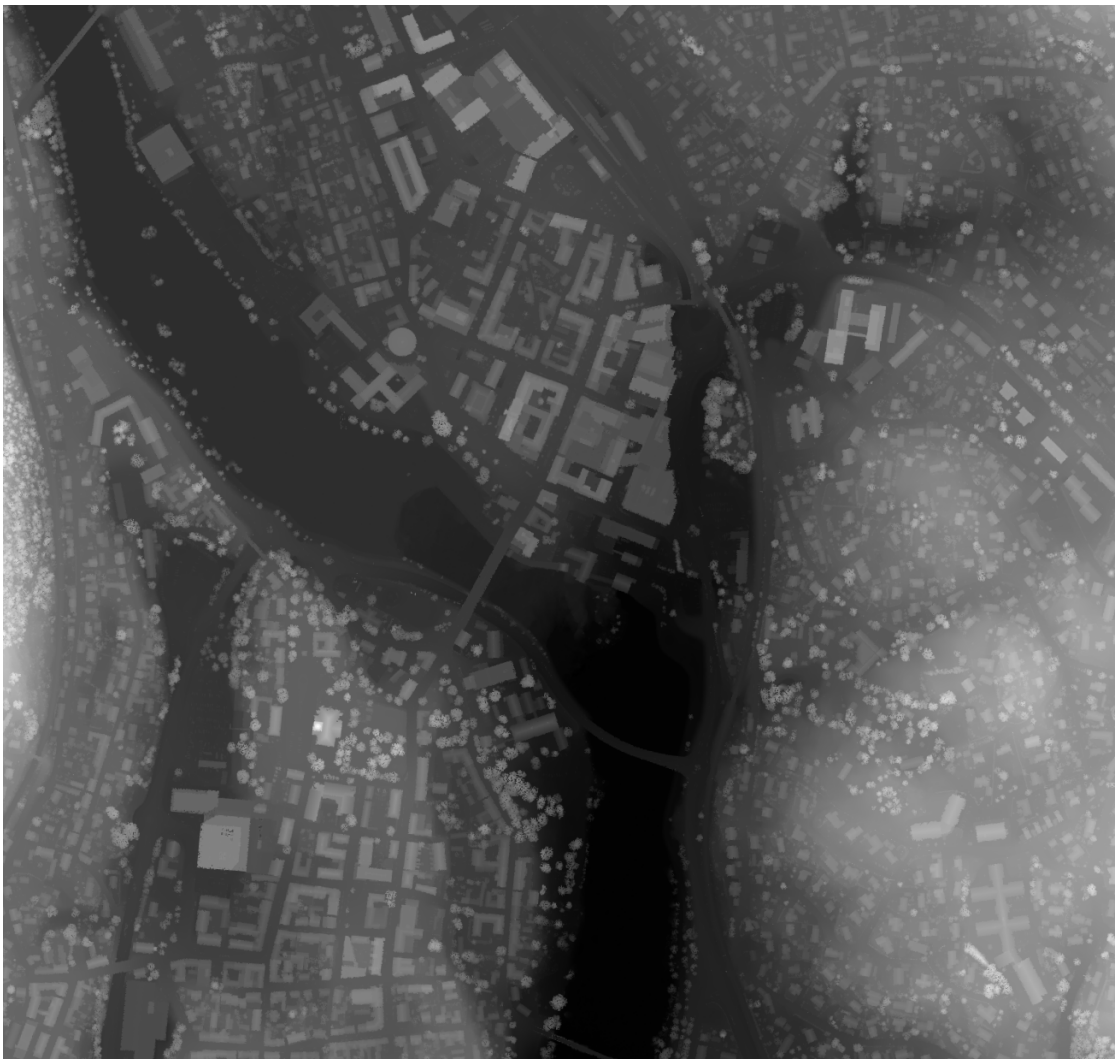


Figure 8.10.: Example of greyscale map

- Height for start and stop point
- Accumulated height difference

The lowest point height and highest point heights has to be retrieved from the backend, but we also get the average height as well. As we already have a function to get the height of a point, and we have the start and stop point, we get the height from the start and stop point easy as well.

We wanted to figure out the accumulated height of the path. There is already a function that finds the height in a specific coordinate. We also have an array of coordinates that make up the path. To get the accumulated height we take the difference in height and the next point and add this to the total accumulated height. After we run through the entire array we get the total accumulated height.

```
accumulatedHeight: function () {
  let height = 0;
  for (let i = 1; i < this.recivedCoordinates.point.length; i++) {
    //find the height of the points
    let point1 = this.getPointHeight(
      store.recivedCoordinates.point[i - 1].x,
      store.recivedCoordinates.point[i - 1].y
    );
    let point2 = this.getPointHeight(
      store.recivedCoordinates.point[i].x,
      store.recivedCoordinates.point[i].y
    );
    //adds the total difference to the
    //accumulated height difference
    height = height + Math.abs(point1 - point2);
  }
  console.log(height);
}
```

As this was one the last functions to be added to the service, the UI element are not done. The functions are finished but haven't yet a way for them to presented.

9. Backend

The backend is the glue that holds the entire application together. Without it, there would be no website, and thus no service. The backend is implemented as an HTTP server, listening to REST API calls. Build instructions and a general overview of the source code can be found in Appendix B. The REST API reference can be found in Appendix F.

9.1. Programming language

We want the backend to be fast, secure and reliable. We should therefore choose a language which will fit all these criteria. Traditionally, developers have chosen to write their applications in C or C++ when performance is important. In our case, backend performance is not a hugely important thing, because the main factor limiting performance is going to be the pathfinding modules.

Security is an important factor for us. We are going to expose the service to the internet, and as such, using a language like C or C++ is a bad idea, due to the lack of memory safety provided by those languages. It is certainly possible to write good, sound code, especially when one utilises the more modern features of C++. However, there is still a risk. According to Microsoft, 70% of security vulnerabilities come from memory issues[22].

As a famous example, consider Heartbleed. Heartbleed was a bug in OpenSSL which allowed clients to read server memory. There was a buffer overflow in the heartbeat handling. The heartbeat would receive data of some specified length, and send it back to the client to verify that the connection was open. All an attacker had to do was to maliciously set the length field to be greater than actual length of the data, and the server would happily send back data that lived past the end of the buffer[23].

We therefore chose to write the backend in Rust. Rust is a systems programming language which enforces memory safety at compile time[24, Chapter 4]. These safety guarantees also include thread-safety, which helps boost productivity, as it is impossible (provided we do not use unsafe and opt out of this safety) to rack up a number of memory-safety related bugs which can be hard to track down. Rust can provide all this with performance comparable to C++, as it is compiled down into optimised machine code.

Rust is also Håkon's preferred programming language, and since he is the main developer of the backend, it was a natural choice.

9.2. Tests

Every REST endpoint has associated unit tests which are run continuously as we develop the backend. Testing of a web application can be quite a task, but because the backend relies so heavily on Redis to manage it's internal state, we can spoof situations. By carefully setting values, we can simulate any state the backend can find itself in. The downside of this is that we cannot run the tests in parallel as they would interfere with each other, but we don't have hundreds or thousands of tests, so it doesn't really matter.

When we implemented pathfinding module uploading as per RQ.4.3, the state issue was made worse than it already was. While Redis allows the selection of multiple, separate databases, the Docker daemon doesn't really have an equivalent. Afterwards, we completely abandoned the idea of parallelising our tests, which is done by default in Rust.

We tried to test one piece of code per test as much as possible. This means that our tests are rather long because we try to test as many situations as possible for each piece of code. For example, we try to extensively test each endpoint when fed both valid and invalid inputs. The result is that most tests are > 100 lines long. However, we have quite good coverage (> 80%)[1]. The main reason it isn't higher is that there is a fair bit of setup code which doesn't get run when we run the tests.

9.3. Pathfinding modules

Pathfinding modules are managed by the backend using Docker[2]. Docker provides a REST API which we tap into using a library called Bollard[3]. We have complete control of the Docker daemon this way. A module consists of a Docker image, which is built from user-supplied files.

9.3.1. Communications

Like we explained in Section 4.4.2, Redis is a great choice for inter-process communications. We use the list approach for the communications link between the backend and the pathfinding modules. Most messages are multiplexed over a single key such that we only have to have one task[4] listening for events at once. The backend implements the architecture we described in Chapter 7.

[1] `https://codecov.io/gh/LAPS-Group/laps/branch/master`
[2] See Section 4.3 for more information about Docker.
[3] url: `https://github.com/fussybeaver/bollard`
[4] «Task» is the term used to describe an asynchronous operation running in the background when working with $n \longrightarrow m$ threading. The first part of Steve Klabnik's talk *Rust's journey to Async/Await*[25] is a great introduction to the topic.

9.3.2. Shutdown and Startup Messages

Our architecture requires modules to register themselves in the system. These come in the form of startup and shutdown messages. We use these to keep track of which modules are available to the user. Essentially, we store it as a Redis set. Each member of the set is one pathfinding module. Using a set allows us to easily determine if a module is active by using the Redis command `SISMEMBER`.

We keep track of how many instances of the same module have started. Whenever we receive a startup message, we increment this number by one. Conversely, we later decrement the number when a module shuts down. When we read a startup message, we add the module to the set of running modules. Using a set helps us here because trying to add a set member which already exists is a no-op.

The counter comes into play when we receive a shutdown message. After decrementing it, we check if it has reached zero. If it has, then we can remove the module from the set of running modules. When this happens, we make sure to cancel any jobs that the module has in it's queue, and clear the cache of any jobs using this module. More on the cache in Section 9.3.5.

It is important to cancel the queued jobs. If we don't, the user will be waiting for their jobs to complete, but they never will. It is much better to return some kind of error message to the user to improve the user experience. We respond to every job in the queue with a cancellation, and then delete the module's work queue. This is done to prevent «zombie jobs» where if the module is restarted, it will calculate jobs which are no longer needed.

Originally, we only had the set. This caused problems when we wanted to run multiple instances of the same module; if one of them were to shut down, the module would appear to be down. With multiple workers, this wouldn't be true as the remaining could still receive jobs. Implementing the counter approach solved this issue.

9.3.3. Module management

In order to achieve F.4[5], in particular RQ.4.2 and 4.3, we need to have some way to manage modules. Management as per these requirements is to upload, start, stop and delete any module independently of one another.

We do not have to explicitly keep track of which modules have been uploaded in a database. Instead, all we have to do is to get a list of Docker images using the API, and use the data in tags to tell us the name and version of each module. A Docker image can be tagged with a name and a version, which we use to tell which module the image refers to.

[5]See Section 10.4

Uploading of New Modules

When a pathfinding module is uploaded, a Docker image is built from the upload. The upload consists of a multipart form with four parts: a name, a version, number of allowed workers (which we will get back to) and a tape archive (`.tar`) containing the source for the module. The name and version fields are used by the backend to tag the image, so we can identify it later.

The archive must contain at least two files: `requirements.txt` and `main.py`. `requirements.txt` is a list of Python packages required to run the module, and they will be installed into the image when it's built. This will be different for every module. `main.py` is the entry point for the module. This is the file that gets executed when the module starts up. The LAPS module running library, `laps.py`[6], is included into the image by default and should not be supplied by the user. Figure 9.1 shows the file system of a minimal module containing only the required files.

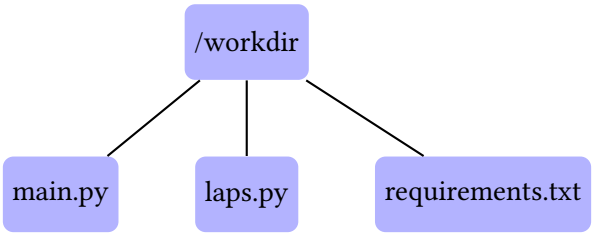


Figure 9.1.: A work directory for a minimal built module image. Note that this figure omits all the operating system components, which are still present, just not relevant.

Removal of Modules

We found that it was necessary to delete modules in some cases. Sometimes one uploads the wrong module by accident, sometimes it has to be modified, etc. To do this, we have to delete any containers which may have been created from the module image, the image itself, and any traces which remain in the database afterwards. This first makes sure that the module being stopped is not currently running.

Starting and Stopping Modules

As we explained in Section 4.3, we have to create a container from a Docker image to actually execute it. This is where the module workers field comes in. When creating the containers, we have to create as many containers from the image as there are workers assigned to the module.

[6]Read more in Section B.5

So that's what we do. We make sure to name our containers in a fixed pattern. One can see a glimpse of this scheme in Figure 9.2. Because container names must be unique, we can't just name each worker container after the module and call it a day. We also include the worker number. The worker number is a number we use to separate the workers from each other. The naming scheme is just the module name, version and worker number, concatenated with hyphens.

Using the names of the containers, we can determine if we have to create the containers or not. If they are already created, we can start them right away. We can end up in this situation by stopping the module at some point, and then deciding to start them up again. Starting them is done through the Docker API.

Figure 9.2 illustrates what this looks like. On the right, we see the images and their tags, illustrating the name and version of the module. This is how one normally uses these tags. When we want to start a module, we create the containers if needed, as shown in Figure 9.3. Note how the name of the container is created from the name and version of the module but separated by a dash instead of a colon. This is because Docker does not support versioned containers, so we roll our own solution.

Note that we use the Docker API to determine what state the module is in: running, failed or stopped. In Section 7.4.1 we explained that modules send a message to the backend when it starts up to register itself with the system. This might seem redundant when we can instead prod Docker to determine the same information. However, this turns out to be quite useful in certain situations.

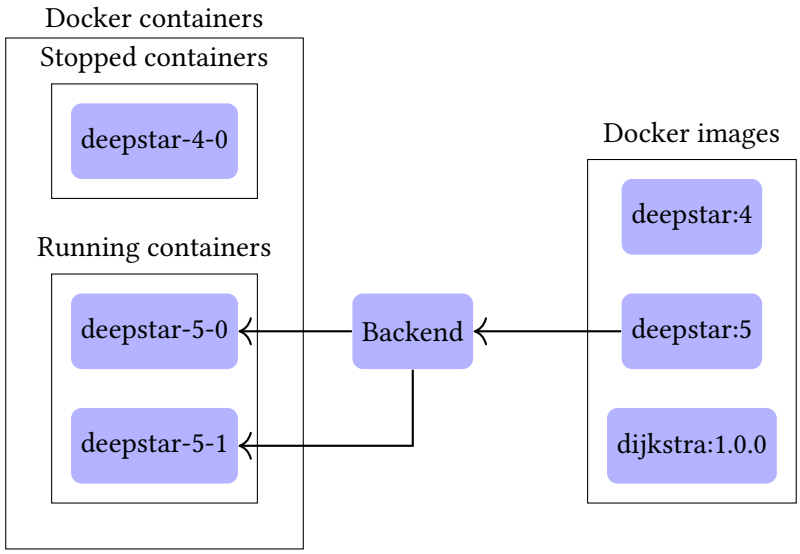


Figure 9.2.: How we create containers from images. Note how no names are capitalised. Docker does not allow us to use uppercase letters in image names, so we have to keep them lowercase.

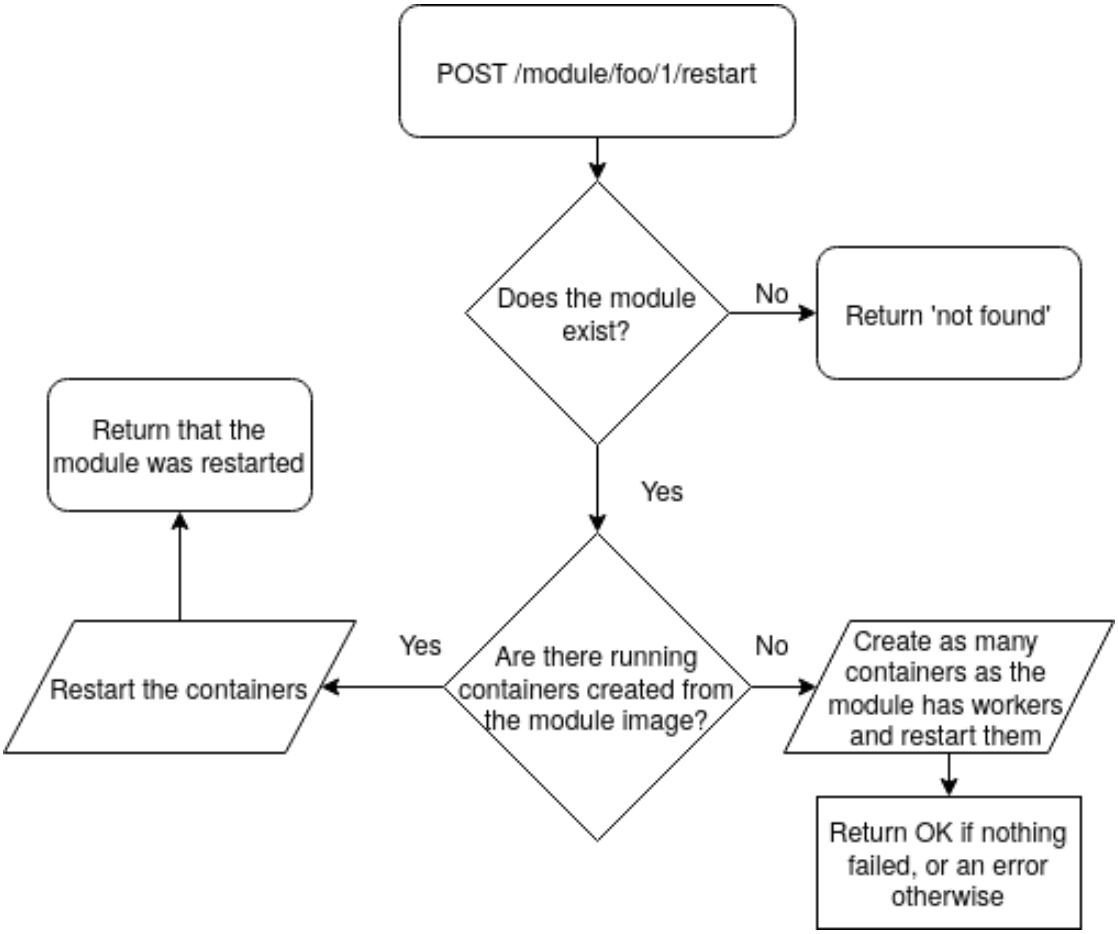


Figure 9.3.: Module startup flow chart

By allowing modules to register themselves from outside the management system, it becomes easier to debug and prototype new modules. We can quickly and easily develop and test our module without having to repackage it, making integration of an algorithm less tedious. It also helps us debug potential backend issues, and is easier to test.

To stop a module, we don't have to do anything special. We just tell Docker to stop each of the containers created for the module.

Getting Module State

A module can be in one of three states: Running, stopped and failed. The states are forwarded to the frontend such that we can get the overview of available pathfinding modules, as per RQ.4.1[7]. For every Docker image, we get every container which is associated with that image.

[7] See Section 10.4

We then investigate each container to determine the state of each module.

Because we are working with several containers, we have to get the state of each worker container of the module. If all the states are the same, we return that the overall state is that state. If there are different states, we build a message string based on the states we retrieve, which can be displayed in the frontend.

If an image has no associated container at all, that means that the module never was started, so we return that the module is stopped. If a container is found to be running, we don't have to do anything and can mark it as running. If however a container has exited, there are two possible reasons:

1. The module was shut down successfully.
2. The module failed with some error.

We can tell if there was an error or not based on the exit status of the container in standard UNIX fashion. A nonzero exit value means that there was a failure. If the exit code is nonzero, we return that the container failed along with the error code.

9.3.4. Logging

Because we are executing arbitrary Python scripts when running a pathfinding module, there are many ways this can fail. To counter this, we had to have some kind of logging infrastructure in place to help the user debug the issue. Docker has builtin log support which would have been a good choice. However, the library we used in the backend to communicate with Docker did not expose this API.

We got around this by making our own logging system. It was not the deciding factor in this decision, because what we really wanted was a logging system which is well-integrated into our system. This also gives us complete control of how we manage the logs, and we use this to put module logs into the server logs as well.

A side effect of this is that the logs are duplicated three times: When we store them in our database, in Docker's stdout logs and in the server logs. The only place we really care about the logs is when they are stored in the database, so we can get rid of the others if this duplication causes lots of disk space.

Like we explained in Section 7.4.4, we receive log messages from the pathfinding modules. Depending on their level, we emit a corresponding log message in the backend as well. Afterwards, we use the information in the message to create a pretty version of the log and store this in a list which is unique to each module. When retrieving the logs, all we have to do is to concatenate each element in the list with newlines in between before we can send it back to the user.

9.3.5. Job submission

In order to submit a job, we communicate with the pathfinding modules through Redis. This is well and good, but it is not something that a user can do themselves for security reasons. Therefore, we expose an API endpoint to submit jobs on their behalf. We take JSON data which describes the job. The exact data is described in Appendix E, but it contains everything we need to execute the job.

When a job is received, we have to do a couple of checks. First, we check if the job is *cached*. More about how the caching works in a bit, but if the job hasn't been seen before, we have to tell a pathfinding module to run the job. First off, we have to make sure that the job is valid. We only cache valid jobs, so we can safely do this *after* the cache check.

A job is considered valid if it fulfils the following requirements:

- The map with the requested ID exists.
- The requested pathfinding module exists.
- All coordinates are positive. Coordinates are based on the pixels in the map image, so negative values are incorrect.
- No coordinates are outside of the map bounds. A map with $n \times m$ pixels cannot contain the position $(0, m + 1)$.

If the job is found to follow these requirements, we can continue. This is where we hand the job over to the module. First, we have to generate a job ID. This ID is used to keep track of which results are for which job. They are generated using a simple counter stored in Redis, which counts up from 1. Job IDs only have to be unique, and if we just count upwards we are guaranteed to have a unique one every time[8].

With the job ID calculated, we can submit the job. We serialise the job to JSON, which is our defined format from the pathfinding modules. Then we push the data onto the queue of the correct pathfinding module, where one of the workers for the module will pick it up.

Next, we generate a random token. This random token is returned to the submitter of the job and is used to get the token back. We can now cache the job submission, more about that in a second. We create a job mapping in Redis which maps this random token to a job id. The main reason for this layer of indirection is that we can tell the difference between a job that has been submitted and is not ready, and a job that has never been submitted at all. If the mapping key does not exist, we know for a fact that the job was never submitted.

Figure 9.4 shows how the mapping is laid out. Notice how there is only ever one mapping which goes to each result. The entire job submission pipeline is illustrated in Figure 9.5.

[8]The counter is a 64-bit signed integer value, so we can submit a job every second for billions of years, and will not run out.

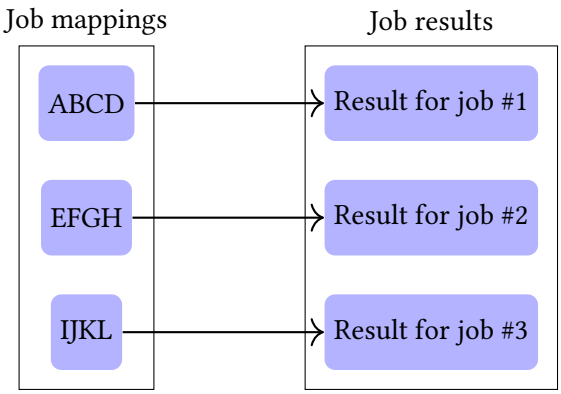


Figure 9.4.: Job mapping. Here one can see the layer of indirection between the mapping and the actual result.

Caching

After generating the mapping of a token to a job ID, we can cache the job. We do this by making a job cache key based on the job's parameters. This means that a job with the exact same parameters will always map to the same cache entry. In the cache entry itself, we store the token which maps to a particular job ID.

This means that job caching is completely transparent to the frontend. If a submitted job hits the cache, we just return the previously generated token for that particular job, just like we would if the job hadn't been completed before.

9.3.6. Getting job results

As stated in Section 7.4.3, the backend receives pathfinding results in one place. To make these results available across the system, we place the job result into a Redis list. The key of the list only depends on the ID of the job which was received. This way, we can see if a job is ready by polling for list elements at the key where the job result ends up. It also means that the list will only ever have one element in it. The list allows us to use blocking operations with Redis.

When the user wants to retrieve the result of a job, all they have to do is to present the job token they received when submitting the job. If a mapping to a job ID exists, we can move on to the next step. If not, we simply return a 404 Not Found because the token has expired or is just invalid. We then look at the job mapping to determine which job ID it maps to. From there, we can wait for a value to be put there. The process is illustrated by Figure 9.6.

The backend will return an error if the job output is an error or a cancellation.

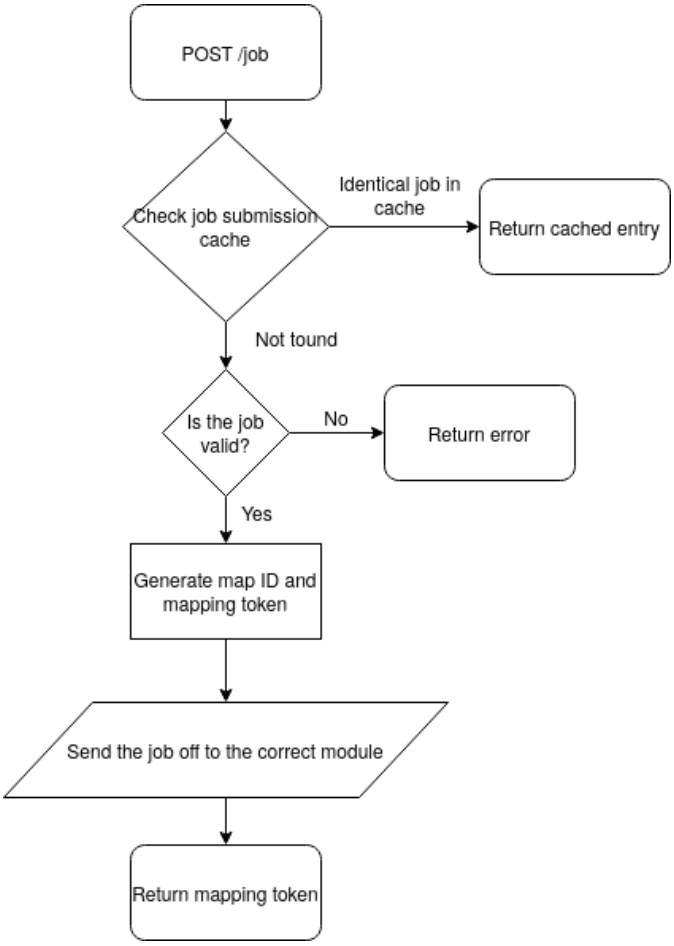


Figure 9.5.: Job submission routine as a flowchart

Long-polling

We do the waiting by using long-polling. Long-polling is a primitive yet effective technique for this kind of thing. The idea is that we perform some blocking operation on the server[9] and respond to the client when the data is ready or a timeout is reached. When the timeout is reached, the client can try to get the resource again if the resource is still desired.

There are better alternatives to long-polling such as a WebSocket. A WebSocket works much like a traditional network socket, but uses HTTP as the transport layer[26]. This makes them a good replacement for long-polling, and can be used for general-purpose two-way communication.

Websockets would be beneficial in this case, however we decided against it as it adds complexity. It is not something that we have experience with from before, which means it might have

[9]Blocking the processing of that particular request at least, not necessarily the whole thread.

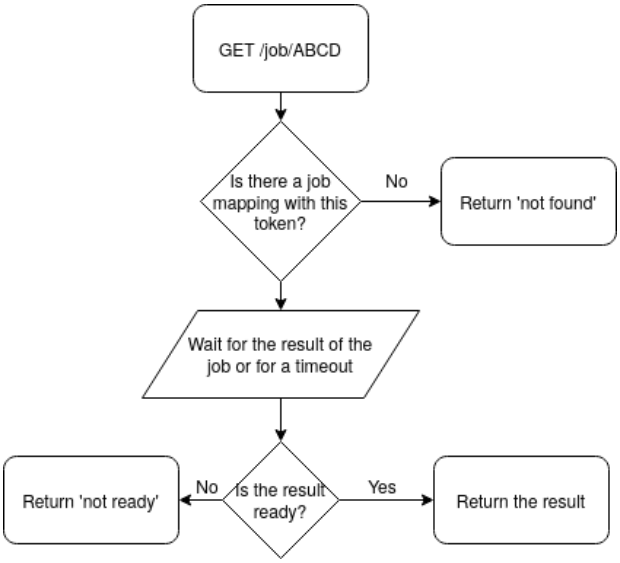


Figure 9.6.: Getting job results as a flowchart

taken up valuable time. It is an area of the service which can be improved upon in the future.

Implementation

We use the Redis BRPOPLPUSH command to poll for a job result. This command takes an element off of the list and pushes it to another list immediately, returning the popped element. By popping and pushing to the same list, we effectively do nothing if the list has only one element. This makes it a good choice for a polling design such as this.

So we perform a blocking list pop operation on the key where jobs come from. If the operation times out, we return and ask the client to poll again. If there is a value, all is well and we can return the result of the job to the user.

Because this operation can take a long time, it will hog the Redis connections we have pooled. This is bad, because if will grind the service to a halt, waiting for the connections to be released. Therefore, we have a separate connection pool specifically for polling jobs. This way, we can hog them for a long time without impacting the rest of the application. If no connections are available, we return a service unavailable error.

We keep track of how many clients are polling using a counter in Redis which is incremented whenever a client is polling, and decremented whenever we are done processing the poll. The maximum number of polling clients can be set using the configuration file.

We initially had a much more complex implementation of this. The reason was that we weren't sure if caching of job results would still work if we kept it as a list. The thinking was that because we empty the list, the expiry of the result would reset. As it turned out, this assumption

was incorrect when we use BRPOPLPUSH because the list does not actually ever get removed. This means that the timeout remains, and the result expires after the allocated time.

Initial, thrown-out implementation

The initial implementation used a normal Redis value with GET. We would see if a job result was available, and if it wasn't, we would sleep for some period of time before trying again, and keep trying in a loop until we either got the result back, or the polling timed out. We could thankfully drop this approach later, reducing server load and making our code cleaner. This also resulted in slightly faster response times.

9.3.7. Map Management

In order to make setup of the service as painless as possible, it was really important to be able to import new maps and delete old ones into the system quite easily. This ties into requirements RQ.4.2 and RQ.4.3.

Import

In order to add new map data, we need to have some kind of conversion routine. To achieve this, we have written a Rust crate (Rust jargon for library), named `laps_convert`. It lives in our repository. It has a couple of functions which can convert from any raster format supported by the GDAL library. GDAL was chosen because it supports a large number of map data formats, and there were ready-made Rust bindings available for it. It parses the file for us such that we can perform processing on it.

These functions in `laps_convert` takes an input raster file and reads the height data contained within them. We also extract some simple metadata, which can be retrieved by any pathfinding module and with the REST API. The height value for each point is given in units above sea level. These units could be anything, depending on the source of the data. We do not touch the unit at all.

When importing the image, we normalise it, such that all height samples are relative to each other. The normalisation algorithm works like this:

- Find the lowest and highest points in the data, H_{min} and H_{max}.
- For every point, use Equations (9.1) and (9.2) to do the conversion $[H_{\text{min}}, H_{\text{max}}] \longrightarrow [0, 255]$ to get a pixel value.
- Export the result to a greyscale PNG.

$$r(A) = A_{max} - A_{min} \tag{9.1}$$

$$h(x, A, B) = (x - n_{\text{min}}) \cdot \frac{r(A)}{r(B)} + B_{\text{min}} \tag{9.2}$$

Equation (9.2) converts a number $x \in A$ to a number $h(x) \in B$.

To do quick-and-dirty conversions between geographical data and data our system can use, we have integrated the `laps_convert` crate into a command-line utility. The utility performs the conversion and can immediately upload it to a running system. The main use for this tool was to upload map data properly before it was added to the REST API. It can still be useful for importing data in bulk.

Deletion

We wanted a way to remove maps from the system, because we found that some maps were way too big. This caused massive memory usage, causing modules to fail and crash. Unfortunately, just deleting the map causes any queued jobs using that map to fail. It's also useful to be able to remove maps which were uploaded by accident.

As it turned out, deleting a map caused chaos in the system. Any jobs in the queue which used the map would immediately fail, taking the modules with them. To counteract this problem, we added a helper function which will fail if the map doesn't exist, and allow the module to skip ahead to the next job if the map does not exist.

To actually perform the deletion, we just delete the map data from both the image hash set and the metadata hash set.

9.4. Administrators

Administrators are the only users able to upload new maps and modules. They are also the only ones who are allowed to start and stop modules. This allows us to manage the service remotely without being logged into the actual server, and be confident that only authorised users have modified the service. We can also use this to see who made what change and when in the server logs.

9.4.1. Passwords

A password is sequence of characters used to authenticate a user account.

Password Hashing

If someone gets access to the database, they can scrape if for passwords and every user's password is compromised. This would allow them to sign in to any other services that the user may have had the same password on. To prevent this, one doesn't store the passwords themselves, but instead a *hashed* version of the password. A hash is an output of a hashing function.

A hashing function is a function which takes a string of bytes as input, and outputs a different string, often of a fixed length. In general, one wants a hashing function without collisions. This means that no two inputs should have the same hash. Furthermore, for password hashing purposes, one wants it to be non-reversible[27].

There are many purpose-built password-specific hashing functions. From 2013 to 2015, the *Password Hashing Competition* was held. It was a competition to design the best possible hashing function for passwords, judged by security experts. The winner of the competition was Argon2[28]. Based on this, we decided to use the Argon2 algorithm for LAPS.

Hash Salting

If several users use the same password, their password hashes will also be the same. If an attacker successfully figures out this password, multiple users can be compromised at once. It is also possible to cross-check with databases obtained from several sources to see who has the same passwords.

We can avoid this problem by *salting* the hash. A salt is just some extra bytes we put at the end of the input data we are hashing. We can store the salt right next to the hash. By using a randomly generated salt, we ensure that each hash is unique, even if the underlying passwords are the same. This means a precalculated hash can no longer be used to determine what the password is.

We salt the hash using an 8-byte salt, generated using a cryptographically secure pseudo-random number generator.

9.4.2. Usernames and Database Storage

Each administrator has their own Redis hash set where we store the data on the administrator. Currently, we store their password hash and whether or not they are a super admin as two separate hash fields.

But what about the usernames? As it turns out, we do not need to store them at all. Instead, the username is encoded in the key used to store the administrator's information. This way we can just generate a key from the their username. If no data exists at the key, we know that the account does not exist. See Figure 9.7 for an example of what the keys look like.

```
laps.backend.admin.admins.jan
laps.backend.admin.admins.karoline
laps.backend.admin.admins.håkon
```

Figure 9.7.: Some example administrator keys. To avoid having case-sensitive usernames, the usernames themselves are always converted to lowercase when creating the key.

9.4.3. Registration

There are two ways to register an administrator: If none have been registered from before, we simply accept the request for what it is, and register whichever administrator as a super admin. We have this mode such that when the service is deployed the first time, we have a way to register the super admin in an easy and painless way. If however; there already exists one or more admins, one can only register an administrator if one is logged in and is a super administrator.

To do the actual registering, we generate the appropriate key based on their username as described earlier. We then generate a random salt for the password hash, and calculate the hash using the Argon2 algorithm using the salt and password. Out comes a string which encodes the hash, the salt, and our hash parameters.

When an administrator is registered, we store the hash settings, hashed password and salt in the database under a single string. When validating a password, this string is parsed to retrieve the settings used to hash it. We can then calculate the hash using the same settings, and if they match, we have successfully authenticated. This also contains the salt of the hash. This also allows us to change the hash settings on registration as much as we'd like without breaking any logins.

9.4.4. Authentication

To log in to the server, a POST request is made to `/login`. This will attempt to authenticate the administrator with the username and password provided. We again generate an admin key based on the username. Then, we get the hash and the super admin status from the database. The provided password is hashed using the hash settings and salt we stored in the database. If there is a match, we successfully authenticated.

9.4.5. Sessions

You don't want to have to send in the password every time a user tries to access restricted content. It would be slow, and needlessly complicate the user interface code. Therefore, we use something called a *session*. A session is a way to keep track of which users have already authenticated.

When a user has authenticated, we create a session object which we can store in the database. This object is used to keep track of which user the session maps to. We could store all kinds of state about the current session in the object if we wanted to, but in our case we only store the username and super admin status of the user.

When the session object tracking our state has been created, we generate a random token. This token maps directly to a session on the server. We write the session token into a private session cookie in the user's browser. The user's browser will then automatically send the session cookie with every request from now on. When a user attempts to do something which requires authentication, we can check if the session token exists and points to a valid session in the database.

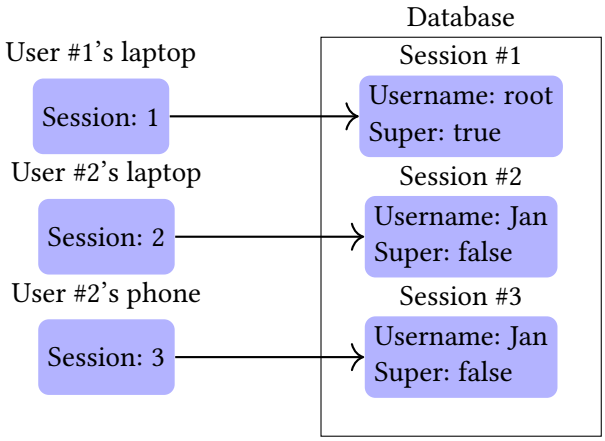


Figure 9.8.: Session example

Figure 9.8 is a visualisation of how sessions work. Notice how a new session was created for user #2 even if they were already logged in elsewhere.

9.4.6. Implementation

Using Rocket's request guards, we can very easily make an endpoint require a valid administrator session. We just add a function parameter of type `AdminSession`. The database lookup is done in the implementation of the `FromRequest` trait for `AdminSession`. The code for this is found in the main repository in `src/web/admin/adminsession.rs`.

When we initially insert the session into the database, we use Redis' expiry feature to automatically expire the session after some time. This forces the user to log back in if they have not used their session for some time. When the session cookie is deleted by the client, there is no way to get the session back. By having an expiry date on the session, we keep our memory clean of unused sessions. To prevent the expiration of sessions which are currently in use, we reset the expiration timer every time a session is used.

If a client connects with a session cookie for an invalid session, we assume this means that the session has simply expired, and delete the cookie ourselves.

9.4.7. Map data

Retrieving the list of maps is simple. As explained in Section 7.7, the data is available in Redis, so all we have to do is run the Redis command `HKEYS laps.mapdata.images`, and serialise the result to JSON. Without prodding at the database manually, it is impossible to have mapdata which consists of only metadata or only the image. This means that we can be confident that only checking one of the keys is enough.

When a map is requested, we just check that the requested ID exists in the database. If it does, we can stream the data directly back to the client.

10. Features and Requirements

A feature is our subdivision of user stories. In this chapter we mention each feature, it's requirements and tests, and explain how we have achieved each one.

10.1. Feature 1

F.1 is about being able to place waypoints. This feature is essential, because it is the main way the user will input which points to calculate a path between. Implementing it is the starting point for the user interface.

F.1	Requirements	RQ.1.1, RQ.1.2, RQ.1.3
	Description	Place start and stop waypoints
	Userstories	U.1

10.1.1. Requirements and Tests

RQ.1.1	User stories	U.1	Tests	T.1.1.1
Done	Category	UI		
Priority	Origin	LAPS		
1	Description	The user must be able to place two markers on a map.		

RQ.1.2	User stories	U.1	Tests	T.1.2.1
Discarded	Category	UI		
Priority	Origin	LAPS		
5	Description	The user must be able to place markers by typing in coordinates.		

RQ.1.3	User stories	U.1	Tests	T.1.3.1-2
Done	Category	UI		
Priority	Origin	LAPS		
2	Description	The user must be able to move and change waypoints.		

T.1.1.1	Requirements	RQ.1.1	Related	
Passing	Description	Test if markers work		
Done	Criteria	Click on the map, and a marker appear. Attempt to click more than the limited amount of marker, and check if you are stopped.		

T.1.2.1	Requirements	RQ.1.2	Related	
Passing	Description	Test if the user can manually input coordinates		
Done	Criteria	Identify the input fields. Insert a random value in all the field. Press the create path button. If the values are invalid an error message is displayed. If the values are valid, no error appears.		

T.1.3.1	Requirements	RQ.1.3	Related	
Untested	Description	Test if markers can be deleted		
Not Done	Criteria	Select a placed marker in the marker menu, press the delete button. Run a test and make sure the marker is not counted		

All of these requirements are describe the user interface. It is important to us that it is intuitive and easy to use, and we believe that the requirements reflect this. We believe that the tests verify that the requirements are met pretty well.

10.1.2. Implementation

10.1.3. RQ 1.1: Place markers on map

This requirement was successful and we where able to generate a path. The explanation of how it works can be found in Section 8.8.2.

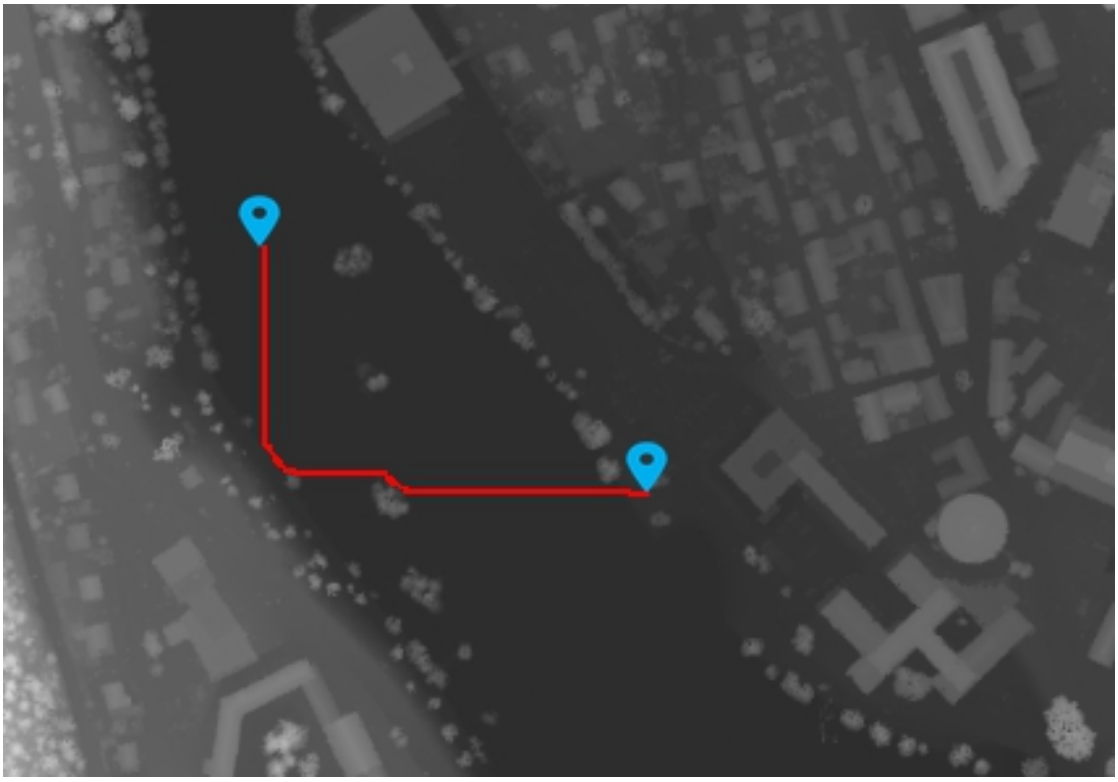


Figure 10.1.: A calculated path with markers

10.1.4. RQ 1.2: Typing in coordinates

This requirement was discarded as most of it functionality is covered wither in rq 1.1 or rq 1.3

10.1.5. RQ 1.3: Move waypoints

This requirements was solved by placing the first marker on the first click, the second marker on the second click, and on the third click move the first marker again and so on. How it was done is hereSection 8.8.3 in the main document.

10.2. Feature 2

F.2 is the main driving force behind LAPS. The pathfinding subsystem is of critical importance for the project to succeed. In order to find these paths, some architecture must be in place. This feature allows the user to actually use the pathfinding subsystem, and select which algorithm to use.

F.2	Requirements	RQ.2.1-2.7
	Description	Path calculation between two waypoints
	Userstories	U.1

10.2.1. Requirements

Requirement RQ.2.1 is by far the most important one. Without it, nobody would be able to find the path between two points. RQ.2.6 specifies that there are multiple pathfinding models to choose from. This is a central part of our application and it is visible in this requirement.

RQ.2.1	User stories	U.1	Tests	T.2.1.1
Done	Category	General		
Priority	Origin	LAPS		
1	Description	The user must be able to find a path between two points.		

RQ.2.2	User stories	U.1	Tests	T.2.2.1
Discarded	Category	UI, Backend		
Priority	Origin	LAPS		
5	Description	The user must be shown an estimated processing time for the algorithm.		

RQ.2.3	User stories	U.1	Tests	T.2.3.1
Discarded	Category	UI, Backend		
Priority	Origin	LAPS		
7	Description	The user must be warned if the estimated processing time exceeds 5 minutes.		

RQ.2.4	User stories	U.1	Tests	T.2.4.1
Not Done	Category	UI		
Priority	Origin	LAPS		
6	Description	The user must be able to export a calculated path as a file.		

RQ.2.5	User stories	U.1	Tests	T.2.5.1
Done	Category	UI, Backend		
Priority	Origin	LAPS		
2	Description	The user must be able to see the available pathfinding algorithms		

RQ.2.6	User stories	U.1	Tests	T.2.6.1
Done	Category	UI, Backend		
Priority	Origin	LAPS		
2	Description	The user must be able to select their desired algorithm.		

RQ.2.7	User stories	U.1	Tests	T.2.7.1
Not Done	Category	Pathfinding		
Priority	Origin	LAPS		
4	Description	The calculated path must be at least one meter above the heightmap.		

10.2.2. Tests

T.2.1.1	Requirements	RQ.2.1	Related	
Passing	Description	Test that the service works with different models		
Done	Criteria	Do a test run in one model, then another, if no problems occur the test is passed		

T.2.1.2	Requirements	RQ.2.1	Related	
Passing	Description	Verify that the backend can properly submit jobs and get their result.		
Done	Criteria	Run the unit test for checking the submit job endpoint and the get result endpoint.		

T.2.1.3	Requirements	RQ.2.1	Related	
Passing	Description	Verify that job submission and polling rate-limiting works.		
Done	Criteria	Run the unit test verifying the rate-limiting.		

T.2.2.1	Requirements	RQ.2.2	Related	
Passing	Description	Test if the service can generate a path between two points		
Done	Criteria	Use a test model that draws a straight line between the two points. Make sure a line is created		

T.2.3.1	Requirements	RQ.2.3	Related	
Untested	Description	Test that user is warned if the calculation time is over 5 minutes		
Not Done	Criteria	Run a test model that will never finish, see if the user is warned		

T.2.4.1	Requirements	RQ.2.4	Related	
Untested	Description	Test if the user can save/export a path made by a model		
Not Done	Criteria	Attempt to export a path and run it on a secondary computer or a new instance of the service		

T.2.5.1	Requirements	RQ.2.5	Related	
Untested	Description	Test that the user can access a description of a model		
Not Done	Criteria	After selecting a model make sure a information panel can be opened to provide necessary information		

T.2.5.2	Requirements	RQ.2.5	Related	
Passing	Description	Verify that the backend can properly list the registered pathfinding algorithms.		
Done	Criteria	Run the unit test for the list algorithms endpoint.		

T.2.6.1	Requirements	RQ.2.6	Related	
Untested	Description	Test if the service shows an estimate of time		
Not Done	Criteria	As time is based upon the module, its only required that a time is shown at all, make sure that a time is present somewhere on the screen after a model as been run		

T.2.7.1	Requirements	RQ.2.7	Related	
Untested	Description	Test that the paths the models make are at least one meter from anything in the height map		
Not Done	Criteria	Generate at least five different paths and manually check that it is at least one meter away from anything from in the height map		

10.2.3. Implementation

RQ.2.1	User stories	U.1	Tests	T.2.1.1
Done	Category	General		
Priority	Origin	LAPS		
1	Description	The user must be able to find a path between two points.		

Requirement RQ.2.1 states that the user must be able to find a path between two points, without specifying any characteristics of the path. Therefore the chosen method for approaching this requirement, was to implement a pathfinding module using the Dijkstra's algorithm. This would allow us to both test returning a path, as well as it would give us the theoretical best paths which we could compare our own algorithms to. This would make testing RQ.11.4 easier, which states that the calculation time of the algorithm must be faster than that of traditional algorithms.

The current implementation of Dijkstra's algorithm we are using is taken from `https://gist.github.com/econchick/4666413` and is an optimised version of Dijkstra for Python. This code was later modified to work with the Section 6.7

To be able to find a path between two points, we use the architecture that we designed. One can read about the architecture in Chapter 7. To find the path between two points, we turn the request into a pathfinding job, and submit the job using the architecture. The user can then retrieve the result once it has been calculated by one of our pathfinding modules. Being a web service, the only way a user can submit jobs is through the API provided by the backend. Our frontend is one implementation of a client.

How the backend handles the submission is somewhat complicated and is detailed in Sections 9.3.5 and 9.3.6.

Frontend

The service uses two waypoints, split into an x and y axis. This gives start x, start y, end x and end y. The frontend automatically converts them into the correct JSON format and sends it to the backend. A function is set to wait for an answer from the backend. An async function is used a it will not finish before a promise is either fulfilled or broken. This allows the function to wait for response, especially for jobs that might take a while. In return a list of coordinates are given on JSON format. The returned coordinates are pushed into the store where they can be accesses by other components. Format on both sent and received messages can be found in the api backend section Appendix E. The library Axios was used to make the http request.

RQ.2.2	User stories	U.1	Tests	T.2.2.1
Discarded	Category	UI, Backend		
Priority	Origin	LAPS		
5	Description	The user must be shown an estimated processing time for the algorithm.		

This requirement was not completed due to lack of time and interest: other requirements were deemed more important.

RQ.2.3	User stories	U.1	Tests	T.2.3.1
Discarded	Category	UI, Backend		
Priority	Origin	LAPS		
7	Description	The user must be warned if the estimated processing time exceeds 5 minutes.		

This requirement depends on RQ.2.2, which was discarded. As a result, this one was too.

RQ.2.4	User stories	U.1	Tests	T.2.4.1
Not Done	Category	UI		
Priority	Origin	LAPS		
6	Description	The user must be able to export a calculated path as a file.		

This requirement was not completed due to lack of time. However, it could all be done in the frontend code.

RQ.2.5	User stories	U.1	Tests	T.2.5.1
Done	Category	UI, Backend		
Priority	Origin	LAPS		
2	Description	The user must be able to see the available pathfinding algorithms		

In order see the available pathfinding algorithms, we have to keep track of which modules have registered themselves. We can then return a list of all the registered modules. As explained

in Section 9.3.2, we store all registered modules in a set. All we have to do from the backend perspective is to retrieve the members of that set and return it as a list.

RQ.2.6	User stories	U.1	Tests	T.2.6.1
Done	Category	UI, Backend		
Priority	Origin	LAPS		
2	Description	The user must be able to select their desired algorithm.		

In order to allow for the selection of algorithms, we had to bake this into the software architecture from the beginning. As one can see in the specification in Appendix E, each module has it's own designated work queue. This makes it easy to send the job to the correct module.

Backend

For the end-user to be able to select which algorithm they want, the REST API had to have some sort of parameter to tell the backend which module to use. Instead of using an ID system like we do for maps, we instead take the name and version of the module as a string. See Appendix F for details.

Frontend

The user should be able to select an algorithm from those currently implemented. The front end does a request to the backend. The format should be in the api section as it might change. In return a list of all available algorithms is returned. The available to be used is presented in a dropdown menu. A for loop runs through all the names in the received list, pushes them to options. For the data to be added to vue reactive element it is important that they are pushed into the array rather than setting its location through index. By pushing the new data, the new data is added the vue reactive element. Only the name and version is used in the dropdown menu, and when the name is selected, it is used to reference in the received data. This way if more elements are added to the algorithms list, it will not break. The selected algorithm is sent to the store where it can be used by other components.

10.3. Feature 3

This feature is about displaying pathfinding results to the user. From U.1 it is clear that this should be done directly in the website, as opposed to having to download it and open it in some external program.

F.3	Requirements	RQ.3.1, RQ.3.2, RQ.3.3, RQ.3.4
	Description	Map display with path
	Userstories	U.1

10.3.1. Requirements

RQ.3.1	User stories	U.1	Tests	T.3.1.1-2
Done	Category	UI		
Priority	Origin	LAPS		
1	Description	The resulting path must be displayed on the map, at least partially.		

RQ.3.2	User stories	U.1	Tests	T.3.2.1
Not Done	Category	UI		
Priority	Origin	LAPS		
5	Description	The website must be well-behaved on mobile devices.		

RQ.3.3	User stories	U.1	Tests	T.3.3.1
Not Done	Category	UI		
Priority	Origin	LAPS		
1	Description	Height data for a calculated path must be shown to the user.		

RQ.3.4	User stories	U.1	Tests	T.3.4.1
Not Done	Category	UI		
Priority	Origin	LAPS		
4	Description	Every feature must be usable with a touch screen.		

Our requirements emphasise how we want to display the results on the map we will implement in F.1. From U.2 we derive that we want the website to designed with mobile devices in mind, which is further reflected in these requirements.

10.3.2. Tests

T.3.1.1	Requirements	RQ.3.1	Related	
Passing	Description	Test that the path is visible inside the map		
Done	Criteria	Generate at least five different paths and visually check that they are inside the map		

T.3.1.2	Requirements	RQ.3.1	Related	
Passing	Description	Verify that the backend can return a list of map data and retrieve the data.		
Done	Criteria	Run the unit test for checking the maps list endpoint and get map endpoint.		

T.3.2.1	Requirements	RQ.3.2	Related	
Untested	Description	Test that the service works on mobile devices		
Not Done	Criteria	Open the service on a mobile device. Test that all menus can be opened. Click various buttons, to assure they are not small and unresponsive. Place two markers on the map, calculate a path.		

T.3.3.1	Requirements	RQ.3.3 done	Related	
Untested	Description	Make sure heights are displayed in the map		
Not Done	Criteria	visually see if height data is displayed. Compare heights with another map the make sure the data is correct		

T.3.4.1	Requirements	RQ.3.4	Related	
Untested	Description	Test the service is usable on a mobile device		
Not Done	Criteria	Open the service on a mobile device. Test that all menus can be opened. Click various buttons, to assure they are not small and unresponsive. Place two markers on the map, calculate a path.		

10.3.3. RQ 3.1: Display Map Path on top of the Map

To display the map, the backend has to provide the frontend with mapdata. This is explained in Section 9.4.7.

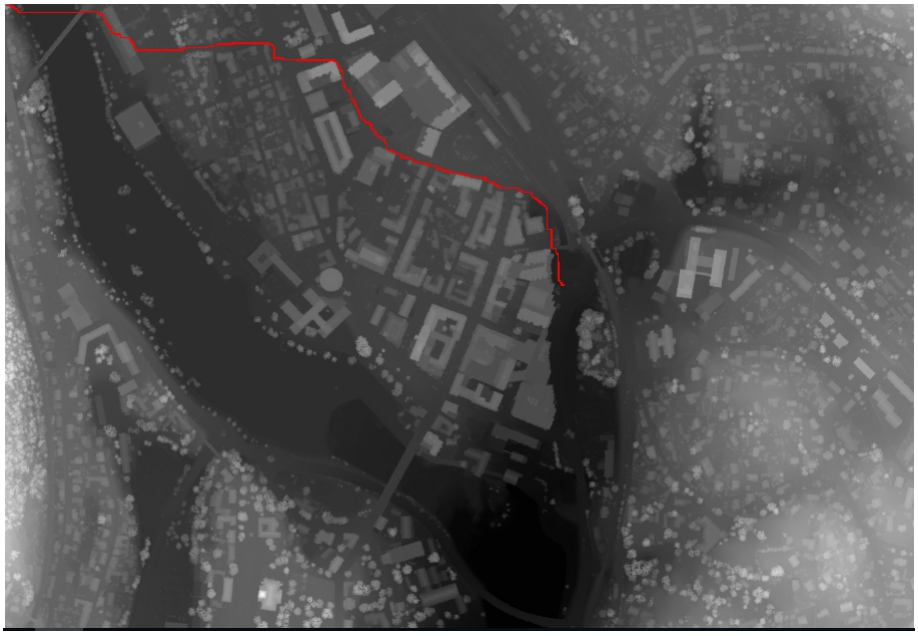
Frontend: Map Display

This components will take an map id display the corresponding map. This component will also automatically call upon its child component, the draw map component. This component will request the coordinates data from the store. It then loops through the given coordinate array, and for each point places a red dot. As the called draw component is called as the child will have the same starting point as where the map is placed on the site.

Frontend: Path Display

Display of the path in the front end. The front end receives an array with points and a corresponding map. For each point in the array a corresponding marker is placed on the map. For now the path is generated by assuming every point to complete the path is given. The map uses pixels of the image as the coordinates. As a single pixel quite difficult to see, each point is instead 2x2 pixels. The map is only meant for visual aid, and if an precise path is needed, the

coordinates themselves should be used.



10.4. Feature 4

This feature is all about administration of the service as a whole, being able to insert new algorithms and map data.

F.4	Requirements	RQ.4.1-RQ.4.6
	Description	Remote access admin panel.
	Userstories	U.2

10.4.1. Requirements

RQ.4.1	User stories	U.2	Tests	
Done	Category	UI, Backend		
Priority	Origin	Jan Dyre Bjerknes		
3	Description	An administrator must be able to see an overview of pathfinding modules.		

RQ.4.2	User stories	U.2	Tests	
Done	Category	UI, backend		
Priority	Origin	Jan Dyre Bjerknes		
3	Description	An admin must be able to start and stop pathfinding modules.		

RQ.4.3	User stories	U.2	Tests	
Done	Category	UI, backend		
Priority	Origin	Jan Dyre Bjerknes		
3	Description	An admin must be able to upload new pathfinding modules, and delete existing ones.		

RQ.4.4	User stories	U.2	Tests	
Not Done	Category	UI, backend		
Priority	Origin	LAPS		
4	Description	The admin must be able to view and change the settings of pathfinding modules, if there are any		

RQ.4.5	User stories	U.2	Tests	
Done	Category	UI, backend		
Priority	Origin	LAPS		
7	Description	A super admin must be able to add other administrators.		

RQ.4.6	User stories	U.2	Tests	
Done	Category	Backend, UI		
Priority	Origin	LAPS		
3	Description	Administrators must be able to add new map data to the system, and delete them.		

RQ.4.7	User stories	U.2	Tests	
Done	Category	Backend, UI		
Priority	Origin	LAPS		
3	Description	Administrators must be able to view module logs.		

10.4.2. Tests

T.4.1.1	Requirements	RQ.4.1	Related	
Passing	Description	Backend unit test testing that modules can be uploaded and are listed correctly.		
Done	Criteria	Log in as an administrator, and use the module upload endpoint to upload the test module. Verify that the list of returned modules contain that module, and that it is not yet running. Upload a module which exits immediately with a failure, and start it. Verify that the list of modules contain both modules where one has failed and the other is stopped.		

T.4.2.1	Requirements	RQ.4.2	Related	
Passing	Description	Backend unit test to verify that modules can be started, restarted and stopped.		
Done	Criteria	Log in as an administrator, and upload the test module with the upload module endpoint. Verify that it is not currently running. Start it with the restart module endpoint and check that the status code is correct and that the module is running. Restart it again, this time check that the status code indicating a restart was returned, and check that the module is still running. Now stop the module with the stop endpoint and verify that it was stopped. Start it back up, and verify that it is running. Finally, stop it twice, and verify that an error is returned when trying to stop it when it isn't running.		

T.4.3.1	Requirements	RQ.4.3	Related	T.4.1.1
Passing	Description	Backend unit test to verify that modules can be deleted		
Done	Criteria	Log in as an administrator, upload the test module and start it. Try to delete it, which should fail. Stop it, and then use the module deletion endpoint to remove the module. Check that there are no containers with the module image, and that the module image is removed. Verify that there are no remnants of the module in the database.		

T.4.5.1	Requirements	RQ.4.5	Related	
Passing	Description	Backend unit test to verify that registration works.		
Done	Criteria	Use the register endpoint to verify that the initial super admin can be registered. Try to register a new admin without signing in, which should fail. Sign in as the super admin, and register a new admin. Verify that the created admin is not a super admin. Try to create another admin with the same username, which should fail. Try to create admins with too long and too short passwords, which should fail. Log in with the non-super admin registered earlier, and try to register an admin, which should fail.		

T.4.6.1	Requirements	RQ.4.6	Related	
Passing	Description	Backend unit test for adding mapdata.		
Done	Criteria	Log in as an administrator. Use the map upload endpoint to try to upload an invalid image, which should fail. Upload a valid map and check that it gets the ID of 1. Upload another and check that the ID is 2. Use the deletion endpoint to delete one of the maps. Verify that no trace of it remains in the database.		

T.4.7.1	Requirements	RQ.4.7	Related	
Passing	Description	Backend unit test for module logs.		
Done	Criteria	Log in as an administrator, upload the test module, and start it. Any module sends a log message when it registers itself. Use the get logs endpoint to check for this message in the module logs.		

10.4.3. RQ.4.1: Pathfinding Module Overview

As explained in Section 9.3.3, we handle pathfinding modules using Docker. Each module has it's own image. To get the overview of pathfinding modules, we simply return a list of all images which currently exist. Each image is tagged with the module name and version. We simply look at this tag to get the name and version out. To every element of this list, we add a state field to show what state the module is in. We get the state of each module as described in Section 9.3.3.

When showing this data in the frontend, we bind the module data to a reactive element which displays the information.

10.4.4. RQ.4.2: Start and Stop Modules

Admins starting and stopping modules was covered in Section 9.3.3.

As shown in Section 11.2.2, the list of modules has buttons on each module entry which calls the restart and stop module API endpoints. The stop button is only shown if the module is running.

10.4.5. RQ.4.3: Uploading and Deleting Pathfinding Modules

The backend exposes an endpoint for uploading a module on `POST /module`, as described in Appendix F. The handling of this is described in Section 9.3.3.

Conversely, there's a deletion endpoint as well. We wrote about this in Section 9.3.3.

In the frontend, each module list entry has a delete button which will make the delete module API call. There is a dedicated uploading panel as seen in Figure 11.5. It builds the correct request to delete a module when clicked, and is only shown if the module is stopped.

10.4.6. RQ.4.5: Registering New Administrators

To add new administrators, we have to create an entry for them in the database. We describe the process in Section 9.4.3. The frontend has an admin registration form shown to the admin if they are a super admin, which can be seen in Figure 11.7

10.4.7. RQ.4.6: Import and Removal of Mapdata

In the REST API, there are endpoints for importing and deleting maps. We have detailed how this is done in Section 9.3.7.

10.4.8. RQ.4.7: Module Logs

We describe how the logging system works in Sections 7.4.4 and 9.3.4.

10.5. Feature 5

F.5	Requirements	RQ.5.1-RQ.5.5
	Description	Admin Authorization.
	Userstories	U.2

10.5.1. Requirements

RQ.5.1	User stories		Tests
Done	Category	UI, Backend	
Priority	Origin	LAPS	
3	Description	Administrators should be authenticated using a password.	

RQ.5.2	User stories		Tests
Done	Category	Backend	
Priority	Origin	LAPS	
4	Description	Password authentication must use a modern cryptographic hashing algorithm	

RQ.5.3	User stories		Tests
Done	Category	Backend	
Priority	Origin	LAPS	
5	Description	The password must be any valid UTF-8 string.	

RQ.5.4	User stories		Tests
Done	Category	Backend, UI	
Priority	Origin	LAPS	
7	Description	The password cannot be longer than 128 bytes.	

RQ.5.5	User stories		Tests	
Done	Category	Backend		
Priority	Origin	LAPS		
3	Description	Only registered administrators are allowed to access administration features.		

10.5.2. Tests

T.5.1.1	Requirements	RQ.5.1	Related	
Passing	Description	Backend unit test to check that login works		
Done	Criteria	Register an administrator. Try to sign in as a user which does not exist, and again with the wrong password, where both should fail. Then log in with the correct password. Verify that we got a session cookie back.		

T.5.4.1	Requirements	RQ.5.4	Related	T.4.5.1
Passing	Description	Backend unit test to verify that the backend denies passwords which are too short or too long.		
Done	Criteria	This test is done in the same unit test as T.4.5.1. Try registering with passwords which are too short and too long, which should fail. Check that a password of correct length is accepted and registers an admin.		

10.5.3. RQ.5.1: Authentication With a Password

In Section 9.4.4 we describe how we use a password scheme to authenticate administrators.

The login page shows up when someone goes to the login page directly, or tries to access the admin panel without being logged in. If logged in, the user is redirected to the admin panel as logging in again is unneeded. It is just a very simple page that sends a login request to the backend.

10.5.4. RQ.5.2: Use a Modern Password Hashing Algorithm

As we show in Section 9.4.1, we decided to use Argon2. Argon2 is an algorithm that was chosen as the best in the Password Hashing Competition[27].

10.5.5. RQ.5.3: Passwords Can be any Valid UTF-8 Sequence

We achieve this requirement by simply using the `String` type from the Rust standard library in the registration form. The `String` type is guaranteed to be valid UTF-8. Therefore, the form will fail to parse if it is not valid UTF-8.

10.5.6. RQ.5.4: Password Cannot be Longer than 128 bytes

To achieve this requirement, we simply set the max password length in the configuration file. We simply refuse a registration request if the password is found to be too long. We also require passwords to be of a configurable minimum length.

10.5.7. RQ.5.5: Authorisation Required for Admin Features

We achieve this requirement by using the session system described in the report. Every administration feature is gated to require an administrator session, which is trivially easy by using a request guard from Rocket.

10.6. Feature 6

F.6 is not quite a feature in and of itself, but it is a way for us to declare requirements for the pathfinding API, which has to be stable and powerful.

F.6	Requirements	RQ.6.1-6.3
	Description	API for pathfinding modules.
	Userstories	U.2

RQ.6.1	User stories	U.1	Tests	T.6.1.1
Done	Category	Backend		
Priority	Origin	LAPS		
1	Description	Height data must be available across the system		

RQ.6.2	User stories	U.1	Tests	
Done	Category	Backend		
Priority	Origin	LAPS		
1	Description	The backend must be able to get paths from pathfinding modules		

RQ.6.3	User stories	U.1	Tests	
Done	Category	Backend		
Priority	Origin	LAPS		
1	Description	Pathfinding modules must get start and end points, as well as map ID, from the system.		

These requirements were all listed in Section 7.1, where we explain some of the thinking behind our software architecture.

T.6.1.1	Requirements	RQ.6.1	Related	
Passing	Description	Test that pathfinding module have access to height data		
Done	Criteria	Pathfinding modules are able to connect to the Redis database, and get map data from the laps.mapdata.image and laps.mapdata.meta keys.		

10.7. Feature 7

F.7	Requirements	RQ.7.1-RQ.7.3
	Description	Select multiple pathfinding algorithms.
	Userstories	U.3

We wanted this feature to allow the user to compare the outputs of pathfinding algorithms side by side. This could have been implemented entirely in the frontend. We had to drop this feature due to a lack of time.

10.7.1. Requirements

RQ.7.1	User stories	U.3	Tests	
Not Done	Category	Backend, UI		
Priority	Origin	Jan Dyre Bjerknes		
5	Description	The user must be able to choose at least 2 pathfinding algorithms, and display the result from these at the same time.		

RQ.7.2	User stories	U.3	Tests	
Not Done	Category	UI		
Priority	Origin	Jan Dyre Bjerknes		
8	Description	The user must be able to see statistics of each path from the different algorithms		

RQ.7.3	User stories	U.3	Tests	
Not Done	Category	UI		
Priority	Origin	LAPS		
8	Description	The user must be able to toggle visibility of each result separately.		

10.8. Feature 8

F.8	Requirements	RQ.8.1-RQ.8.4
	Description	Pathfinding module statistics comparison.
	Userstories	U.3

We wanted to collect statistics on the performance of pathfinding modules over time. We also imagined running a lot of purely random jobs to help build a database of statistics for how fast the module runs and how good each path is on average. We would not have displayed any of this data ourselves, instead opting to export it into a format like CSV which could be imported into databases or spreadsheets for processing.

The original vision for the feature was to run hundreds of jobs in bulk and return the statistics of that run. However, accumulating data over time would have been much better. This is because it would not prevent others from using the service in any way, and the data would be more indicative of the kind of jobs which actually get run.

We decided to not pursue this feature as we ran out of time.

10.8.1. Requirements

RQ.8.1	User stories	U.3	Tests	
Not Done	Category	Backend		
Priority	Origin	LAPS		
8	Description	The service must be able to generate random waypoints and run pathfinding algorithms on these.		

RQ.8.2	User stories	U.3	Tests	
Not Done	Category	Backend, UI		
Priority	Origin	LAPS		
9	Description	The user must be able to choose a sample size.		

RQ.8.3	User stories	U.3	Tests	
Not Done	Category	UI		
Priority	Origin	LAPS		
5	Description	The user must be able to view and compare statistics of run samples between the algorithms.		

RQ.8.4	User stories	U.3	Tests	
Not Done	Category	UI		
Priority	Origin	LAPS		
8	Description	The user must be able to export statistics data.		

10.9. Feature 9

F.9	Requirements	RQ.9.1-RQ.9.4
	Description	Jobs panel.
	Userstories	U.3

Our vision for this feature was a redone queue system which was based on your user. The plan was to add some queue management which would allow users to see a reactive list of jobs which are specific to them. One would be able to see how far along their jobs were in the queue, and cancel them if they wanted to. Administrators would be special and would be able to see every single job, not just for their own user.

By requiring a user to submit jobs, we could limit how many jobs which can be submitted at once. In the current system, this is just not feasible. We don't have a reliable way to tell user apart. There are many approaches to have a crack at this problem, but none are as effective as requiring login. Basing it on users IP addresses would not be the best solution, due to the following reasons:

1. Users behind the same network most likely have the same address.
2. Easily circumvented with the use of a proxy or VPN, which changes the IP the requests are coming from.

After requiring signing in to submit a job, the next step would be to change the backend to have queues specific to users. From there we could have a global queue which give a user a slot to run one of their jobs. When that slot is reached, we could run a job from the user's queue. This way we could be selective about how many slots a user can have in the global queue, with the goal being to more evenly distribute resources between users.

We could then simply give the user a stream of updates about what happens to their queued jobs. They would receive updates when a job is sent to a pathfinding module, when a job is completed, and more. It's hard not to imagine storing a list of their completed jobs as well, acting as an archive of what jobs have been completed before. We would probably use a websocket to implement this, which is like a traditional network socket, but is built upon HTTP[26] for this. It would allow the client to act as a passive subscriber, rather than as an active consumer. This would make the implementation simpler, and save resources on the server.

This is another feature which we had to abandon due to a lack of time. It would probably have been the feature which would have required the most changes overall to the system, and therefore be the most time-consuming of them all.

10.9.1. Requirements

RQ.9.1	User stories	U.1, U.3	Tests	
Not Done	Category	Backend		
Priority	Origin	LAPS		
4	Description	The back-end must be able to queue tasks.		

RQ.9.2	User stories	U.1, U.3	Tests	
Not Done	Category	UI, Backend		
Priority	Origin	LAPS		
5	Description	The user must be able to view status of tasks which they have in the queue.		

RQ.9.3	User stories	U.1, U.2, U.3	Tests	
Not Done	Category	UI, backend		
Priority	Origin	LAPS		
5	Description	An administrator must be able to view the status of each task.		

RQ.9.4	User stories	U.1, U.3	Tests	
Not Done	Category	UI, backend		
Priority	Origin	LAPS		
5	Description	The user must be able to cancel queued tasks.		

10.10. Feature 10

F.10	Requirements	RQ.10.1-RQ.10.2
	Description	Valid path checking.
	Userstories	U.3

The idea behind this feature was to help an algorithm developer by letting the user know whether a returned path is actually valid. Exactly how it would look and function was never decided. The most important thing to have done when implementing this would be to set some criteria that any valid path must have.

This is another feature which was abandoned due to lack of time.

10.10.1. Requirements

RQ.10.1	User stories		Tests	
Not Done	Category	Backend		
Priority	Origin	LAPS		
3	Description	The service must check if the path is flyable.		

RQ.10.2	User stories		Tests	
Not Done	Category	UI		
Priority	Origin	LAPS		
4	Description	The UI must highlight the unflyable parts of a path if it is unflyable.		

10.11. Feature 11

F.11	Requirements	RQ.11.1-RQ.11.4
	Description	Machine learning-based pathfinding
	Userstories	U.4

10.11.1. Requirements

RQ.11.1	User stories	U.4	Tests	
Not Done	Category	Backend		
Priority	Origin	LAPS		
6	Description	The computing time must vary less than 10%.		

RQ.11.2	User stories	U.4	Tests	
Not Done	Category	Backend		
Priority	Origin	LAPS		
6	Description	The computing time must not grow exponentially with grid size		

RQ.11.3	User stories	U.4	Tests	
Done	Category	Backend		
Priority	Origin	LAPS		
6	Description	Computing time cannot vary based on map complexity		

RQ.11.4	User stories	U.4	Tests	
Not Done	Category	Backend		
Priority	Origin	LAPS		
6	Description	The calculation time for finding a path must be faster than that of traditional algorithms		

10.12. Feature 12

Feature 12 is the investigation of using Reinforcement Learning to perform pathfinding. The turn short Python module is also covered in this section, as it was utilized by this feature, as well as Feature 13.

10.12.1. Requirements

RQ.12.1	User stories	U.5	Tests	T.12.1.1-2
Not Done	Category	Pathfinding		
Priority	Origin	LAPS		
2	Description	Return the resulting path as a list of points.		

RQ.12.2	User stories	U.5	Tests	T.12.2.1
Not Done	Category	Pathfinding		
Priority	Origin	LAPS		
2	Description	Calculate path using deep Q-learning, with states as possible movement.		

RQ.12.3	User stories	U.5	Tests	
Not Done	Category	Pathfinding		
Priority	Origin	LAPS		
2	Description	The Machine Learning model shall be in ONNX format.		

RQ.12.4	User stories	U.5	Tests	
Not Done	Category	Pathfinding		
Priority	Origin	LAPS		
3	Description	Path takes consideration to drone stats.		

RQ.12.5	User stories	U.5	Tests	
Not Done	Category	Pathfinding		
Priority	Origin	LAPS		
2	Description	The module must be API compatible.		

The requirements are similar to that of Feature 13, as they're both concerning machine learning investigations, so they're similar in nature.

10.12.2. Risks

RI.3.1			Related		
	Category	Machine learning			
	Description	Training server is not powerful enough to train our models	Impact	Prob.	Risk
	Mitigation	Make sure we have adequate time to train models, and consider acquiring more powerful server, or reduce complexity of models.	3	2	6

			Related		
RI.3.3	Category	Machine Learning			
	Description	The model does not converge on a solution to the problem.	Impact	Prob.	Risk
	Mitigation	Simplify problem domain.	3	3	9

These two risks were perhaps the most prominent ones. The biggest issue with these risks and mitigating them, is that it is extremely hard to predict or have any idea how details of the models, the training data, and the parameters will affect the training.

For the first risk; simplifying the neural network model will reduce the training time it takes for it to converge, as the expense of reducing the accuracy it will achieve. So in some scenarios this is a suitable mitigation, but the reduced accuracy can result in the converged model being insignificant.

Similarly with the latter risk. In some scenarios it is a suitable mitigation, but in scenarios where the problem has been simplified, but gradually increasing the complexity causes it to not converge, simplifying the problem is not a valid mitigation.

These risks where encountered and discussed in the Reinforcement Learning sections of the Technical Pathfinding chapter in the report.

			Related		
RI.3.5	Category	Machine Learning			
	Description	Problem is too complex to solve with our limited experience and time	Impact	Prob.	Risk
	Mitigation	Focus on simple representations of the problem that give meaningful results, instead of trying to focus on solving the problem immidiately.	5	3	15

Focusing on simple representations of the problem and slowly increasing the complexity was done from the beginning, and is discussed in the Technical Pathfinding.

			Related		
RI.3.6	Category	Machine Learning			
	Description	Model overfitting	Impact	Prob.	Risk
	Mitigation	Do frequent error function tests on validation data, and stop training if it gets worse.	3	4	12

A paper released in 2018 showed that overfitting is an issue with reinforcement learning as well, and not just supervised learning[29]. Therefore it is important to validate the trained models on validation data, separate from the training data, prior to using the model in production.

10.12.3. Reinforcement Learning General

Reinforcement Learning (RL) problems are divided into two main parts; an environment, and an agent trying to learn the problem.

This feature documentation will mainly focus on the environments and how they are implemented. The learning algorithms and agents are discussed step by step in the report.

10.12.4. Environments

The environment is the code that runs the problem, or some sort of representation of the problem. It is responsible for keeping track of the environment's state as well as handling the selected action and its effect on the state. Because the selected language for the machine learning is Python, the same language is used for the environments.

OpenAI's python module called gym, is used for developing the environments. It aims at standardising Reinforcement Learning environments, so that they are easier to use, find references for, and to implement. They provide some example environments such as Atari games, as well as a collection of physics based ones[30].

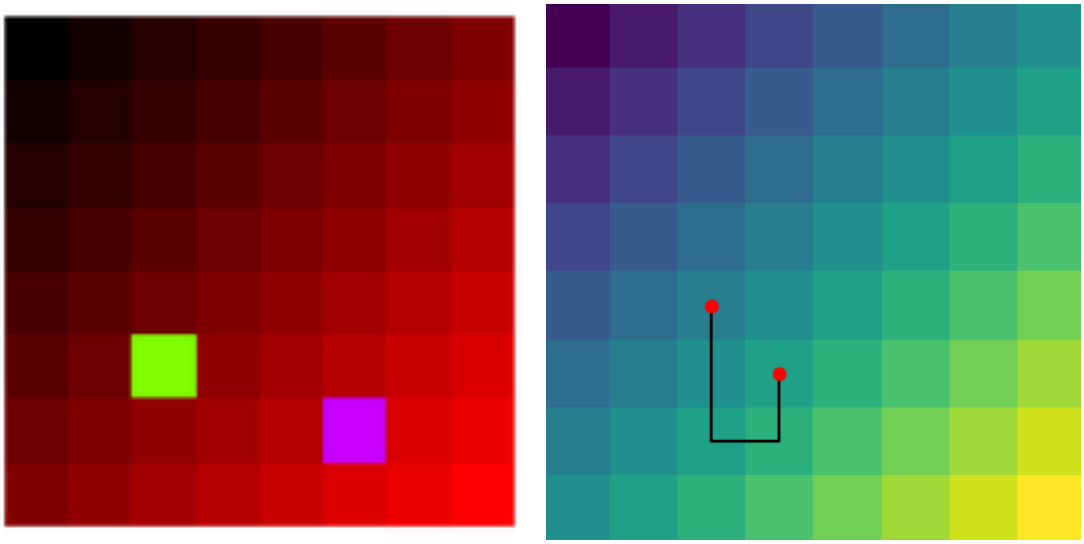
The structure of an OpenAI environment consists of a class, that contains the following methods:

- **Init** - initialise the environment and members of the environment class as needed.
- **Reset** - reset the environment, and return the new state of the environment.
- **Step** - perform the action as given by the argument, and return the new state of the environment, the reward of the action, a boolean for if the episode is over or not, and optional information which is not to be used by the agent.
- **Render** - create a visualisation of the environment in the mode indicated by the argument. What modes are supported depends on the implementation of the environment.

Both the Reset and Render methods were implemented in the Notebooks as well, because they were often modified which takes long time to install after every time you make a modification. In addition, the render methods use PyTorch for putting the images on the GPU, which could not / should not be done in a package.

When trying to use the environment, it is essential for the implementation of the agent to be able to know what it is expected to give as an action, as well as how the environment will be returning the state. Therefore OpenAI implements a module named spaces. Spaces is to be used in the Init function to declare both the shape of the states by defining self.observation_space, and similarly the shape of the actions with self.action_space.

Environments are implemented in the LAPS-Group/gym-drone GitHub repository, and are structured as a Python package, which means they can be installed using Python's package



(a) Notebook (b) rgb_array

Figure 10.2.: Render modes of environments

installer pip. Once installed they can be imported, and initialised using the gym environments make function.

There are three different environments in the module:

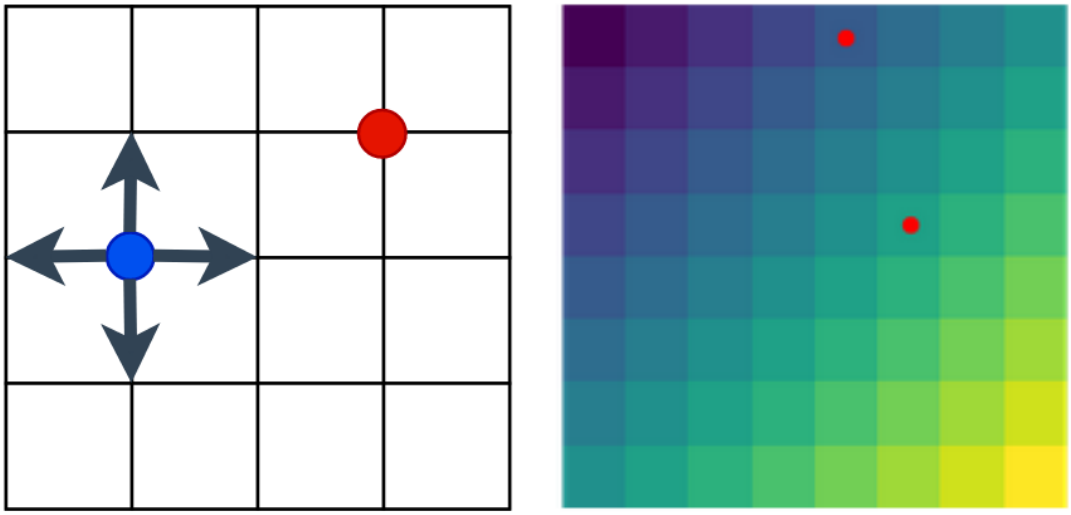
- CardinalDirectionsEnv
- TurnShortV0/V1

Each environment has their own behaviour and observation space. They all have a number of optional parameters, and some mandatory, to the Init function implemented via Python kwargs. These are listed and explained in the environment's individual explanation.

To initialise the gym environment, the make function from the OpenAI gym module is used. It takes as argument the name of the environment type to be initialised, as well as kwargs for any parameter of the environment to be set.

10.12.5. Cardinal Directions Environment

The initial simplification of the problem chosen to investigate was one which the actions are simply moving in cardinal directions on a grid one node at a time. Start position as well as goal position is selected randomly upon restart, and the environment is considered done when the goal is reached.



(a) Movement in the cardinal directions environment

(b) The static map, with dark blue low value, and yellow high

Figure 10.3.: The cardinal directions environment

This environment has a discrete action space with four possible values, which are zero through four, with 0 being right, 1 up, 2 left, 3 down. In other words the agent can choose between the numbers 0 through 3, when selecting an action.

$$(0 \dots 3) \tag{10.1}$$

The observation space is a tuple of four discrete values, first and second being x and y of the drone, and similarly third and fourth the x and y of the goal position:

$$(drone_x, drone_y, goal_x, goal_y) \tag{10.2}$$

This is returned as a tuple when the Step function is called, or the Reset function. Additionally the _get_obs() (get observation) function serves to offer a way of getting this information, without affecting the state of the environment.

The reward at each step is the negative altitude at each point, with the exception of the goal position where the reward is zero. Each episode is only done when the drone reaches the goal position, and trying to move outside the grid moves the drone back to the position it was before it tried to move outside.

The static map used for the height values is upper left cell having zero in height, and all other cells has height equal to the lattice distance from this cell. This static heightmap is visualised in Figure 10.3b.

Init options, passable to the OpenAI gym.make function used to initialise the environment, are:

- rows / columns - the number of rows and columns can be specified. The heightmap will be initialised in the shape specified by these two parameters. If no argument given, the default value of both is 8.
- memory_capacity - number of visited points to keep track of. This is used to display the path in the notebook render, as shown in Figure 10.2a.
- ax - a matplotlib plot object, which is where the notebook renders will be plotted to. This is to allow the render to be shown in a subplot, instead of forcefully taking up the entire plotting area.

10.12.6. TurnShortV0/V1 Environments

The turn short environment was made to explore an alternative approach to the drone path problem, with much of the path being decided by another algorithm, instead of the machine learning deciding all of it. It is dependent on the turn_short Python Module, which is a part of the same package the gym_drone module comes in.

General overview of this environment is that it is an environment using the turn short algorithm to create a path from a Start point to a Goal point, by having the action at each step placing a waypoint.

Difference between V0 and V1, is that V0 can only handle one turn. V1 can handle one turn, but has an Init option allowing amount of turns to be specified. Because of this, V0 is deprecated in favour of V1. Because of this, V1 will be described here.

Init and Reset

Available Init options for this environment are:

- training_data_dir (required) - the path leading to the directory containing training data, in the form of PNG heightmaps.
- training_samples - amount of training samples to be loaded in. Default is 0, which will load every image in the folder.
- shape - row/column shape of the heightmap grids.
- subsampling - how many chunks to divide width/height into.
- steps - number of steps, each step is one waypoint to be placed. 4 for example means placing four waypoints, resulting in four turns.

- turn rate - turn rate of the drone. This determines how sharp the turns are done. This is discussed again in the Section 6.7.
- ax - a matplotlib plot object, which is where the notebook renders will be plotted to. This is to allow the render to be shown in a subplot, instead of forcefully taking up the entire plotting area.

This environment uses heightmaps as opposed to a static map. Therefore it will upon initialisation of the environment load the amount of training data as specified by *training_samples*, from the *training_data_dir*. For this, it uses the Python Imaging Library Pillow.

The function *_load_training_data* is the function responsible for this loading. It initialises the instance variable (member) *_training_data*, which is a numpy array to contain all the heightmaps loaded from the data directory. The array is three dimensional: first index specifying which image, second and third the row and the column of that image respectively.

```
self._training_data = np.zeros(
                (number, rows, columns), dtype=np.uint8)
```

Listing 7: Initialising the empty array, which is to contain all the height data.

In case of a grey scale image, there is only one channel the height data can be in. For RGB images, it is assumed that the height data is stored in the red channel, and the other channels are ignored.

Upon resetting, a random start and stop position is picked, as well as a new height map is loaded. Because choosing a path when the two points are right next to each other is rather meaningless, the selected points are selected randomly until they have a Manhattan distance of at least 10.

Heightmap loading is done by selecting a random image from the previously mentioned *_training_data*, and passed to the *get_heightmap* function. If the training data resolution is larger than the grid size provided by *shape* (the Init option), *get_heightmap* will return a crop from the heightmap the correct size, from a random position in the image.

Actions and Observations

Observation space for this environment, is similar to that of the CardinalDirectionsEnv, with drone positions as well as a goal position. But because it is necessary for the agent to know which point was the previous, as that will affect the next chosen turn, this is stated as well.

$$(last_x, last_y, drone_x, drone_y, goal_x, goal_y) \tag{10.3}$$

It was intended to make the heightmap a part of the observation space as well, as that is necessary information for the agent to know when making its decision. But because only convolutional neural networks were used and lack of time, this information is instead only conveyed through the rendering instead. This way the agent will still have access to it.

The actions available at each step, is selecting which chunk the next waypoint should lie in. Therefore the action space is simply one discrete number, up to the final chunk index. As the number chunks to divided the width and height into is defined by the subsampling, the number of chunks total will be this square. Subtract one as its an index.

$$(0 \dots subsampling^2 - 1) \tag{10.4}$$

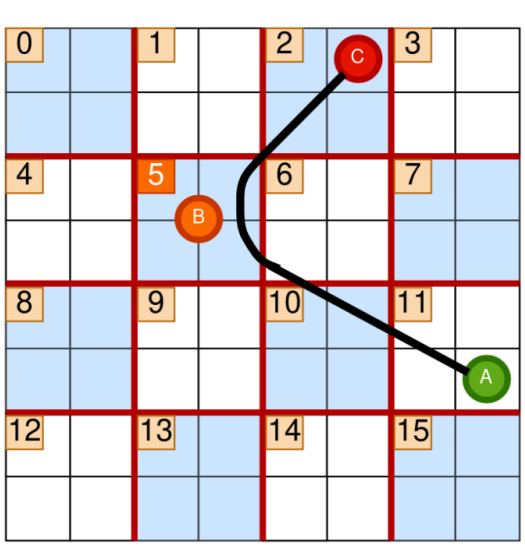
How big a chunk is is defined by the *subsampling* Init option, which allows for increasing the chunk sizes so that there will be less chunks, and therefore less actions to learn. Notice in Figure 10.4 that when selecting chunks, the middle point of the chunk is used. Whereas with the Start and Goal point, they lie in centre of grid cells.

When waypoints to be placed is only one, the first and final points of the turn is decided. The action selected will only place the turning point, as shown in Figure 10.4a, where B is the turning point.

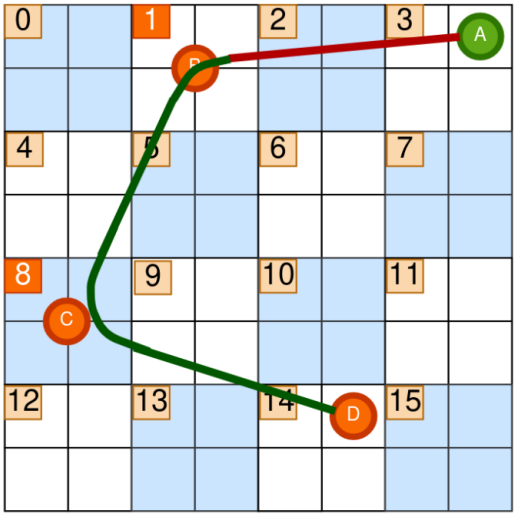
In the case that there are two or more points to be placed, the behaviour is a bit different. Looking at Figure 10.4b, when placing one point at a time:

- In the first step, A is the start point, and B is placed. As there is only two points, the reward will only take into account the height of the straight line formed.
- Step two, A and B are the last two waypoints placed, and C is to be placed next. But because there are only two steps, it is forced to consider D as the end point so that the path ends properly. Therefore the height considered is the BCD turn.

Reward after selecting an action will be the sampled height of the path that was decided upon that step.



(a) 8x8 grid, divided into 4x4 chunk grid. This is a 1 step path, and action 5 is selected.



(b) 2 step path, with first action 1 then 8 selected. For first step, the green line height is used for reward. For the second, the red.

Figure 10.4.: Turn short actions, and their paths, visualized

10.13. Feature 13

Feature 13 is pathfinding using a convolutional neural network. Because of a convolutional neural networks ability to extract features we think CNNs are a good candidate to solve the pathfinding problem.

Because of convolutional neural networks ability to extract features from images we think they are a good fit for extracting information from a heightmap. There are a multitude of ways to generate a path with a neural network but we decided on choosing a midpoint a then running it recursively to generate a path of sufficient detail. What gives convolutional neural networks the ability to extract features from an image is their use of convolution. Convolution works by combining multiple pixels into one pixel by weighted summing. This has been proven with networks such as alexNet, and currently all leading image classification algorithms uses CNN.

10.13.1. Requirements

RQ.13.1	User stories	U.5	Tests	
Done	Category	Pathfinding		
Priority	Origin	LAPS		
2	Description	Take a heightmap, start and stop as an input.		

To make sure there is some level of compatibility between versions and to guide us in development there is an requirement for both the input and the output of the algorithm. The input and output requirements come from features and our job as the developers is to create an internal structure for the algorithm for fulfil all these requirements.

RQ.13.2	User stories	U.5	Tests	
Done	Category	Pathfinding		
Priority	Origin	LAPS		
2	Description	Approximate the midpoint of the most efficient path, using convolutional neural networks.		

As there is a multitude of ways for the network to output a path one has to be chosen. This requirement specifies that the network will use the midpoint method. Here the network will always attempt to predict the median point of the optimal path between two points.

As the optimal path is hard to define there is no requirement for what the optimal path is, this is to keep the investigation flexible. So in our case a optimal path refers to the lowest cost path with the restrictions outlined in Chapter 6.

10.13.2. Docker

All the machine learning and pathfinding is done in Python via a Jupyter Notebook, which is hosted using Docker. This development environment is documented in the «Machine Learning Development Environment»Section 6.1.

10.13.3. Implementation

For the input of the network we decided to give it greyscale images of size 256×256 with the value at each pixel being the height for that pixel. In addition the network was fed the start and stop point for each endpoint. The positions was given in the form of relative values in the range $[0, 1]$, where $(0, 0)$ was at pixel $(0, 0)$ and $(1, 1)$ was at pixel $(256, 256)$

The output of the network comes in the form of a label. Each position that the network can choose is assigned a label, so for a 256×256 image there would be 65536 labels. To reduce the labels and simplify the network the amount of labels was downscaled to 128×128 chunks, this gave the network 16384 labels.

Because convolutional neural networks is a supervised machine learning algorithm we had to generate the training data. To generate training data Dijkstra was used to find the optimal path between the start and stop point and the midpoint of the was extracted. This was then stored in a .csv file with the start and stop points.

It is important that the training data we have is of high quality and has no missing parts, and an evenly distributed start and stop points. The risk here is that randomly a small area has very little data and the network never learns to deal with that area. This is a hard mistake to spot as

			Related		
RI.3.9	Category	Machine Learning			
	Description	Unbalanced dataset used to train model, resulting in portions of the output never being trained properly	Impact	Prob.	Risk
	Mitigation	Make sure the dataset is balanced, otherwise employ methods such as upsampling, downsampling, or SMOTE	3	3	9

Figure 10.5.: Risk RI.3.9

			Related		
RI.3.4	Category	Machine Learning			
	Description	Unexpected results in edge cases.	Impact	Prob.	Risk
	Mitigation	Test models in multiple scenarios and detect any anomalies.	2	4	8

Figure 10.6.: Risk RI.3.4

it may not reveal itself until the network happens to try and predict a path through that area. Further risks include that the network is over-fitted, this happens when a network instead of predicting the output just learns the training data. It will then have really good accuracy on the training data, but terrible accuracy in the real world. It is important to take all this into consideration when creating the data set.

			Related		
RI.3.6	Category	Machine Learning			
	Description	Model overfitting	Impact	Prob.	Risk
	Mitigation	Do frequent error function tests on validation data, and stop training if it gets worse.	3	4	12

Figure 10.7.: Risk RI.3.6

By having a large dataset of high quality meaning the data is properly normalised, has no missing fields and is correctly labelled, the risk of over fitting can be kept low. In addition to this the training notebook used to train Deepstar has visualisations that can show if the network is over-fitted.

This risk was especially prominent during the first half of the project as we did not have access to a dedicated computing server. To mitigate this all models was deliberately kept simple and the scenarios where simplified to ensure we could run the models in a reasonable time frame. As time went on and we made the algorithm run on the GPU on the server this risk was reduced significantly, allowing us to venture into more and more complex models.

			Related		
RI.3.1	Category	Machine learning			
	Description	Training server is not powerful enough to train our models	Impact	Prob.	Risk
	Mitigation	Make sure we have adequate time to train models, and consider acquiring more powerful server, or reduce complexity of models.	3	2	6

Figure 10.8.: Risk RI.3.1

			Related		
RI.3.7	Category	Machine Learning			
	Description	Not enough data for machine learning	Impact	Prob.	Risk
	Mitigation	Prepare data early in model investigations, to ensure you have enough time to generate and prepare data	4	3	12

Figure 10.9.: Risk RI.3.7

For deepstar data generation turned out not to be much of a risk, this was because we had the ability to generate training data. This means we essentially have unlimited training data we just have to invest the time into generating it. Because of this when we got access to the school computing server training data became a non-issue.

There are a multitude of ways to visualise data sets, and for our purposes we build a custom data visualiser for each model we are using. So each model has a corresponding Jupyter notebook for visualising the model output and the input data used for training. For Deepstar we kept it simple to just display the start, end and midpoints of the data set on the map, all at once. For the output we simply just calculated the optimal path and the predicted path from the module and displayed them side by side.

Because the network predicts the midpoint between two points to get a proper path the module has to do this recursively. How many times depends on the distance but between 4-5 runs is satisfactory for the paths we are computing. To fill inn the missing gaps just draw lines from point to point.

10.13.4. Tests

These tests are for a general machine learning based pathfinding module as described in Section 9.3.

T.13.1.1	Requirements	RQ.12.1, RQ.13.1	Related	
Passing	Description	Test that the resulting path is sent to the correct Redis channel		
Done	Criteria	Resulting path is sent to correct Redis key, depending on whether the model is deployed or in staging.		

This test is to verify that the output of the module is sent to the correct Redis channel so that it can be properly read by the backend.

T.13.1.2	Requirements	RQ.12, RQ.13	Related	
Passing	Description	Test that resulting path is in correct format		
Done	Criteria	Resulting path adheres to JSON schema.		

This test is to verity that the output is in the correct JSON schema, this is important especially if you manually construct the output for the module.

T.11.1.1	Requirements	RQ.11.1	Related	
Untested	Description	Test that the computing time varies less than 10%		
Done	Criteria	Run at least 20 different paths inside a model, check time shortest and longest are within 10%		

T.11.2.1	Requirements	RQ.11.2	Related	
Passing	Description	Test that the time doesn't grow exponential with time		
Done	Criteria	Run a model in 10 map sizes, make sure the doesn't grow exponentially larger		

T.11.3.1	Requirements	RQ.11.3	Related	
Untested	Description	Test that map complexity doesn't effect computing time		
Done	Criteria	Run at least 10 different maps with the same size, all results should be within 10%		

To ensure the customer will have a relative predictable computing time for the algorithm we have included this test. Traditional algorithms suffer especially from increasing computing time depending on distance, so we hope to achieve more consistency.

Part IV.

Conclusion

11. The Final Product

We were unable to meet all of our requirements and several features were dropped. However, we ended up with a functional product which is good enough to keep building upon without major changes.

11.1. Pathfinding

From the beginning there was much uncertainty about how far the pathfinding would come. In the end not all requirements where met, there where many reasons for this. For example the ONNX requirement turned out to be not as important. There where also a lack of tests for all machine learning solutions.

In the end three pathfidning modules was created. The first and easiest one is a simple Dijkstra based pathfinding module. This module runs a Dijkstra and is able to find the optimal path, however this path is not flyable.

The second module that was created was a DeepStar based pathfinding module as shown in Figure 11.3. This module is mean to demonstrate how far DeepStar got during the development. As explained before deep star was sadly not able to produce any accurate results. However everything around it works and the limiting factor is research into how to make the machine learning algorithm work.

The last module that was created was created to show off the modified turn-short based Dijkstra algorithm. This module is extremely slow but it serves as a good baseline for creating flyable paths that are Dijkstra optimal, withing the constrains of our simplifications at least.

11.2. The Service

The backend is what glues the entire project together. Without it, there would be no link between the pathfinding and the user interface. A well-written backend just works in the background, and the user does not even notice that it's there. If the user were to notice it, it would probably be bugs or some kind of issue.

The backend provides a REST API which can be used by any client to run the frontend. On the surface, the backend allows users to run pathfinding jobs and get the results. However, there is

more to it than that. The backend also manages the pathfinding modules themselves, and gives administrators and developers the tools they need to control every aspect of the service.

11.2.1. The UI

The frontend managed to complete all priority 1 requirements for the frontend which was considered the minimal goal. Beyond this many of the lower priority requirements where also finished, and some where not finished as was expected.

The finished result have given us a website the offers most of what we set out todo. It allows for casual user who might not be familiar with the modules to be able to use them. First we wanted the user the be able to select variables in which the module would run on. This included the start and stop points, which map to be used and module to be used.

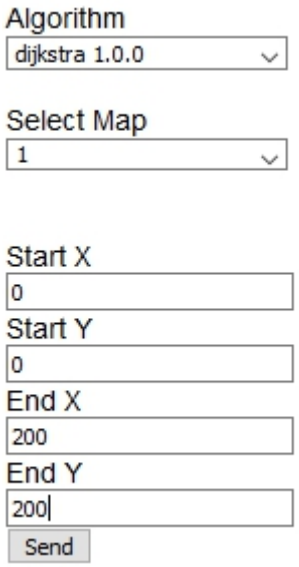


Figure 11.1.: Selecting map, module, and start/stop points

We can allow the user to preview the map before they submit. Here they can also select the markers positions from clicking on the map.

Figure 11.2.: Map where a user have selected a start stop point

Finally we can display a path that was created from one of the pathfinding modules.

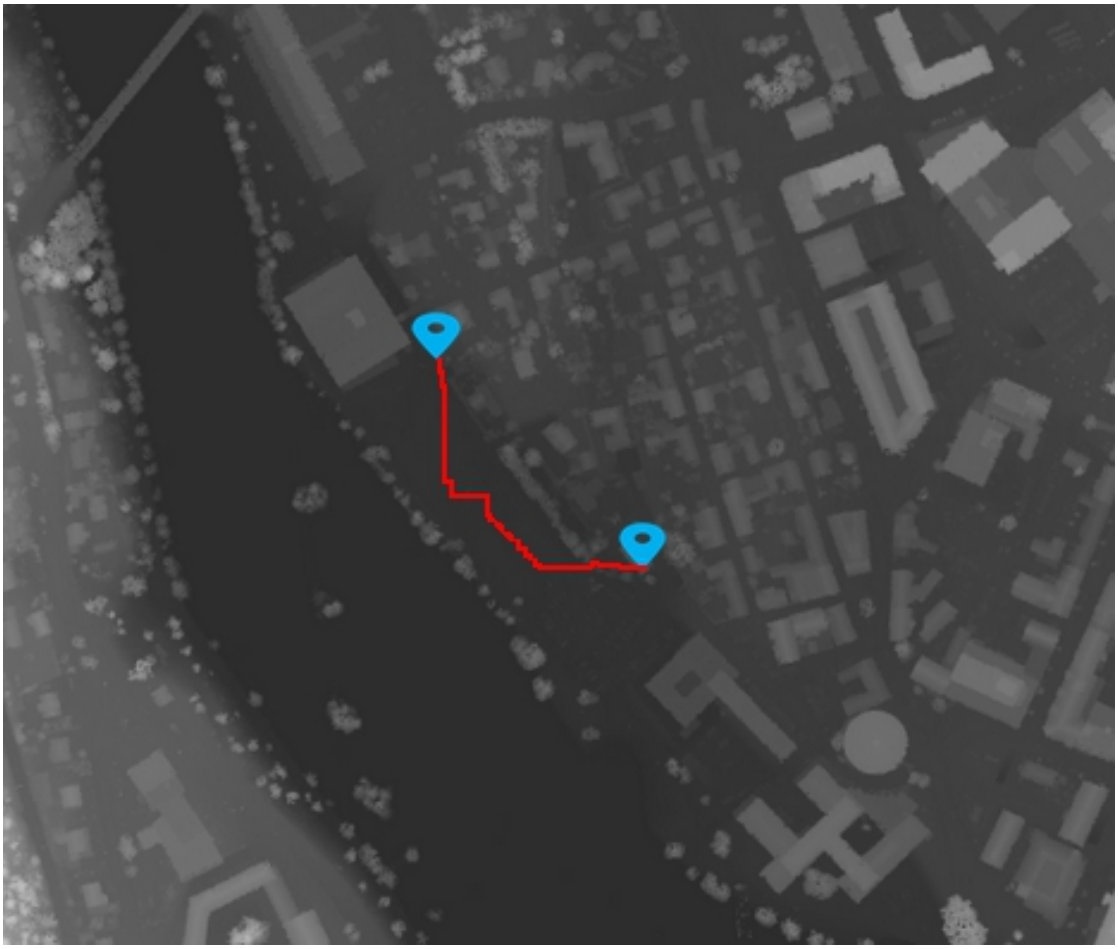


Figure 11.3.: Display of a path generated by a module

11.2.2. Admin Panel

The administration functionality requires authentication, which uses a modern, purpose-built password hashing algorithm[1]. Once authenticated, an administrator has a number of options available to them.

Module Management

The first thing an administrator will see when they go to the admin panel at `/admin` is the list of available modules. This can be seen in Figure 11.4. From there they can manage the modules. It is possible to:

[1]See Section 9.4.1 for details

Figure 11.4.: The admin panel list of pathfinding modules.

- See the state of the modules. Notice how the Dijkstra module has partially failed, letting the administrator know how many instances have failed, and the unique exit codes for debugging.
- Get the logs. The link takes the admin to a page where all the logs are for that module.
- Restart the module. If the module isn't running, a restart is the same as starting it normally.
- Stop a module if it is running.
- Delete stopped modules.

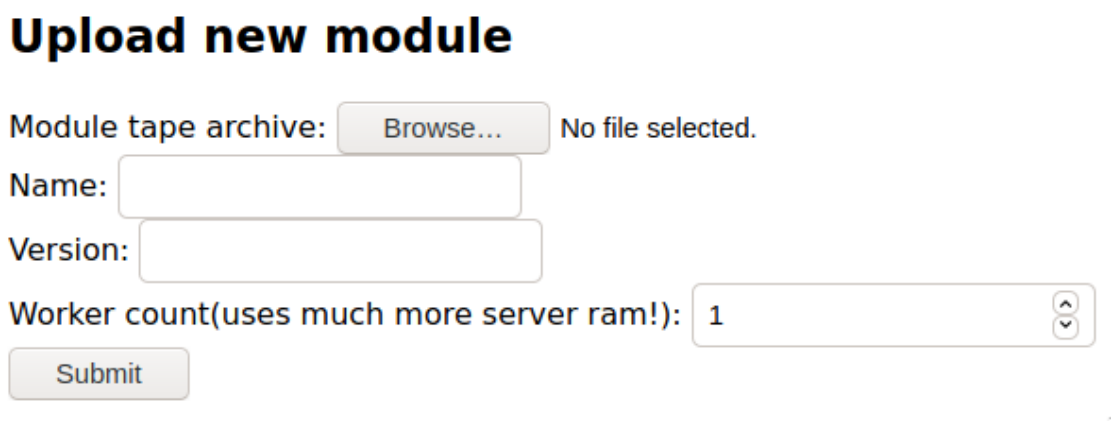


Figure 11.5.: The module uploading panel

There is also the module uploading panel, as shown in Figure 11.5. This allows the administrator to upload modules. The admin explicitly states what the name and version of the module is. The upload itself has to be a tape archive as we talk about in Section 9.3.3. The last field, the number of workers, describe how many workers which can be spawned by the backend. A higher number means more jobs can be processed at once, but will increase memory usage on the server significantly.

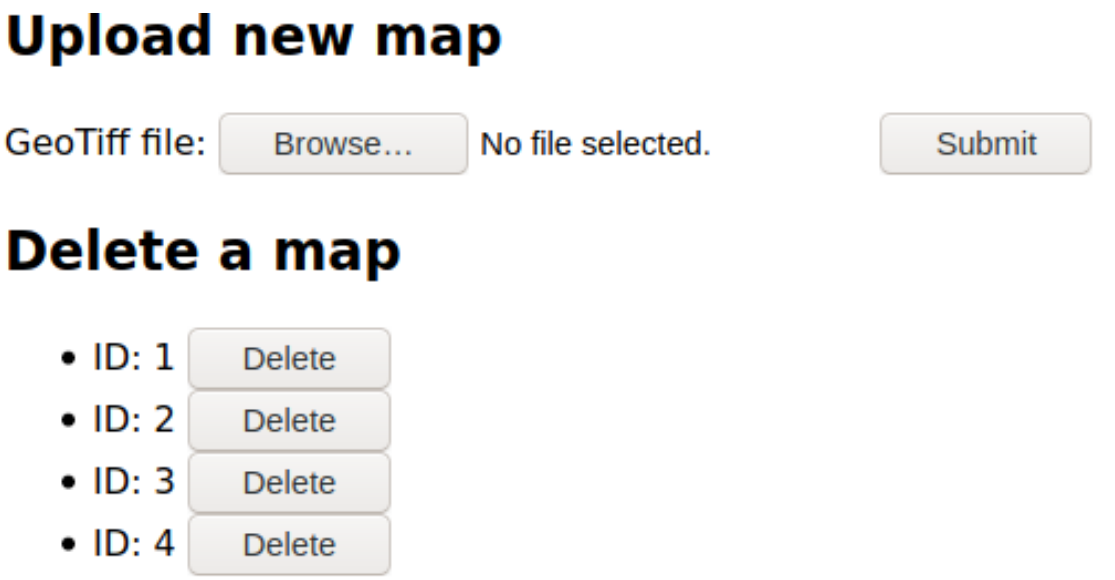


Figure 11.6.: Admin map management

Map Management

Administrators can also manage maps. By using the upload panel here, one can upload map data from GeoTiff files. All the necessary conversions will be done to integrate the map into the service. They can also delete the maps with a very simple interface, as illustrated in Figure 11.6.

Admin Registration

If the administrator is a super admin, an admin registration widget is shown to the user. The confirm box is just there to verify that the password which is typed in is correct. It updates the status as the password and confirm boxes are shown in Figure 11.7.

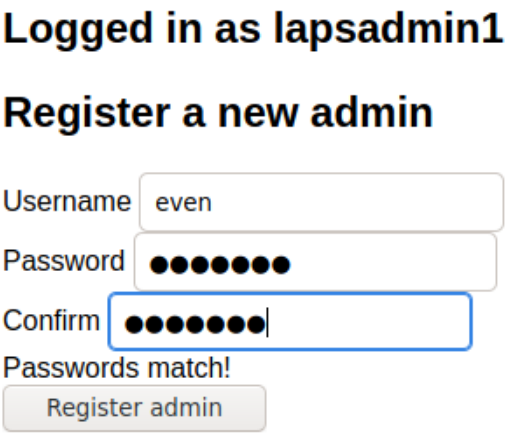


Figure 11.7.: Administrator registration

.

12. Lessons Learned about Machine Learning for Pathfinding

Though we did not make great progress into solving the problem, we were able to come up with and explore some potential solutions for it. What we have experienced, based on both our limited experience with this subject but as well as the general research area, you either get it to work or you don't know why it doesn't work.

This is a side effect of how "unscientific" machine learning can be, as there is no clear and mathematically defined way to approach a problem with. Machine learning is often explained as more of an art than a science, you need experience and even with that you might not know where to begin. And even then to tune the algorithms hyper parameters is a case of trying and seeing what sticks.

For future projects we would recommend having a more methodical approach, with less variations between the attempts, to be able to pinpoint the effects of changing the hyper parameters. There are multiple technologies that enables this by having the computer go through many different hyper parameters. We did not invest time into this at the beginning, for both the reason of computing power and time.

The problem turned out to be much harder than anticipated. As it was a problem domain not explored much before, it was hard to judge whether or not the decisions we made were the right ones, and if it brought us any closer to a meaningful solution or not.

In addition we had limited experience within the field of machine learning, only being familiar with the main concepts beforehand and no experience with PyTorch, as machine learning is not something we had been taught in our program.

We were also only two people working on the machine learning investigation, which did not make the matter any easier. All these factors adds up, and all in all the months we had to investigate the problem was too little time to be able to get huge progress on the problem.

We did not manage to find any evidence against machine learning being a viable solution, but what we did find were some methods which seemed promising, but were not explored well enough to be certain in our findings.

In particular, we did not manage to find how well a more advanced network structure with more training time would be able to perform on the reinforcement learning Turn Short approach. The complexity increase from placing one point to two points or more is not too big, as it is mostly

the same operation. Therefore if the agent would be able to meaningfully well place one point, this may indicate that it would be able to do the same with more waypoints.

Using more advanced learning algorithms may both increase the learning speed and the accuracy it is able to achieve, so it is possible that there is much more well suited algorithm for the problem which would be able to learn it.

13. Reflections

In this chapter, we reflect on the project as a whole.

13.1. The Process Model

The model worked quite well in the beginning. We were able to follow it to a tee, and didn't really have any problems. It became apparent that the tree structure of features, requirements and tests were not the best way to organise, but it worked well enough to not cause any major issues.

Unfortunately, the machine learning half of the group had trouble following the model. This was due to the nature of their work being investigative. It is very hard to come up with requirements for a model like that, and we didn't even know if using machine learning here was even possible. For future projects that decide to do a half investigative project it might be smart to look into somewhat separate project models as well.

However, the service half had more conventional tasks. This worked better with the model pipeline. However, some tasks took a very long time, which made the model hard to follow. We had a Kanban card up for about a month, which was stuck. This happened because the task took a lot longer than expected.

The card got stuck, because there was an API change in the backend. This change was necessary because there was originally no way to select which algorithm one wanted to use, and there was only one available. This change meant that some of the frontend code had to be changed. The reason it took so long was that a dropdown menu wasn't able to grow dynamically.

The card eventually became a conglomeration of every frontend task required to achieve a minimal viable product, making it too large for it's own good. With hindsight, we should have changed it. Another issue was that we also had exams during this period.

13.1.1. The COVID-19 Shakeup

When the university, and indeed all of Norway, was closed due to the pandemic, the project model fell apart. Because we now were working from home, it became harder to collaborate effectively. This meant that over time, we used the model less and less. One thing which may have prevented this could have been pair programming. The problem with that, is that we

all had different responsibilities, and therefore there were few things which more than one member had had their hands on.

We didn't need any equipment other than a development environment and remote servers. This meant that it was mostly business as usual in terms of our ability to work, but it did cause issues with communications. For example one plan was to do pair code review, this fell through when we where forced to work from home. We did identify this problem and decided to instead verify our own code within the two sides of the project.

Another important part is how the pandemic affected our working habits. Sitting at home, it is harder to remain focused on the task at hand. The line between work and free time became fuzzy. Some of us dealt with this better than others. Despite general productivity taking a dive, we were still able to produce results.

The pandemic did not just impact us negatively, because we where all working from home, including our customer, we where able to have daily meetings with him. This allowed us to get rapid feedback on our progress and new ideas on what to do for that day. This mean for the last four weeks of our progress we managed to do a lot of rapid prototyping for the machine learning part of our project. This arrangement mostly benefited the machine learning part.

The Project Model

After the pandemic hit, our project model continued as normal for a while. It eventually broke down by a lot, as it was harder to communicate with each other. When communicating via text, it is harder to keep others responsible for certain things. Nonetheless, we kept certain aspects of the model for a long time.

In particular, we kept doing the standup meetings every day. This worked quite well as a way to keep in touch with what was happening in the different parts of the project. Because we were using a project model which demanded independence, we were pretty well prepared for this situation, as everyone kept working on their parts.

13.2. Working Together

In general it was pretty easy to work together. We all went to the same high school (videregående) in Kongsberg, and some members have known each other since primary school. This meant that we were good friends from the beginning, which helped us be comfortable with each other and speak our mind.

In the beginning, there were many arguments as to how things should be done. Sometimes we ended up pretty annoyed at each other. However, this did not end up plaguing the project too much. Usually everything had cooled down the next day and we were back to normal.

These arguments pretty much faded away as the project went along. It seems like we did all our arguing early in the project. When the deadline for this report was approaching fast, we all felt the stress, but we were all on good terms.

13.3. Hours Worked

We kept track of how many hours we worked during the course of the project. After finishing our work for the day, we put the hours into a spreadsheet. Each week report has a list of hours worked that week. See Figures 13.1 and 13.2 for an overview.

13.4. The Product

In this section, we break down how we think the final product turned out.

13.4.1. Frontend

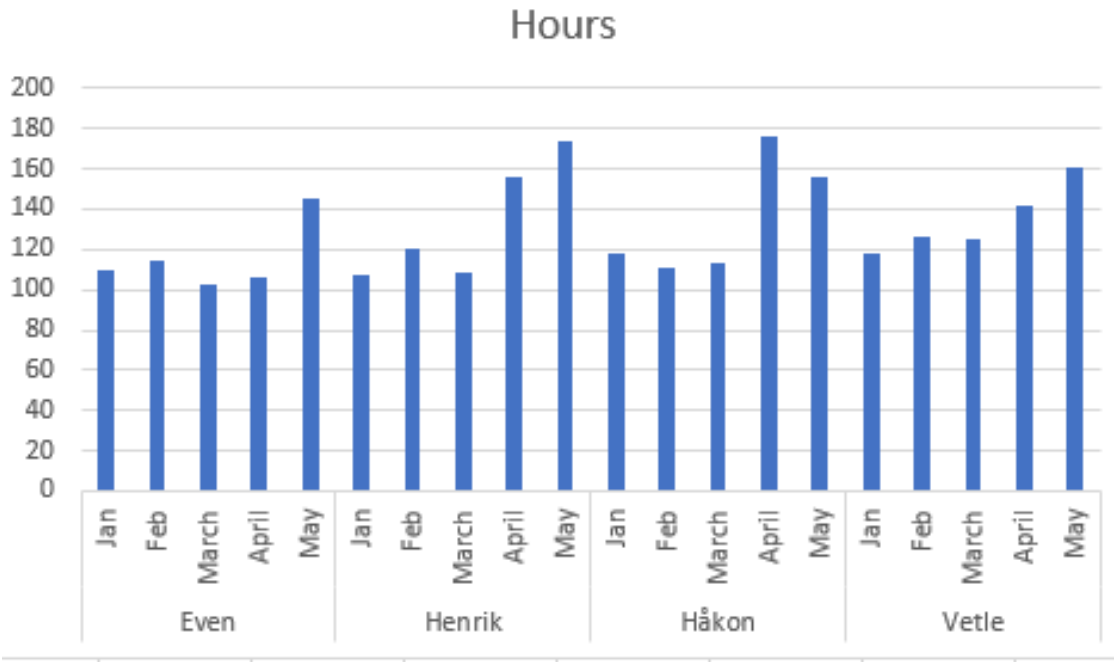
Overall the frontend succeeded in what it set out to do. The goal was to let users who where unfamiliar with how the modules works to have some interactions with them. All the requirements that had priority 1 was implemented. Many of the lower level requirements where also implemented. Some of the features implemented in the backend needed a frontend element to allow users to interact with. The frontend where able to implement all features that the backend needed it to.

There still more things the frontend ideally should implemented. Many requirements that where purely in the frontend where deprioritized. As we wanted the features implemented in the backend to also have their frontend UI. This left the frontend to a certain degree more of a way to interact with the backend/modules than its own things. This isn't necessarily bad, and was the main purpose of the frontend, but I would have liked to see more frontend only features.

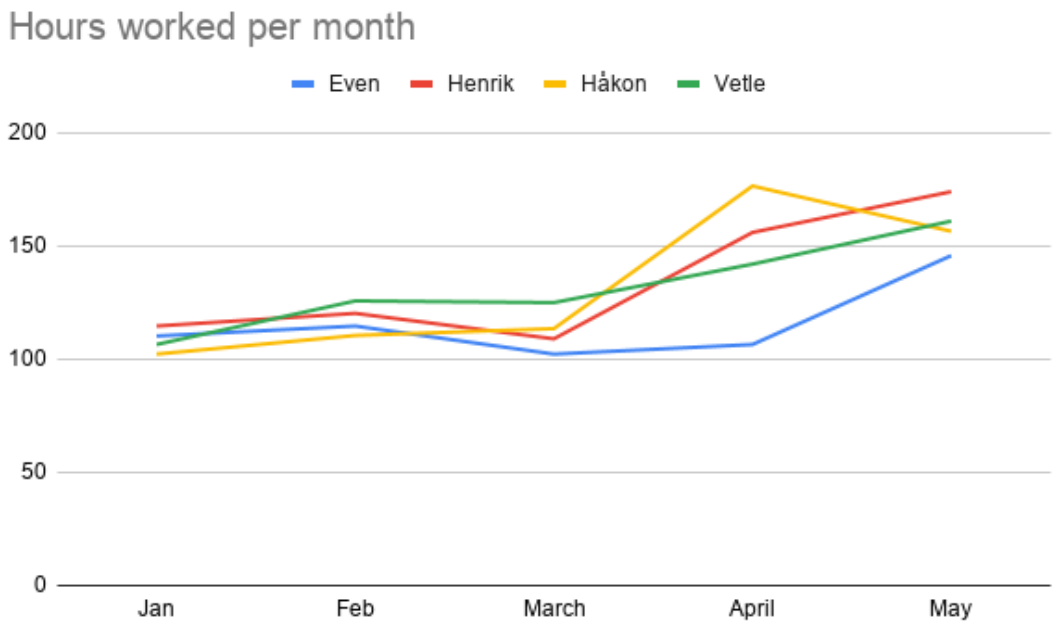
13.4.2. Backend

In general, we believe that the backend turned out quite well. An important part of this is the extensive error handling. Writing it in Rust really helped in this regard. If a function can fail, one uses a wrapper type, and to get the resulting value, one has to unwrap this type, as shown in Listing 8. This means that one always knows which function calls can fail, and forces one to handle them in some way. In the vast majority of cases, we bubble them up to the user and return an appropriate error code.

The code is pretty well-tested. Like we said in Section 9.2, most parts of the backend code tests which verifies that it behaves correctly. One of the reasons there test suite is so expansive is



(a) Hours worked as columns.

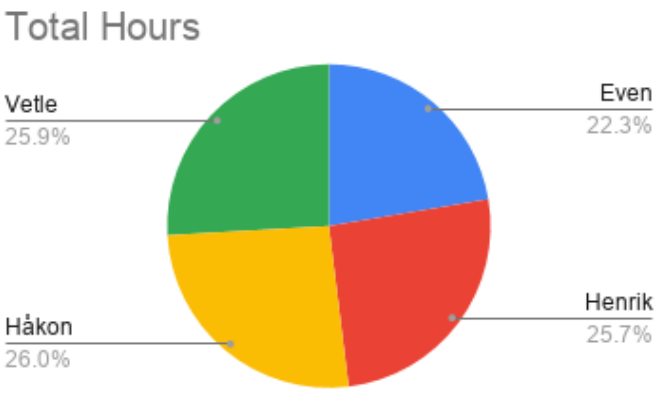


(b) Hours worked as a line graph.

Figure 13.1.: Number of hours worked by each team member per month.

Member	Total Hours
Even	579.4
Henrik	666.4
Håkon	675.45
Vetle	671.75

(a) Total number of hours as a table.



(b) Relative hours worked as a pie chart.

Figure 13.2.: Total number of hours worked by each team member

```rust
fn divide(x: i32, y: i32) -> Result<i32, &'static str> {
  if y == 0 {
    Err("Cannot divide by zero!")
  } else {
    Ok(x / y)
  }
}

divide(1, 0); // -> Err("Cannot divide by zero!")
divide(25, 5); // -> Ok(5)

match divide(1, 0) {
  Ok(n) => println!("Got {}", n),
  Err(e) => println!("Error: {}", e)
}
```

Listing 8: Rust error handling example

because the frontend was developed separately. This forced us to write a lot of tests, but this was very useful as complexity grew and we touched older parts of the code.

As a result of the well-done error handling and thorough tests, we believe that the backend turned out quite well. All it's functionality is well-tested and it has for the most part just worked as we gradually got the frontend in place.

13.5. Further Developments

A project can rarely be called complete. There are many things which we can improve upon. In this section we shed some light on what can be done in the future to further the product as much as possible.

There are some features and some requirements which we did not get to complete due to a lack of time.

13.6. Frontend

A few days before our group was starting to do purely documentation the customer asked to get some some data about returned path. Some days of work but where put into this, and the feature where mostly done, but had to get the rest of documentation in place. Future developments would certainly contain finishing this feature. It might also be done, but as documentation takes priority, it will be finished after this and therefore not be properly documented.

The aesthetic of the website should also be improved. The website never looked terrible, but just a bit simple. Artistic design isn't the strong suit of the frontend developer either.

Another feature that was coming up in developments was the ability to have multiple paths on the map at once, and therefore see some comparison of how modules would generate different paths. This would been nice to see the difference between something like Dijkstra and one of our machine learning algorithms.

13.6.1. Backend

Besides the aforementioned code improvements and completions of requirements, there are a lot of things to add in the backend. For example, there should be a way to refresh a module. There may be changes to the software architecture which have to be handled in the runtime library.

After we developed and tested the backend, there has been a significant update to Redis, version 6.0.0[1]. This version adds user functionality. This means that we can further isolate the pathfinding modules from the rest of the system. We could use this feature to disallow modules from changing certain keys and more. This would allow us to make it impossible for the modules to cause issues by overwriting backend data.

Sometimes we make changes to the module library because we have made changes to the general architecture. If we haven't changed the API of the module library yet, we should be able to refresh the module with the latest version of the library. This would mean that we need to store the module tarballs uploaded to the service. If we rebuild the module image from the tarball, then we will be up to date.

13.7. Milestones

As we expected, we did not complete all our milestones. We have made an overview of which milestones we have completed in Figure 13.3 as of handing in the report. As one can see, we have achieved about half of the milestones we set up.

[1]Changelog:`https://raw.githubusercontent.com/antirez/redis/6.0/00-RELEASENOTES`

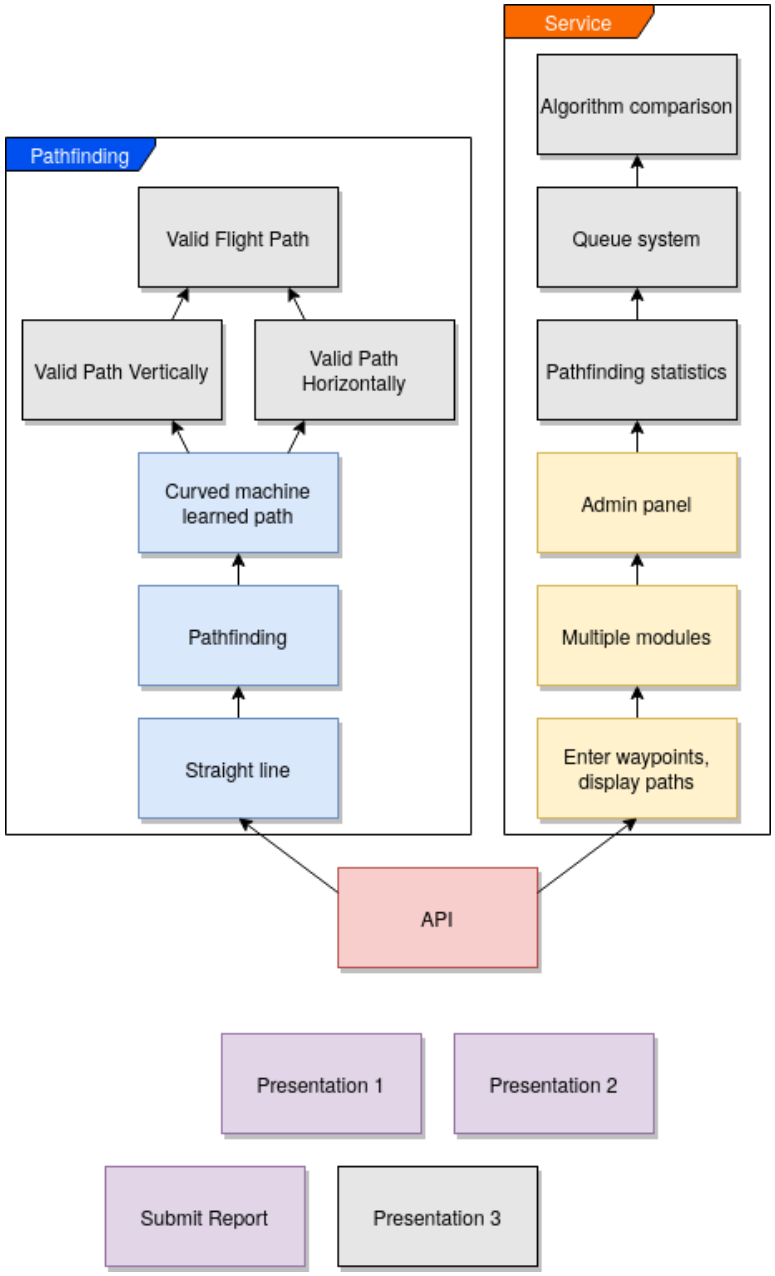


Figure 13.3.: Completed Milestones. The grey colour symbolises that we have yet to achieve that milestone.

13.8. General Thoughts

13.8.1. Even

Going into this project I knew very little about JavaScript and never used Vue before. Having to learn a new software while product was being developed offered some challenges. Writing code while still being unfamiliar with the language, meant a lot of it wasn't gonna be ideal. This meant that a lot the code was probably going have to be rewritten at some point. I also had figure what was good enough to let stay. I could have designed the frontend not as a web service, and used language i am more familiar with. However I wanted to challenge my self, and this seemed like a good way to do it. I also my now have a better understanding of these languages, for latter uses. It's not unrealistic to expect, while working in software, that you are going to enter a project where you have to learn a new language, while simultaneously having to show progress on the project itself. I thought this served as good experience of that, as the bachelor project is a more serious project, that we wish to do well in.

One disappointment was that the COVID situation came in at a very unfortunate time. Development speed was picking up and I was getting over it the initial learning curve. Having to reorganise working rhythms and having to work more alone it became harder to keep motivation up.

I think overall i learned a lot from this project. During the entire project I felt that I was being challenged by the tasks at hand. I ended up getting some good experience about working on a larger project than I had previously worked on.

13.8.2. Henrik

Even though as of writing this we did not manage to achieve a proper machine learning based solution to pathfinding I still think this project went well. Prior to this i only had partial theoretical knowledge about machine learning. So over the course of this project i learned much about how to develop a proper machine learning solution, and the difficulties of how to do so.

The knowledge I gained in the PyTorch library is especially valuable, this is because it is a commonly used framework in both academia and production settings. So whenever i go i can take my experience with building machine learning solutions with me.

As has been described before in this report there was so much trial and error involved in investigating this. I don't think I quite realised it when I began working just how much guesswork goes into developing a machine learning solution. The best I could do was to try to emulate what i read in research papers and tutorials and adapt them into my situation.

I am very thank full that my half of the project was from the beginning designed to be investigative, i now have a lot of respect for people who can develop working machine learning algorithms on a schedule.

13.8.3. Håkon

I am very pleased with how the backend turned out. I wasn't used to documenting my code as extensively as I have done, but I got used to it pretty quick. I think the code base itself is one of my better works. Nothing is ever perfect, and the backend code isn't either. There are a few things I want to tear out and re-think, but all in all it is good.

In some places we assume that a certain error will never happen. Things like serialising a struct to JSON, reading module responses and such. In certain places we should probably handle the error. For example we assume that modules follow the protocol. If it sends a message on an unexpected format, we just unwrap the error, which causes the thread to panic. This sometimes crashed the backend when I was developing the module development library. That could be better, but in the real world it does not matter that much because modules are already considered trusted.

13.8.4. Vetle

During this project, I was very often worried and stressed out about our lack of progress regarding pathfinding. Nonetheless, I had a good time working on this project. I got to work on a group with close friends, still staying friends by the end haha.

I am very thankful for the opportunity to work on a machine learning related project such as this, as it is a field I find very interesting but have not yet had the chance to work with much before this. It always made me super happy whenever I got to speak with somebody knowledgeable on the subject, and discuss our problem. I have learned so much during this project about the subject machine learning, the process, as well as the tools, which I am happy about. All in all it has been a great experience.

Bibliography

[1] Dario Izzo, Christopher Sprague and Dharmesh Tailor. "Machine learning and evolutionary techniques in interplanetary trajectory design". In: (Feb. 2018).

[2] David J. Anderson. *Kanban: Successful Evolutionary Change for Your Technology Business*. Blue Hole Press, 2010. ISBN: 9780984521401. URL: https://books.google.no/books?id=RJ0VUkfUWZkC.

[3] *Redis Online Documentation*. URL: https://redis.io/documentation (visited on 08/05/2020).

[4] *Information technology – The JSON data interchange syntax*. Standard. Geneva, CH: International Organization for Standardization, Nov. 2017.

[5] Wikipedia, the free encyclopedia. *Neural Network Example*. [Online; accessed March 5, 2020]. 2020. URL: https://upload.wikimedia.org/wikipedia/commons/thumb/3/3d/Neural_network.svg/800px-Neural_network.svg.png.

[6] Wikipedia, the free encyclopedia. *Graph Convoltution Example*. [Online; accessed March 6, 2020]. 2020. URL: https://miro.medium.com/max/576/1*Y2xcReChWy3JYGSw0_Rt0Q.png.

[7] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. ISBN: 978-0262193986. URL: http://incompleteideas.net/book/the-book-2nd.html.

[8] Jie Zhou et al. *Graph Neural Networks: A Review of Methods and Applications*. 2018. arXiv: 1812.08434 [cs.LG].

[9] Naimish Agarwal, Artus Krohn-Grimberghe and Ranjana Vyas. "Facial key points detection using deep convolutional neural network-NaimishNet". In: *arXiv preprint arXiv:1710.00977* (2017).

[10] L. Wolf, T. Hassner and I. Maoz. "Face Recognition in Unconstrained Videos with Matched Background Similarity". In: *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*. CVPR '11. USA: IEEE Computer Society, 2011, pp. 529–534. ISBN: 9781457703942. DOI: 10.1109/CVPR.2011.5995566. URL: https://doi.org/10.1109/CVPR.2011.5995566.

[11] *Autoencoder Schema*. URL: https://en.wikipedia.org/wiki/Autoencoder#/media/File:Autoencoder_schema.png (visited on 21/05/2020).

[12] *Introduction To Network Visualization*. URL: http://www.martingrandjean.ch/introduction-to-network-visualization-gephi/ (visited on 19/05/2020).

[13] Springer Verlag GmbH, European Mathematical Society. *Markov property, Encyclopedia of Mathematics*. Website. URL: http://encyclopediaofmath.org/index.php?title=Markov_property&oldid=26609. Accessed on 2016-10-11.

[14] OpenAI. *OpenAI CartPole Gym*. https://gym.openai.com/envs/CartPole-v1/. 2016.

[15] *Define Reward Signals*. URL: https://www.mathworks.com/help/reinforcement-learning/ug/define-reward-signals.html (visited on 19/05/2020).

[16] Itai Caspi et al. *Reinforcement Learning Coach*. Dec. 2017. DOI: 10.5281/zenodo.1134899. URL: https://doi.org/10.5281/zenodo.1134899.

[17] Satwik Kansal and Brendan Martin. *Learndatasci*. URL: https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/.

[18] Tom Everitt, Ramana Kumar and Marcus Hutter. *Medium*. Aug. 2019. URL: https://medium.com/@deepmindsafetyresearch/designing-agent-incentives-to-avoid-reward-tampering-4380c1bb6cd.

[19] Patryk Chrabaszcz, Ilya Loshchilov and Frank Hutter. "Back to Basics: Benchmarking Canonical Evolution Strategies for Playing Atari". In: *CoRR* abs/1802.08842 (2018). arXiv: 1802.08842. URL: http://arxiv.org/abs/1802.08842.

[20] *Vue Website*. URL: https://vuejs.org/ (visited on 21/05/2020).

[21] *Axios GitHub page*. URL: https://github.com/axios/axios (visited on 23/05/2020).

[22] Matt Miller. "Trends, Challenges, and Strategic Shifts in the Software Vulnerability Mitigation Landscape". BlueHat IL. 2019. URL: https://www.youtube.com/watch?v=PjbGojjnBZQ.

[23] Marco Carvalho et al. "Heartbleed 101". eng. In: *IEEE Security & Privacy* 12.4 (2014), pp. 63–67. ISSN: 1540-7993.

[24] Steve Klabnik and Carol Nichols. *The Rust Programming Language*. No Starch Press, 2019. ISBN: 9781718500440. URL: https://nostarch.com/Rust2018.

[25] Seve Klabnik. "Rust's Journey to Async/Await". QCon New York. 2019. URL: https://www.youtube.com/watch?v=lJ3NC-R3gSI.

[26] I. Fette and A. Melnikov. *The WebSocket Protocol*. RFC 6455. http://www.rfc-editor.org/rfc/rfc6455.txt. RFC Editor, Dec. 2011. URL: http://www.rfc-editor.org/rfc/rfc6455.txt.

[27] M Singh and D Garg. "Choosing Best Hashing Strategies and Hash Functions". eng. In: *2009 IEEE International Advance Computing Conference*. IEEE, 2009, pp. 50–55. ISBN: 9781424429271.

[28] George Hatzivasilis. “Password-Hashing Status”. eng. In: *Cryptography* 1.2 (2017). ISSN: 2410387X. URL: http://search.proquest.com/docview/2124637250/.

[29] Chiyuan Zhang et al. “A Study on Overfitting in Deep Reinforcement Learning”. In: *CoRR* abs/1804.06893 (2018). arXiv: 1804.06893. URL: http://arxiv.org/abs/1804.06893.

[30] OpenAI. *Getting Started with Gym*. https://gym.openai.com/docs/. 2016.

Appendices

A. Code review process

- Am I able to *understand* the code easily?
  - Tabs/Spaces
  - Ensure that proper naming conventions
  - Code should fit in the standard 14 inch laptop screen.
  - Code commented
    - Each public function has a description
    - All parts of every function has adequate explanation
    - You can understand the code in a reasonable timeframe
  - Avoid multiple if/else blocks.
- Is the code written following the *coding standards/guidelines*?
  - Proper file structure
  - Code follows pattern
  - Code is maintainable
  - No hard coding, use constants/configuration values.
  - Too much generalisation
  - Is the code properly logged
- Is the same code *duplicated* more than twice?
  - Group similar values under an enumeration (enum).
  - Use framework features, wherever possible instead of writing custom code.
- Can I *unit test / debug* the code easily to find the root cause?
- Is this function or class *too big*? If yes, is the function or class having too many responsibilities?

B. Guide to the backend source code

B.1. Introduction

This is an overview of the source code for the backend. The source code can be found in our main repository on our GitHub, `https://github.com/Laps-group/laps`. It is structured like pretty much every other Rust project, with the backend specific source code in the `src` folder. The frontend code lives in the same repository in the `frontend` folder.

B.1.1. Build instructions

To compile and run the backend, one needs a nightly Rust compiler and an installation of the GDAL library. For convenience, a Nix expression resides in the repository which automatically sets up the required libraries if one is using the Nix package manager. Note that a nightly Rust toolchain must still be provided. A nightly Rust toolchain can be set up through rustup[1].

Check out the `gdal_sys` crate[2] for information on how to successfully link the backend with the GDAL library. Other than this, the only dependencies are Rust ones which are downloaded and built automatically by Cargo.

The backend is the root crate in the workspace, so in order to build only the backend, running `cargo build` is enough. The backend can conversely be built and run with `cargo run` as per usual.

The backend will serve the required files from the frontend if they are built. To build the frontend, install the JavaScript dependencies by running `npm install`. The frontend can then be built using `npm run build`, or alternatively with `npm run build_prod` for building a minimised version.

B.1.2. Logs

The backend uses logging extensively. Depending on the Rocket environment which has been set, the backend will log differently. On certain log levels, the log output from Rocket itself is disabled entirely. The best way to get a feel for what each environment sets is to read the source code of the `setup_logging` function in main.rs.

[1] url:`https://rustup.rs`
[2] url: `https://github.com/georust/gdal/tree/master/gdal-sys`

B.2. Workspace

The backend consists of a workspace of crates. There are only two extra crates as of now, `laps_convert` and `laps_convert_cli`. One is our crate for converting map data into usable formats, and the other is the command-line utility which allows one to do so manually. The backend also uses `laps_convert` in the admin panel code.

B.3. High-level overview

Most of the code resides in the `web` module. This is the code which defines the REST endpoints to be used in the frontend. It contains a function which bootstraps the entire backend. It will spawn the module handling code, the web server itself, and connect to Redis.

There's a couple other files in the source directory. The `module_handling` module contains all the needed code to handle pathfinding modules. The handlers within listen to every message sent from the modules.

Furthermore, there's a module which defines error types, `types`, and another which defines a lot of utility functions for creating Redis keys etc.

B.3.1. Web modules

Each of the web modules have self explanatory names which explain what category they are. In each of the files are the function handlers for each endpoint, as well as a set of tests which test that endpoint.

B.4. Tests

The backend has been developed using tests. We would not call it test driven development, but there is a lot of test code. The test code resides at the bottom of various source files, and some tests reside in their own file dedicated to tests as the volume of test code was high.

As we have mentioned, the tests can only be run sequentially. To make this easier, we are using a library called `serial_test`, which forces all tests to run in sequence.

B.5. Module runtime library

Because the pathfinding module API can change a bit between releases, we eventually decided to put the library in the same repository as the backend. This was especially apparent when we started to build Docker images for each module automatically in the backend.

The `module_runner` directory contains `laps.py`, which is the class responsible for giving modules an API for interfacing with the rest of the system. It also contains the Dockerfile used to build the module images from the input tarball. The API reference can be found in Section D.6.

C. Guide to the machine learning source code

This is an overview of the machine learning source code, the source code can be found in the GitHub repository at `https://github.com/Laps-group/Pathfinding`. The most important files can also be found as PDFs on the memory stick under source_code/machine_learning.

C.1. Utilities

C.1.1. Custom python loader

In the top level one will find the loader.py file. This is a custom python loader that was created to let us import Jupyter notebooks into other Jupyter notebooks.

C.1.2. LAPS.py

For development, the repository has a copy of the LAPS runtime library.

C.2. DeepStar and GraphStar

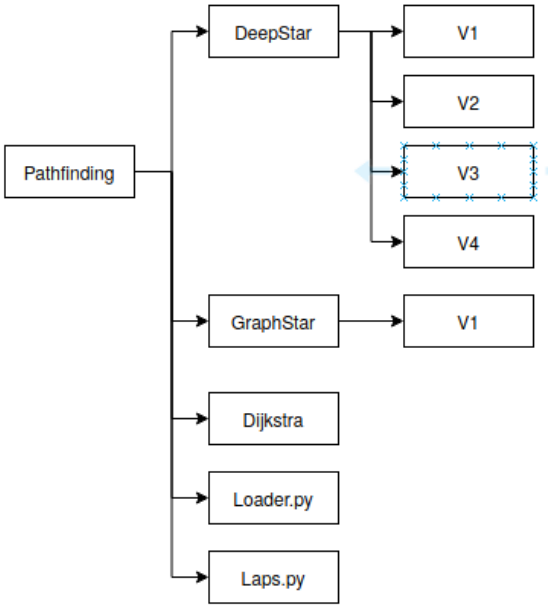


Figure C.1.: DeepStar and GraphStar overall file structure.

As both DeepStar and GraphStar were created using supervised learning and by the same person they follow identical structures that will be explained in this chapter.

The algorithms are split into versions, where each version represents a major idea or modification to the code. There is no hard rule for what deserves a version bump, but generally a version bump occurs when the input or output is no longer compatible with previous versions. This versioning system is strictly for internal use and is not comparable to semantic versioning.

Each version contains a minimum of four Jupyter notebook files and two folders. The first folder is the data that is used for that version. This folder is itself split into training and validation data. The split between training and validation data is done manually, in order to have increased control of what is used in each from version to version. The second folder is called models and contains all previously saved models.

C.2.1. Training Notebook

Of the four Jupyter notebooks, one is shared between versions. This is the training notebook. This notebook started out in version one as a simple, purpose-built model trainer. When GraphStar came around, it morphed into a general-purpose model training notebook. The specifics of this notebook and its evolution is discussed. Apart from this notebook, the rest are purpose-built for each version.

C.2.2. Model Notebook

The first file called Model.ipynb contains the PyTorch model definition of the network and a accompanying data loader. The model defines the internal structure of the network as well as the input and output. In addition to this, each model definition has two methods, called `get_optim` and `get_loss`. They define the optimisation and loss algorithms used by the training notebook to train the network.

The data loader defines how the data folder will be structured and how to load the data into memory. It does not define how the data is fed to the model during training as there is a data transformation layer between it and the model. This is to allow for the re-purposing of data loaders for new models.

C.2.3. Data Generation Notebook

The next file, DataGeneration.ipynb, is responsible for generating the data that the data loader can use. This data generator is guaranteed to generate data compatible with the data loader. Some versions share the data generation if their inputs are similar enough.

C.2.4. Visualisation Notebook

Then there is the last file that is usually created in a versions lifespan called Visualizaiton.ipynb. This file is the file that contains all visualisations that are related to that version. This can range all the way from model structure visualisation to output comparison.

C.2.5. Reinforcement Learning

The reinforcement learning code is split into two parts:

- A Python Package - A Python package containing two modules: gym_drone, which is all the OpenAI reinforcement learning environments, and turn_short.
- Jupyter Notebooks - Implementations of the learning algorithm Double Deep Q-Learning, as well as normal Q-learning. These notebooks are all based on the same notebook, and then iterated on.

The gym_drone environments are explained in Section 10.12, and the turn_short module is explained in Section 6.7.

Jupyter Notebooks for reinforcement learning were not changed much, as it is mainly the environments which were changed. The notebooks are:

- cardinal_directions_q_learning.ipynb - Q-Learning implementation, connected to the Cardinal Directions Environment.

- cardinal_directions_ddqn.ipynb - Double Deep Q-Learning connected to the Cardinal Directions Environment.
- turn_short.ipynb - Double Deep Q-Learning algorithm connected to the Turn Short Environment.

C.2.6. Running the code

All code is currently setup so that you can pull from the repository, and as long as all the required depndencies has been installled it should run. Currently it is recommended to run on the GPU as running any models on the CPU takes a considerable amount of time.

D. Writing a Pathfinding Module

In this chapter we explain how to write a pathfinding module to work with the system.

D.1. Introduction

Modules are written in Python. There is a set protocol that each module has to follow to work with the system. Modules communicate with the backend using Redis, but this is not something a module developer needs to concern themselves with. We have written a library which module developers should use instead of using the protocol directly. One can read about the protocol in Appendix E.

When a module receives a pathfinding job, it has to calculate a path from the job and return it to the backend. This is done using a callback. The callback is called whenever a job is received, and must return the result of the job. The returned path is expected to be continuous from the start point to the end point. The system does not interpolate at all, if this is needed, it is the responsibility of the module developer.

D.2. Getting Started

To get started writing a pathfinding module, one needs administrator access to a running instance of the service. It is also possible to work somewhat outside the system by running the module in development on a server which hosts the service. This is the easiest way, because it gives more direct feedback without having to package the module, import it into the system, and start it.

If one wants to run the module locally, it is important to make sure one has a current version of the runtime library. This can be gotten from the source tree of the backend and frontend, in the `module_runner` directory. The file should lay in the same folder as the module code.

All modules have an entry point which must be named `main.py`. Begin by importing the runtime library and creating an instance of the `laps.Runner` class like so:

```
import laps

# The scope is important to handle shutdowns correctly
```

```
with laps.Runner() as runner:
  pass
```

The `Runner` class constructor automatically parses the command-line arguments.

Next, define the callback and register the module:

```
import laps

def callback(runner, job):
  # Extract the start and stop points from the job object.
  start_x, start_y = (job["start"]["x"], job["start"]["y"])
  stop_x, stop_y = (job["stop"]["x"], job["stop"]["y"])

  # Return some path(a dummy in this case)
  return [{"x": x, "y": y} for x,y in [(1, 2), (2, 3), (3, 4)]]

with laps.Runner() as runner:
  runner.run() # Register and run the module.
```

And that is pretty much all one needs to write a very simple module. Of course, this one is no good as it does not perform any actual calculations, it returns the same path no matter what. A good path would go from the start point to the stop point, where all the points are next to each other.

Map data is stored as greyscale PNG images. Retrieving the byte data for one is as simple as using the `Runner.get_map_data` function. While it is possible to get the data from Redis directly, this is not considered good practice because these helper functions perform error handling automatically[1]. The higher the intensity of a pixel, the taller the map is at that point.

```
import laps, io
# Or use some other library capable of decoding PNG images
from PIL import Image

def callback(runner, job):
  # If the metadata is relevant for this algorithm,
  # one can retrieve this as well:
  metadata = runner.get_map_metadata(job)

  image_bytes = runner.get_map_data(job)
  with Image.open(io.BytesIO(image_bytes)) as img:
    pixels = img.load()
    # do stuff with the pixel data here
```

[1]See Section 9.3.7 for details.

D.3. Running the Module

If one has a local development environment, all one has to do is to execute the module entry point script. It takes some command line arguments which are mostly for use in the deployed version of the service. To run it locally, one only has to specify the name and version that the module should register itself as. Something like

```
$ python3 main.py my-module 0.1.0
```

should be enough. Log messages saying that the module is registered should be seen in the terminal. The module is now waiting for jobs. It might be worth having a look at the help to see the available command line options:

```
$ python3 main.py --help
```

If not working locally, the module must be packaged first.

D.4. Packaging and Uploading a Module

Modules are packaged as uncompressed tape archives. One can put whatever files the module needs in there. This means one can split it up into multiple files quite easily. The two required files are `main.py`, the entry point of your module, and the requirements file containing a list of pip packages the module needs to run. The list of requirements must be separated with newlines, and must be available on pip. Place the list into a file called `requirements.txt`.

When the files are in order, one can create the archive:

```
$ tar cvf module.tar main.py requirements.txt
```

To upload the module, go to the admin panel. There one will find a form to submit a module. Select the tape archive created earlier, and give it a name and a version. Consider setting the number of workers to a higher value, and press the upload button. If all goes well, refresh the page. It is now possible to start the module from the menu. Depending on which dependencies are needed, the module might take some time to upload and be converted into a Docker image.

D.5. Best Practices

When writing a module, it is a good idea to put as much of the initialisation code into the Runner scope as possible:

```
do_setup()

with laps.Runner() as runner:
  runner.run(callback)
```

Bad

```
with laps.Runner() as runner:
  do_setup()

  runner.run(callback)
```

Good

The reason for this is that the good code will catch any error that might occur in the `do_setup` function and log the error, allowing one to see what the error is inside the system. If done outside, the module will simply exit with an error, and the only way to figure out why is to look at the Docker logs directly.

D.6. `laps.py` API Reference

The entire file consists of two classes.

D.6.1. The `JobFailure` Exception

This exception is an constitutes a recoverable error. Any other exception will cause the worker to exit. By raising this in the handler callback, the job will be registered as failed, and the module will start listening for a new one. It inherits from `Exception` and can be used how one would expect.

Several internals rely on this exception to work correctly, so it should *never* be caught in module code.

D.6.2. The `Runner` class

The `Runner` class is what communicates with the backend and allows one to write a module. The Redis connection it uses internally can be used from `self.redis`. If one wants to store data in Redis, one must use the helper methods on the class, to avoid name clashing.

`def run(self, handler):`
Connect to the backend and start listening for pathfinding jobs. `handler` is a callback function which is called whenever a job is received. The callback is called with `(self, job)` as arguments. The `job` object is a dict whose fields describe the job.

The handler function must return a list of points which were found in the path. This list has to be a list of dicts with «x» and «y» as fields for every point. The list of points should be a continuous path from the start point to the end point.

`def get_map_data(self, job):`
Get map data as a greyscale PNG, from the map id specified in the `job` object. Because this returns the raw data, a PNG decoder must be used to get the pixel data out. It is recommended to use this method instead of a pure Redis call, because the

`def get_map_metadata(self, job):`
Get the metadata for the map specified in the `job` object. This returns a dict containing the data.

`def create_redis_key(self, name):`
Use `name` to create a Redis key unique to this module. If one wants to use Redis, one should always use a key returned by this method, to prevent name collisions. Note that all workers of a module will get the same keys. If one wants to store data unique to a module, one must include `self.worker_number` in the name parameter.

`log_error, log_info, log_debug, log_warn`
Takes a message and logs it. The log output will be visible in the admin panel, in stdout, and in the server logs. The different methods denote different severities.

E. Pathfinding Module Protocol

In this appendix, we specify the pathfinding module protocol. This is the communication scheme used by the backend and the pathfinding modules to communicate. Read more about the design and rationale in Chapter 7.

All communications use Redis as a message broker[1] and are serialised to JSON[2]

Below, each message is described. An example for each one is provided, in order to get a feel for what the message looks like.

E.1. Registration

Key: `laps.backend.register-module`

```
{
  "name": "some-module",
  "version": "0.0.0"
}
```

When a new pathfinding module is started, it will notify the backend that it exists. It does this by specifying it's own name and version, and pushing this to a list of registration messages.

E.2. Shutdown

Key: `laps.backend.module-shutdown`

```
{
  "name": "some-module",
  "version": "0.0.1"
}
```

The shutdown message should be sent whenever a pathfinding module gets shut down. The only difference between this message and the registration message is the list it is pushed to.

[1] See Section 4.4.2 for a refresher on how this works
[2] See Section 4.7.

E.3. Job Submission

Key: `laps.runner.«name»:«version».work`

```
{
  "job_id": 1,
  "map_id": 2,
  "start": {
    "x": 1,
    "y": 2,
  },
  "stop": {
    "x": 1,
    "y": 2,
  }
}
```

Replace «name» and «version» with the name and version of a pathfinding module. This is the job submission message. It is sent to a pathfinding module by the backend. The job id field is meant for the backend to keep track of which job result is which. The map ID field indicates which map the job is for.

E.4. Job Result

Key: `laps.backend.path-results`

```
{
  "outcome": "success",
  "job_id": 1,
  "path": [
    {"x": 10, "y": 20}, {"x": 11, "y": 25}, {"x": 30, "y": 30}
  ]
}
```

In this message, the path is passed as a list of points with x and y values. It is sent by a pathfinding module when it completes a job. It is vital that the module replies with the job ID it received when it initially received the job. This is used by the backend to tell job results apart.

The outcome field can be one of three different states (in lowercase):

1. Success: The job completed without issue.

2. Failure: An error made this job impossible to complete.
3. Cancelled: Refusing to complete this job for an unknown reason. This is usually sent by the backend when a module has shut down, and it is cancelling all the jobs in that module's queue.

If the outcome is a failure or cancellation, the path field can be absent.

E.5. Logging

Key: `laps.moduleLogs`

Log messages are sent whenever a module logs an event. These are received and stored by the backend such that an administrator can look at the logs later in case there is an issue. The instant field is the UNIX timestamp when the entry was logged. The module field is used to tell logs apart.

```
{
  "message": "Hello, world",
  "level": "info",
  "module": {
    "name": "some-module",
    "version": "1.0.0",
  },
  "instant": 1587635315
}
```

F. Backend REST API

This appendix describes in detail everything there is to know about the REST API exposed by the backend to power the web application.

F.1. Pages Served by the Server

While not technically a part of the API, it is important to know what the server serves. The following pages are served by the server and are meant to be the pages a user navigates to in their browser. Each page has it's own corresponding JavaScript file meant for each page.

- `/`: The index of the application. Meant to hold the pathfinding functionality of the service.
- `/admin`: The admin panel. The user will be redirected to the login page if not logged in.
- `/login`: The login page. The user will be redirected to the admin panel if logged in.

Any images which are part of the frontend are served statically under the `/images` prefix. The served JavaScript files are: `/index.js`, `/admin.js`, and `/login.js`. These are meant to be the output of a JavaScript bundler such as webpack. Any stylesheets must be bundled as part of these files.

F.2. Jobs

`POST /job`

RETURN VALUE
202 Accepted with a base64 encoded token for polling the result.

DESCRIPTION
Submit a job to be run using the given algorithm.

PARAMETERS
JSON serialised values

```
{
  "start": { "x": 0, "y": 0 },
  "stop": { "x": 1, "y": 1 },
  "map_id": 1,
  "algorithm": {
    "name": "sample-algorithm",
    "version": "1.0.0"
  }
}
```

GET /job/«token»

RETURN VALUE
A JSON-encoded list of points on success, 204 No Content when pending. Can also return 503 Service Unavailable if too many clients are already polling.

DESCRIPTION
Poll the result of a pathfinding job.

PARAMETERS
The token given as part of the URL.

EXAMPLE RESULT

```
{
  "points": [
    { "x": 0, "y": 0 }, { "x": 1, "y": 1 }
  ]
}
```

F.3. Maps

GET /maps

RETURN VALUE
A JSON-encoded list of strings.

DESCRIPTION
Get a list of each available set of map data.

PARAMETERS
None.

EXAMPLE RESULT

```
{
  "maps": ["1", "2"]
}
```

GET /map/«ID»

RETURN VALUE
A greyscale PNG image with the map data.

DESCRIPTION
Get an image of the map with a given ID.

PARAMETERS
The ID as part of the URL.

GET /map/«ID»/meta

RETURN VALUE
A JSON-encoded data structure containing the metadata for the given map.

DESCRIPTION
Retrieve the metadata for a given map ID.

PARAMETERS
The ID as a URL segment.

EXAMPLE RESULT

```
{
  "x_res": 1.0,
  "y_res": 1.0,
  "max_height": 300.0,
  "min_height": 100.0,
  "average_height": 200.0
}
```

F.4. Algorithms

GET /algorithms

RETURN VALUE
A JSON-encoded list of available algorithms.

DESCRIPTION
None

PARAMETERS

EXAMPLE RESULT

```
[
  {
    "name": "some-module",
    "version": "0.1.0"
  },
  {
    "name": "some-module",
    "version": "1.0.0"
  }
]
```

F.5. Administration

All administration endpoints apart from POST `/login` require the user to be logged in.

F.5.1. User management

GET /admin/me

RETURN VALUE
Username and super admin status of the currently logged-in user.

DESCRIPTION
Get information about the currently logged-in user.

PARAMETERS
None.

POST /login

RETURN VALUE
204 No Content with a session cookie on success, 403 Forbidden on authentication failure.

DESCRIPTION
Log in to an administrator account.

PARAMETERS
Plain form with keys username and password for username and password respectively.

POST /register

RETURN VALUE
201 Created on success, 403 Forbidden if not authorised.

DESCRIPTION
Register a new administrator. Only available for a super admin, or when no admins have been registered yet.

PARAMETERS
Plain form with username and password, the same as the login endpoint.

F.5.2. Map management

POST /map

RETURN VALUE
An unsigned integer with the ID of the new map.

DESCRIPTION
Add a new map to the system from a GeoTiff file.

PARAMETERS
A multipart form with a field «data» of mime type «image/tiff» containing the GeoTiff data.
EXAMPLE RESULT

```
{
  "username": "admin1",
  "super": 0
}
```

DELETE /map/«ID»

RETURN VALUE
204 No Content on success, 404 Not Found if no such map exists.

DESCRIPTION
Delete a map from the system.

PARAMETERS
The map ID from the URL.

F.5.3. Module management

`POST /module`

RETURN VALUE
201 Created on success.

DESCRIPTION
Upload a module to the system, making available to be launched.

PARAMETERS
A multipart form with the following fields:

- `name`, text field with the name of the uploaded module.
- `version`, text field with the version of the uploaded module.
- `workers`, an optional text field with the number of workers the service can run of this module in parallel. Defaults to 1 it not set.
- `module` the actual tar archive containing the module files. Must be a file field with MIME type `Application/x-tar`.

`GET /module/all`

RETURN VALUE
JSON-serialised list of metadata for all pathfinding modules.

DESCRIPTION
List all pathfinding modules, and their state. State is one of «running», «stopped», «other» «failed». When the state is «failed», the exit code of the module is also returned. See the example. If the state is «other», an additional field, *message* should be displayed instead.

PARAMETERS
None.

EXAMPLE RESULT

```
[
  {
    "state": "running",
    "name": "some-module",
    "version": "0.1.0"
  },
  {
    "state": "failed",
    "exit_code": 1,
    "name": "running-module",
```

```
    "version": "1.0.0"
  }
]
```

GET /module/«name»/«version»/logs

RETURN VALUE
200 OK On success with the log contents as the body in plain text. 404 Not found if the module does not exist.

DESCRIPTION
Get the log output of a pathfinding module as plain text.

PARAMETERS
The name and version of the module as URL parameters.

POST /module/«name»/«version»/restart

RETURN VALUE
204 No Content on restart, 201 Created if a module was previously stopped and has been started. 404 Not Found if module does not exist.

DESCRIPTION
Start or restart a pathfinding module.

PARAMETERS
The name and version of the module as URL parameters.

POST /module/«name»/«version»/stop

RETURN VALUE
204 No Content on success, 400 Bad Request if module was not running.

DESCRIPTION
Stop a pathfinding module, allowing it to complete it's current job before shutting down.

PARAMETERS
The name and version of the module as URL parameters.

`DELETE /module/«name»/«version»`

RETURN VALUE
204 No Content on success, 400 Bad Request if module is running. 404 Not Found it the module does not exist.

DESCRIPTION
Delete a pathfinding module, leaving no trace of it on the server.

PARAMETERS
The name and version of the module as URL parameters.

G. List of User Stories

U.1	As a user, I want to insert start and end points to calculate a path between. I want to see this path directly in the user interface without pop-ups or having to download a report file. It would be convenient if I could access the service from any device, because I'm not pulling out my laptop in the middle of the woods while flying my drone.		
	Features	Related	Origin
	F.1, F.2, F.3		Jan Dyre Bjerknes

U.2	As an administrator, I want to be able to add trained models to the website. I don't want to have to redeploy the service just to add a new model if it can be avoided, so I want to do this through a web interface.		
	Features	Related	Origin
	F.4, F.5, F.6		LAPS

U.3	As a machine learning developer, I want to see the differences between the algorithms and if the flight paths outputted are flyable. This lets me compare the results of my machine learning model with others, as well as letting me know if the model working correctly.		
	Features	Related	Origin
	F.7, F.8, F.9, F.10		Jan Dyre Bjerknes

U.4	As a user, I would like to have the computational time of the model reasonably consistent. I would also like the time to be overall shorter than common pathfinding solutions.		
	Features	Related	Origin
	F.11		Jan Dyre Bjerknes

U.5	As the customer, I want to be able to investigate whether machine-learning-based pathfinding is suitable for my needs.		
	Features	Related	Origin
	F.8, F.11, F.12, F.13		Jan Dyre Bjerknes

H. List of Risks

			Related		
RI.0.0	Category	Group | Website | Machine learning	RI.0.0		
	Description	Example description.	Impact	Prob.	Risk
	Mitigation	Example mitigation	0	0	0

Each risk is assigned an ID in the grey field on the left. The first two letters stand for risk, and the two numbers stand for feature and risk index respectively. Further we have the person responsible for managing this risk. This includes mitigating it and making sure it is taken into consideration when implementing new features. Then we have a description of what the risk is followed by a explanation of how to mitigate it. Last we have a list of related risks and last there is the risk matrix. Each risk is assigned an impact and probability between 0 and 5, then by multiplying them together we get the risk factor. The colour represents the urgency of the risk. Green should be addressed, but isn't urgent. Yellow is urgent, and should be addressed soon as possible. Red is very urgent and should be addressed as fast as possible.

Refer to Figure H.1 for the risk matrix with this setup.

Risk are categorised into the following categories:

1. Group risks
2. Website risks
3. Pathfinding risk

			Related		
RI.1.1	Category	Group			
	Description	A person on the group quits	Impact	Prob.	Risk
	Mitigation	Make sure everybody in group is content	4	1	4

5	5	10	15	20	25
4	4	8	12	16	20
3	3	6	9	12	15
2	2	4	6	8	10
1	1	2	3	4	5
	1	2	3	4	5

Figure H.1.: Risk matrix

RI.1.2			Related		
	Category	Group			
	Description	The electronic kanban board fails	Impact	Prob.	Risk
	Mitigation	Export all information the an external data storage on regular basis	2	2	4

RI.1.3			Related		
	Category	Group			
	Description	A member is sick	Impact	Prob.	Risk
	Mitigation	Arrange for the member to work from home if possible. Attempt to keep all group members up to date on work so a healthy member can take over the sick members work, if needed.	2	4	8

RI.1.4			Related		
	Category	Group			
	Description	Parts of the project is plagiarized	Impact	Prob.	Risk
	Mitigation	Make sure the group members are aware of the consequences of plagiarism. Make sure the report is proof read for plagiarism	5	1	5

RI.1.5			Related		
	Category	Group			
	Description	Work or information is not properly documented	Impact	Prob.	Risk
	Mitigation	Set of more time to documentation. Ask other group member to proof read to make the standard is good	3	3	9

			Related		
RI.1.6	Category	Group			
	Description	Members of the group fail to use the Kanban task board regularly	Impact	Prob.	Risk
	Mitigation	Check the board regularly for tasks that have not moved, to detect infrequent Kanban updates	3	2	6

			Related		
RI.2.1	Category	Website			
	Description	Pathfinding module finds a invalid path.	Impact	Prob.	Risk
	Mitigation	Implement path validation into service to make sure path is flyable.	2	2	4

			Related		
RI.2.2	Category	Website			
	Description	User inputs invalid data	Impact	Prob.	Risk
	Mitigation	Do user input validation on backend and frontend.	2	2	4

			Related		
RI.2.3	Category	Website			
	Description	The webiste host server can't keep the server online all the time	Impact	Prob.	Risk
	Mitigation	Change host or host the website ourself.	1	2	2

			Related		
RI.2.4	Category	Website			
	Description	User connection is broken midway through interaction	Impact	Prob.	Risk
	Mitigation	Save session data and give on reconnect.	2	2	4

			Related		
RI.3.1	Category	Machine learning			
	Description	Training server is not powerful enough to train our models	Impact	Prob.	Risk
	Mitigation	Make sure we have adequate time to train models, and consider acquiring more powerful server, or reduce complexity of models.	3	2	6

			Related		
RI.3.2	Category	Machine learning			
	Description	Pathfinding time estimate is consistently off.	Impact	Prob.	Risk
	Mitigation	Keep track of pathfinding time estimate accuracy.	1	3	3

			Related		
RI.3.3	Category	Machine Learning			
	Description	The model does not converge on a solution to the problem.	Impact	Prob.	Risk
	Mitigation	Simplify problem domain.	3	3	9

			Related		
RI.3.4	Category	Machine Learning			
	Description	Unexpected results in edge cases.	Impact	Prob.	Risk
	Mitigation	Test models in multiple scenarios and detect any anomalies.	2	4	8

			Related		
RI.3.5	Category	Machine Learning			
	Description	Problem is too complex to solve with our limited experience and time	Impact	Prob.	Risk
	Mitigation	Focus on simple representations of the problem that give meaningful results, instead of trying to focus on solving the problem immidiately.	5	3	15

			Related		
RI.3.6	Category	Machine Learning			
	Description	Model overfitting	Impact	Prob.	Risk
	Mitigation	Do frequent error function tests on validation data, and stop training if it gets worse.	3	4	12

			Related		
RI.3.7	Category	Machine Learning			
	Description	Not enough data for machine learning	Impact	Prob.	Risk
	Mitigation	Prepare data early in model investigations, to ensure you have enough time to generate and prepare data	4	3	12

			Related		
	Category	Machine Learning			
RI.3.8	Description	Multiple people working in the same directory on the same computer may cause conflicts for eachother, especially when doing git version control.	Impact	Prob.	Risk
	Mitigation	Setup multiple development environments when working on server.	2	3	6

			Related		
	Category	Machine Learning			
RI.3.9	Description	Unbalanced dataset used to train model, resulting in portions of the output never being trained properly	Impact	Prob.	Risk
	Mitigation	Make sure the dataset is balanced, otherwise employ methods such as upsampling, downsampling, or SMOTE	3	3	9

I. List of Tests

T.0.0.0	Requirements	RQ.0.0	Related	T.0.0.0
Untested	Description	Example description		
Not Done	Criteria	Example criteria		

T.1.1.1	Requirements	RQ.1.1	Related	
Passing	Description	Test if markers work		
Done	Criteria	Click on the map, and a marker appear. Attempt to click more than the limited amount of marker, and check if you are stopped.		

T.1.2.1	Requirements	RQ.1.2	Related	
Passing	Description	Test if the user can manually input coordinates		
Done	Criteria	Identify the input fields. Insert a random value in all the field. Press the create path button. If the values are invalid an error message is displayed. If the values are valid, no error appears.		

T.1.3.1	Requirements	RQ.1.3	Related	
Untested	Description	Test if markers can be deleted		
Not Done	Criteria	Select a placed marker in the marker menu, press the delete button. Run a test and make sure the marker is not counted		

T.1.4.1	Requirements	RQ.1.4	Related	
Untested	Description	Test that markers can be marked as either start or stop for the path		
Not Done	Criteria	Have at least to waypoints placed on the map. Go to the marker menu and mark one as start and another stop, do a test run and make sure the service uses the correct marker as start and the correct marker as stop.		

T.2.1.1	Requirements	RQ.2.1	Related	
Passing	Description	Test that the service works with different models		
Done	Criteria	Do a test run in one model, then another, if no problems occur the test is passed		

T.2.1.2	Requirements	RQ.2.1	Related	
Passing	Description	Verify that the backend can properly submit jobs and get their result.		
Done	Criteria	Run the unit test for checking the submit job endpoint and the get result endpoint.		

T.2.1.3	Requirements	RQ.2.1	Related	
Passing	Description	Verify that job submission and polling rate-limiting works.		
Done	Criteria	Run the unit test verifying the rate-limiting.		

T.2.2.1	Requirements	RQ.2.2	Related	
Passing	Description	Test if the service can generate a path between two points		
Done	Criteria	Use a test model that draws a straight line between the two points. Make sure a line is created		

T.2.3.1	Requirements	RQ.2.3	Related	
Untested	Description	Test that user is warned if the calculation time is over 5 minutes		
Not Done	Criteria	Run a test model that will never finish, see if the user is warned		

T.2.4.1	Requirements	RQ.2.4	Related	
Untested	Description	Test if the user can save/export a path made by a model		
Not Done	Criteria	Attempt to export a path and run it on a secondary computer or a new instance of the service		

T.2.5.1	Requirements	RQ.2.5	Related	
Untested	Description	Test that the user can access a description of a model		
Not Done	Criteria	After selecting a model make sure a information panel can be opened to provide necessary information		

T.2.6.1	Requirements	RQ.2.6	Related	
Untested	Description	Test if the service shows an estimate of time		
Not Done	Criteria	As time is based upon the module, its only required that a time is shown at all, make sure that a time is present somewhere on the screen after a model as been run		

T.2.7.1	Requirements	RQ.2.7	Related	
Untested	Description	Test that the paths the models make are at least one meter from anything in the height map		
Not Done	Criteria	Generate at least five different paths and manually check that it is at least one meter away from anything from in the height map		

T.3.1.1	Requirements	RQ.3.1	Related	
Passing	Description	Test that the path is visible inside the map		
Done	Criteria	Generate at least five different paths and visually check that they are inside the map		

T.3.1.2	Requirements	RQ.3.1	Related	
Passing	Description	Verify that the backend can return a list of map data and retrieve the data.		
Done	Criteria	Run the unit test for checking the maps list endpoint and get map endpoint.		

T.3.2.1	Requirements	RQ.3.2	Related	
Untested	Description	Test that the service works on mobile devices		
Not Done	Criteria	Open the service on a mobile device. Test that all menus can be opened. Click various buttons, to assure they are not small and unresponsive. Place two markers on the map, calculate a path.		

T.3.3.1	Requirements	RQ.3.3 done	Related	
Untested	Description	Make sure heights are displayed in the map		
Not Done	Criteria	visually see if height data is displayed. Compare heights with another map the make sure the data is correct		

T.3.4.1	Requirements	RQ.3.4	Related	
Untested	Description	Test the service is usable on a mobile device		
Not Done	Criteria	Open the service on a mobile device. Test that all menus can be opened. Click various buttons, to assure they are not small and unresponsive. Place two markers on the map, calculate a path.		

T.4.1.1	Requirements	RQ.4.1	Related	
Passing	Description	Backend unit test testing that modules can be uploaded and are listed correctly.		
Done	Criteria	Log in as an administrator, and use the module upload endpoint to upload the test module. Verify that the list of returned modules contain that module, and that it is not yet running. Upload a module which exits immediately with a failure, and start it. Verify that the list of modules contain both modules where one has failed and the other is stopped.		

T.4.2.1	Requirements	RQ.4.2	Related	
Passing	Description	Backend unit test to verify that modules can be started, restarted and stopped.		
Done	Criteria	Log in as an administrator, and upload the test module with the upload module endpoint. Verify that it is not currently running. Start it with the restart module endpoint and check that the status code is correct and that the module is running. Restart it again, this time check that the status code indicating a restart was returned, and check that the module is still running. Now stop the module with the stop endpoint and verify that it was stopped. Start it back up, and verify that it is running. Finally, stop it twice, and verify that an error is returned when trying to stop it when it isn't running.		

T.4.3.1	Requirements	RQ.4.3	Related	T.4.1.1
Passing	Description	Backend unit test to verify that modules can be deleted		
Done	Criteria	Log in as an administrator, upload the test module and start it. Try to delete it, which should fail. Stop it, and then use the module deletion endpoint to remove the module. Check that there are no containers with the module image, and that the module image is removed. Verify that there are no remnants of the module in the database.		

T.4.5.1	Requirements	RQ.4.5	Related	
Passing	Description	Backend unit test to verify that registration works.		
Done	Criteria	Use the register endpoint to verify that the initial super admin can be registered. Try to register a new admin without signing in, which should fail. Sign in as the super admin, and register a new admin. Verify that the created admin is not a super admin. Try to create another admin with the same username, which should fail. Try to create admins with too long and too short passwords, which should fail. Log in with the non-super admin registered earlier, and try to register an admin, which should fail.		

T.4.6.1	Requirements	RQ.4.6	Related	
Passing	Description	Backend unit test for adding mapdata.		
Done	Criteria	Log in as an administrator. Use the map upload endpoint to try to upload an invalid image, which should fail. Upload a valid map and check that it gets the ID of 1. Upload another and check that the ID is 2. Use the deletion endpoint to delete one of the maps. Verify that no trace of it remains in the database.		

T.4.7.1	Requirements	RQ.4.7	Related	
Passing	Description	Backend unit test for module logs.		
Done	Criteria	Log in as an administrator, upload the test module, and start it. Any module sends a log message when it registers itself. Use the get logs endpoint to check for this message in the module logs.		

T.5.1.1	Requirements	RQ.5.1	Related	
Passing	Description	Backend unit test to check that login works		
Done	Criteria	Register an administrator. Try to sign in as a user which does not exist, and again with the wrong password, where both should fail. Then log in with the correct password. Verify that we got a session cookie back.		

T.5.5.1	Requirements	RQ.5.5	Related	
Untested	Description	Test that admin features are placed behind password authentication		
Not Done	Criteria	Attempt to access admin features without being logged in, make sure the user can't access them. As admin access the admin features, make sure they are available		

T.6.1.1	Requirements	RQ.6.1	Related	
Passing	Description	Test that pathfinding module have access to height data		
Done	Criteria	Pathfinding modules are able to connect to the Redis database, and get map data from the laps.mapdata.image and laps.mapdata.meta keys.		

T.10.1.1	Requirements	RQ.10.1	Related	
Untested	Description	Test the path created from a model is flyable		
Not Done	Criteria	Create a test model that draw a straight line between the start and end point, make sure you get an error. Then run a proper model and make sure the user gets no errors. Visually check that the error messages are given appropriately		

T.11.1.1	Requirements	RQ.11.1	Related	
Untested	Description	Test that the computing time varies less than 10%		
Done	Criteria	Run at least 20 different paths inside a model, check time shortest and longest are within 10%		

T.11.2.1	Requirements	RQ.11.2	Related	
Passing	Description	Test that the time doesn't grow exponential with time		
Done	Criteria	Run a model in 10 map sizes, make sure the doesn't grow exponentially larger		

T.11.3.1	Requirements	RQ.11.3	Related	
Untested	Description	Test that map complexity doesn't effect computing time		
Done	Criteria	Run at least 10 different maps with the same size, all results should be within 10%		

T.12.1.1	Requirements	RQ.12.1, RQ.13.1	Related	
Untested	Description	Test that the resulting path is sent to the correct Redis channel		
Not Done	Criteria	Resulting path is sent to correct Redis key, depending on whether the model is deployed or in staging.		

T.12.1.2	Requirements	RQ.12, RQ.13	Related	
Untested	Description	Test that resulting path is in correct format		
Not Done	Criteria	Resulting path adheres to JSON schema.		

T.12.2.1	Requirements	RQ.12	Related	
Untested	Description	Test Reinforcement Learning Environment		
Not Done	Criteria	Perform gym-drone unit tests.		

T.13.1.1	Requirements	RQ.12.1, RQ.13.1	Related	
Passing	Description	Test that the resulting path is sent to the correct Redis channel		
Done	Criteria	Resulting path is sent to correct Redis key, depending on whether the model is deployed or in staging.		

T.13.1.2	Requirements	RQ.12, RQ.13	Related	
Passing	Description	Test that resulting path is in correct format		
Done	Criteria	Resulting path adheres to JSON schema.		

J. List of Requirements

ID	Category	Feature	Page
RQ.1.X	UI	Select waypoints start and stop waypoints.	2
RQ.2.X	UI,Backend	Path calculation between two waypoints.	2-3
RQ.3.X	UI	Map display with path.	3
RQ.4.X	UI,Backend	Remote access admin panel.	3-4
RQ.5.X	Backend	Admin authorisation.	4
RQ.6.X	Pathfinding	API for pathfinding modules.	4
RQ.7.X	Backend	Select multiple pathfinding algorithms.	5
RQ.8.X	UI	Pathfinding module statistics comparing.	5
RQ.9.X	Backend	Jobs panel.	5-6
RQ.10.X	Backend	Valid path checker.	6
RQ.11.X	Pathfinding	Machine learning-based pathfinding.	6-7
RQ.12.X	Pathfinding	Investigate deep Q-learning as a solution for pathfinding.	7
RQ.13.X	Pathfinding	Investigate convolutional neural networks as a pathfinding solution.	8

RQ.0.0	User stories	U.0	Tests	T.0
Not Done	Category	Example Category		
Priority	Origin	Example Origin		
0	Description	This is a example description		

Each requirement is assigned an ID that is shown in the grey field in the top left. The first two letters is always RQ and stands for requirement, following that is two number that comes from the parent feature id and its index within that feature respectively. Below that there is a status field that currently can be in three states Done, Not Done and Discarded. In the bottom left there is a priority. Priority ranges from 1 to 10, where 1 is the most important.

In the top row you can view what user stories this requirement came from and what test is derived from this requirement. Below that is the category for the requirement and last there is a field for where the requirement originated from and a shortened description of what the requirement is. For a more detailed description and documentation if the requirement is completed see the feature technical documentation.

RQ.1.1	User stories	U.1	Tests	T.1.1.1
Done	Category	UI		
Priority	Origin	LAPS		
1	Description	The user must be able to place two markers on a map.		

RQ.1.2	User stories	U.1	Tests	T.1.2.1
Discarded	Category	UI		
Priority	Origin	LAPS		
5	Description	The user must be able to place markers by typing in coordinates.		

RQ.1.3	User stories	U.1	Tests	T.1.3.1-2
Done	Category	UI		
Priority	Origin	LAPS		
2	Description	The user must be able to move and change waypoints.		

RQ.2.1	User stories	U.1	Tests	T.2.1.1
Done	Category	General		
Priority	Origin	LAPS		
1	Description	The user must be able to find a path between two points.		

RQ.2.2	User stories	U.1	Tests	T.2.2.1
Discarded	Category	UI, Backend		
Priority	Origin	LAPS		
5	Description	The user must be shown an estimated processing time for the algorithm.		

RQ.2.3	User stories	U.1	Tests	T.2.3.1
Discarded	Category	UI, Backend		
Priority	Origin	LAPS		
7	Description	The user must be warned if the estimated processing time exceeds 5 minutes.		

RQ.2.4	User stories	U.1	Tests	T.2.4.1
Not Done	Category	UI		
Priority	Origin	LAPS		
6	Description	The user must be able to export a calculated path as a file.		

RQ.2.5	User stories	U.1	Tests	T.2.5.1
Done	Category	UI, Backend		
Priority	Origin	LAPS		
2	Description	The user must be able to see the available pathfinding algorithms		

RQ.2.6	User stories	U.1	Tests	T.2.6.1
Done	Category	UI, Backend		
Priority	Origin	LAPS		
2	Description	The user must be able to select their desired algorithm.		

RQ.2.7	User stories	U.1	Tests	T.2.7.1
Not Done	Category	Pathfinding		
Priority	Origin	LAPS		
4	Description	The calculated path must be at least one meter above the heightmap.		

RQ.3.1	User stories	U.1	Tests	T.3.1.1-2
Done	Category	UI		
Priority	Origin	LAPS		
1	Description	The resulting path must be displayed on the map, at least partially.		

RQ.3.2	User stories	U.1	Tests	T.3.2.1
Not Done	Category	UI		
Priority	Origin	LAPS		
5	Description	The website must be well-behaved on mobile devices.		

RQ.3.3	User stories	U.1	Tests	T.3.3.1
Not Done	Category	UI		
Priority	Origin	LAPS		
1	Description	Height data for a calculated path must be shown to the user.		

RQ.3.4	User stories	U.1	Tests	T.3.4.1
Not Done	Category	UI		
Priority	Origin	LAPS		
4	Description	Every feature must be usable with a touch screen.		

RQ.4.1	User stories	U.2	Tests	
Done	Category	UI, Backend		
Priority	Origin	Jan Dyre Bjerknes		
3	Description	An administrator must be able to see an overview of pathfinding modules.		

RQ.4.2	User stories	U.2	Tests	
Done	Category	UI, backend		
Priority	Origin	Jan Dyre Bjerknes		
3	Description	An admin must be able to start and stop pathfinding modules.		

RQ.4.3	User stories	U.2	Tests	
Done	Category	UI, backend		
Priority	Origin	Jan Dyre Bjerknes		
3	Description	An admin must be able to upload new pathfinding modules, and delete existing ones.		

RQ.4.4	User stories	U.2	Tests	
Not Done	Category	UI, backend		
Priority	Origin	LAPS		
4	Description	The admin must be able to view and change the settings of pathfinding modules, if there are any		

RQ.4.5	User stories	U.2	Tests	
Done	Category	UI, backend		
Priority	Origin	LAPS		
7	Description	A super admin must be able to add other administrators.		

RQ.4.6	User stories	U.2	Tests	
Done	Category	Backend, UI		
Priority	Origin	LAPS		
3	Description	Administrators must be able to add new map data to the system, and delete them.		

RQ.4.7	User stories	U.2	Tests	
Done	Category	Backend, UI		
Priority	Origin	LAPS		
3	Description	Administrators must be able to view module logs.		

RQ.5.1	User stories		Tests	
Done	Category	UI, Backend		
Priority	Origin	LAPS		
3	Description	Administrators should be authenticated using a password.		

RQ.5.2	User stories		Tests	
Done	Category	Backend		
Priority	Origin	LAPS		
4	Description	Password authentication must use a modern cryptographic hashing algorithm		

RQ.5.3	User stories		Tests	
Done	Category	Backend		
Priority	Origin	LAPS		
5	Description	The password must be any valid UTF-8 string.		

RQ.5.4	User stories		Tests	
Done	Category	Backend, UI		
Priority	Origin	LAPS		
7	Description	The password cannot be longer than 128 bytes.		

RQ.5.5	User stories		Tests	
Done	Category	Backend		
Priority	Origin	LAPS		
3	Description	Only registered administrators are allowed to access administration features.		

RQ.6.1	User stories	U.1	Tests	T.6.1.1
Done	Category	Backend		
Priority	Origin	LAPS		
1	Description	Height data must be available across the system		

RQ.6.2	User stories	U.1	Tests	
Done	Category	Backend		
Priority	Origin	LAPS		
1	Description	The backend must be able to get paths from pathfinding modules		

RQ.6.3	User stories	U.1	Tests	
Done	Category	Backend		
Priority	Origin	LAPS		
1	Description	Pathfinding modules must get start and end points, as well as map ID, from the system.		

RQ.7.1	User stories	U.3	Tests	
Not Done	Category	Backend, UI		
Priority	Origin	Jan Dyre Bjerknes		
5	Description	The user must be able to choose at least 2 pathfinding algorithms, and display the result from these at the same time.		

RQ.7.2	User stories	U.3	Tests	
Not Done	Category	UI		
Priority	Origin	Jan Dyre Bjerknes		
8	Description	The user must be able to see statistics of each path from the different algorithms		

RQ.7.3	User stories	U.3	Tests	
Not Done	Category	UI		
Priority	Origin	LAPS		
8	Description	The user must be able to toggle visibility of each result separately.		

RQ.8.1	User stories	U.3	Tests	
Not Done	Category	Backend		
Priority	Origin	LAPS		
8	Description	The service must be able to generate random waypoints and run pathfinding algorithms on these.		

RQ.8.2	User stories	U.3	Tests	
Not Done	Category	Backend, UI		
Priority	Origin	LAPS		
9	Description	The user must be able to choose a sample size.		

RQ.8.3	User stories	U.3	Tests	
Not Done	Category	UI		
Priority	Origin	LAPS		
5	Description	The user must be able to view and compare statistics of run samples between the algorithms.		

RQ.8.4	User stories	U.3	Tests	
Not Done	Category	UI		
Priority	Origin	LAPS		
8	Description	The user must be able to export statistics data.		

RQ.9.1	User stories	U.1, U.3	Tests	
Not Done	Category	Backend		
Priority	Origin	LAPS		
4	Description	The back-end must be able to queue tasks.		

RQ.9.2	User stories	U.1, U.3	Tests	
Not Done	Category	UI, Backend		
Priority	Origin	LAPS		
5	Description	The user must be able to view status of tasks which they have in the queue.		

RQ.9.3	User stories	U.1, U.2, U.3	Tests	
Not Done	Category	UI, backend		
Priority	Origin	LAPS		
5	Description	An administrator must be able to view the status of each task.		

RQ.9.4	User stories	U.1, U.3	Tests	
Not Done	Category	UI, backend		
Priority	Origin	LAPS		
5	Description	The user must be able to cancel queued tasks.		

RQ.10.1	User stories		Tests	
Not Done	Category	Backend		
Priority	Origin	LAPS		
3	Description	The service must check if the path is flyable.		

RQ.10.2	User stories		Tests	
Not Done	Category	UI		
Priority	Origin	LAPS		
4	Description	The UI must highlight the unflyable parts of a path if it is unflyable.		

RQ.11.1	User stories	U.4	Tests	
Not Done	Category	Backend		
Priority	Origin	LAPS		
6	Description	The computing time must vary less than 10%.		

RQ.11.2	User stories	U.4	Tests	
Not Done	Category	Backend		
Priority	Origin	LAPS		
6	Description	The computing time must not grow exponentially with grid size		

RQ.11.3	User stories	U.4	Tests	
Done	Category	Backend		
Priority	Origin	LAPS		
6	Description	Computing time cannot vary based on map complexity		

RQ.11.4	User stories	U.4	Tests	
Not Done	Category	Backend		
Priority	Origin	LAPS		
6	Description	The calculation time for finding a path must be faster than that of traditional algorithms		

RQ.12.1	User stories	U.5	Tests	T.12.1.1-2
Not Done	Category	Pathfinding		
Priority	Origin	LAPS		
2	Description	Return the resulting path as a list of points.		

RQ.12.2	User stories	U.5	Tests	T.12.2.1
Not Done	Category	Pathfinding		
Priority	Origin	LAPS		
2	Description	Calculate path using deep Q-learning, with states as possible movement.		

RQ.12.3	User stories	U.5	Tests	
Not Done	Category	Pathfinding		
Priority	Origin	LAPS		
2	Description	The Machine Learning model shall be in ONNX format.		

RQ.12.4	User stories	U.5	Tests	
Not Done	Category	Pathfinding		
Priority	Origin	LAPS		
3	Description	Path takes consideration to drone stats.		

RQ.12.5	User stories	U.5	Tests	
Not Done	Category	Pathfinding		
Priority	Origin	LAPS		
2	Description	The module must be API compatible.		

RQ.13.1	User stories	U.5	Tests	
Done	Category	Pathfinding		
Priority	Origin	LAPS		
2	Description	Take a heightmap, start and stop as an input.		

RQ.13.2	User stories	U.5	Tests	
Done	Category	Pathfinding		
Priority	Origin	LAPS		
2	Description	Approximate the midpoint of the most efficient path, using convolutional neural networks.		

RQ.13.3	User stories	U.5	Tests	
Not Done	Category	Pathfinding		
Priority	Origin	LAPS		
2	Description	The Machine Learning model shall be in ONNX format.		

RQ.13.4	User stories	U.5	Tests	
Not Done	Category	Pathfinding		
Priority	Origin	LAPS		
3	Description	Path takes consideration to drone stats.		

RQ.13.5	User stories	U.5	Tests	
Done	Category	Pathfinding		
Priority	Origin	LAPS		
2	Description	The module must be API compatible.		

K. List of Features

F.1	Requirements	RQ.1.1, RQ.1.2, RQ.1.3
	Description	Place start and stop waypoints
	Userstories	U.1

F.2	Requirements	RQ.2.1-2.7
	Description	Path calculation between two waypoints
	Userstories	U.1

F.3	Requirements	RQ.3.1, RQ.3.2, RQ.3.3, RQ.3.4
	Description	Map display with path
	Userstories	U.1

F.4	Requirements	RQ.4.1-RQ.4.6
	Description	Remote access admin panel.
	Userstories	U.2

F.5	Requirements	RQ.5.1-RQ.5.5
	Description	Admin Authorization.
	Userstories	U.2

F.6	Requirements	RQ.6.1-6.3
	Description	API for pathfinding modules.
	Userstories	U.2

F.7	Requirements	RQ.7.1-RQ.7.3
	Description	Select multiple pathfinding algorithms.
	Userstories	U.3

F.8	Requirements	RQ.8.1-RQ.8.4
	Description	Pathfinding module statistics comparison.
	Userstories	U.3

F.9	Requirements	RQ.9.1-RQ.9.4
	Description	Jobs panel.
	Userstories	U.3

F.10	Requirements	RQ.10.1-RQ.10.2
	Description	Valid path checking.
	Userstories	U.3

F.11	Requirements	RQ.11.1-RQ.11.4
	Description	Machine learning-based pathfinding
	Userstories	U.4

F.12	Requirements	RQ.12.1-RQ.12.5
	Description	Investigate deep Q-learning as a solution for pathfinding.
	Userstories	U.5

F.13	Requirements	RQ.13.1-RQ.13.5
	Description	Investigate convolutional neural networks as a solution for pathfinding.
	Userstories	U.5

Meeting

LAPS Group

Ext. Supervisor Week 2 Meeting

Participants	Even Thonhaugen Røraas Henrik Thue Strocka Håkon Jordet Vetle André Hoffmeyer Neumann Jan Dyre Bjerknes
Recipients	Jan Dyre Bjerknes
Secretary	Vetle André Hoffmeyer Neumann
Location	2263
Time of meeting	27th January 2020 at 12:00

27th January 2020

Discussion

KDA - KDA are interested in the problem and our investigation on it, and they'll be the contractors for the project. The team may in the future be invited to have a presentation about the project at KDA.
Servers - The server plan as of now is to use two different servers, one for hosting the web service itself, and another more powerful server for training the machine learning models. For the web service server a remote renting solution with Linux was discussed, whereas for the other one there is a server located at the school which External Supervisor will investigate if we may use this one for the training.
Budget - Budget for things that are not the server was discussed and External Supervisor will come with a status update later.
Meeting with Kongsberg Digital - External Supervisor will talk to Kongsberg Digital in the near future to possibly set up a meeting regarding machine learning and frameworks.
Simplification - Details that are small regarding the problem may be ignored to focus on the core issue instead, as long as the impact is estimated and the decision is reasoned for.
Service Map Area - Having all of NMA's heightmaps were not a requirement, instead the service should offer a few maps on predefined areas for the user to choose between. The size of the maps was also discussed as too big maps could serve as a problem for the implementation, External Supervisor stated that there was no need for maps bigger than approximately a kilometer, so this was noted as a soft requirement to the service.
What Determines a Good Path - This topic was brought up and lightly discussed but the topic was postponed for now and will be brought up again once we are deeper into the project where it is more relevant.
Expo Presentation Idea - Our Expo Presentation Idea with the Augmented Reality Sandbox at Devotech was brought up and External Supervisor was fond of the idea, and he came up with ideas for flying the generated path in VR, and using a mold of Kongsberg on the sandbox to shape it like Kongsberg.

Meeting

LAPS Group

Meeting with internal supervisor

Participants	Henrik Thue Strocka Håkon Jordet Vetle André Hoffmeyer Neumann Even Thonhaugen Røraas Henning Gundersen
Recipients	Henning Gundersen
Secretary	Håkon Jordet
Location	Online
Time of meeting	28th April 2020 at 14:00

28th April 2020

Discussion

0.1 What we are working on

- Henrik - Graph networks, back to basics, creating a better loss function.
- Vetle - Simplified the problem. Implemented normal Q-learning to verify that it works. By simplifying the problem the training is faster and it's possible to verify that it actually works.
- Håkon - Very basic admin panel UI, focus on documentation, fixing bugs.
- Even - Admin panel work was cut off for a bit, started working on more interactive map.

0.2 Feedback on backend implementation documentation

We received feedback on some of the documentation which we had written. It was generally pretty good, but there were a few things we still have to work on.

Meeting

LAPS Group

Meeting with Ext. Supervisor 4

Participants	Håkon Jordet Vetle André Hoffmeyer Neumann Even Thonhaugen Røraas Jan Dyre Bjerknes
Recipients	Jan Dyre Bjerknes
Secretary	Håkon Jordet
Location	2263
Time of meeting	5th February 2020 at 12:00

5th February 2020

Discussion

Our intended main point of discussion for this meeting was our presentation. However, because it was postponed we didn't get to talk about this.

We discussed the budget for the project. We will have to pay for miscellaneous small items such as snacks in meetings out of pocket.

Meeting with Kongsberg Digital

Finally, we talked about our meeting with Kongsberg Digital on Monday. The best way of presenting our problem to them was discussed. Again, we do not have to make a perfect presentation, we just have to explain our problem well enough to be understood. We should not be afraid to ask questions, since they do not expect us to be fully educated engineers yet.

We need to remember to talk about our architecture questions because it is very relevant to their experience in deploying machine-learning solutions.

Meeting

LAPS Group

Meeting with internal supervisor

Participants	Henrik Thue Strocka Håkon Jordet Vetle André Hoffmeyer Neumann Even Thonhaugen Røraas Henning Gundersen
Recipients	Henning Gundersen
Secretary	Håkon Jordet
Location	2263
Time of meeting	2nd March 2020 at 11:15

9nd March 2020

Discussion

0.1 Process Model

Due to the freedom we are given, we no longer feel there is a need for a replenishment meeting with our customer every week as of now, so we will be dropping that from our process model. We will continue to have a meeting with our customer every week but it will be more of a general meeting. We also introduced our new process for finalizing feature documentation.

0.2 Documentation

We discussed the option of having a numbered version on all feature documentation to indicate how complete that document is. Version 1.0 being when the document has passed the finalize documentation stage of our process model. Further it was discussed about having background information in an appendix to be handed in with the paper.

0.3 Webpage

We discussed our plans for the webpage and that we should set it up by the second presentation so people can view our project easily. It does not need to be anything fancy just some text about us and what we do, and it has to be dynamic.

0.4 Future meeting

We discussed when to do the next meeting as the next two weeks will be put aside for exam preperations. Two dates was possible 19th or 13th, friday the 13th was decide as the date for our next meeting.

Meeting

LAPS Group

Meeting with Ext. Supervisor 8

Participants	Håkon Jordet Vetle André Hoffmeyer Neumann Even Thonhaugen Røraas Jan Dyre Bjerknes
Recipients	Jan Dyre Bjerknes
Secretary	Vetle André Hoffmeyer Neumann
Location	Skype
Time of meeting	17.04.2020 at 14:00

24th May 2020

Discussion

The groups progress war borugh up and vetle was having some issues with his optimazation algorithm for reinforcement learning. Potential solutions was discussed.

Even was asked about dropdown menus and how they are created.

Further discussion about modification to the dijkstra algorithm to make it more realistic behaving, and discussion about new ways to train convolutional neural networks.

Timeframe for the project was discussed.

Meeting

LAPS Group

Meeting with Ext. Supervisor 4

Participants	Håkon Jordet Vetle André Hoffmeyer Neumann Even Thonhaugen Røraas Jan Dyre Bjerknes
Recipients	Jan Dyre Bjerknes
Secretary	Håkon Jordet
Location	Skype
Time of meeting	at 13:00

20th of march

Context

The last week and a half the group have practiced for upcoming exams, however these where cancelled because of corona, as such work have instead resumed on the project. We expect this to last for a bit and for the unforeseeable future all meetings will be online rather than in person. We are also unsure when exams will be held.

Presentation 2

We discussed how we planned to do our presentation 2 as meeting in person is not longer an viable option. As all documentation have to be turned in by tuesday, the group effort towards this.

Server

We informed Jan that we contacted Dag to get the testing server working. However it seems like something have happened and we can no longer access it. Because of the corona we can not freely enter krona to figure out what is wrong. Jan have proposed, as he have access to krona could let us in.

progress

Since last meeting most work have been done towards exams, however since they where cancelled more free time was available

- Vetle- Continued work towards reinforcement learning
- Even - Finishing minimal viable product for the frontend
- Henrik - Finalize documentation for 2. presentation
- Håkon - Finalize documentation for 2. presentation

Meeting

LAPS Group

Meeting with internal supervisor

Participants	Henrik Thue Strocka Håkon Jordet Vetle André Hoffmeyer Neumann Even Thonhaugen Røraas Henning Gundersen
Recipients	Henning Gundersen
Secretary	Håkon Jordet
Location	Online
Time of meeting	30th March 2020 at 14:00

30th March 2020

Discussion

0.1 What we are working on

- Henrik -Working on graph network
- Vetle - Working on reinforcement learning
- Håkon - Working on caching
- Even - Working on documentation for frontend

0.2 Feedback on the presentation

This is the first meeting after our second presentation. We talked about some of the things that could have been better. Our documentation could be better, and we should have focused more about what we have done so far.

0.3 Expo

We talked about expo. Unsure if expo is fully canceled or is made into a digital expo". We don't know yet.

0.4 Documentation

Documentation needs to be improved, especially code documentation.

0.5 Working digitally

We talked about working digital and some of the challenges.

Meeting

LAPS Group

Meeting with Ext. Supervisor 6

Participants	Håkon Jordet Vetle André Hoffmeyer Neumann Even Thonhaugen Røraas Jan Dyre Bjerknes
Recipients	Jan Dyre Bjerknes
Secretary	Håkon Jordet
Location	Skype
Time of meeting	03.04.2020 at 12:00

24th May 2020

Discussion

This meeting was our first since the second presentation. As every member of the group has made good progress, we discussed our progress.

We talked about Henrik's experiments with graph networks. An issue was raised that working on graphs will have the same problems as the traditional pathfinding algorithms have. However, we believe that it can be circumvented by using a special kind of network which generates a path which includes all the nodes, instead of a path from node to node.

Vetle was able to show off his work on a reinforcement-learned algorithm, displaying the paths he's been able to generate thus far. An issue was raised considering that the algorithm might be trying to optimise for two potentially incompatible variables at once. Because of this, we want to have meetings more often to discuss our progress.

Even explained that he has kept on documenting and improving his work.

Håkon talked about having implemented the suggestions he received after the second presentation: submitting the same job twice will now return the result from a cache instantly, and validating job input. Moreover there was talk about how the admin panel feature is coming together, and how it is now possible to upload a GeoTiff file directly into the backend and have it deal with the conversion.

Afterwards we discussed the GUI a little bit more. There was a discussion on features which are desired in the frontend, in particular being able to see the height of the generated path from the side. Another feature which was discussed was to display various pieces of data about the path and map in the frontend, which requires backend tweaks as well.

Meeting

LAPS Group

Meeting with internal supervisor

Participants	Henrik Thue Strocka Håkon Jordet Vetle André Hoffmeyer Neumann Even Thonhaugen Røraas Henning Gundersen
Recipients	Henning Gundersen
Secretary	Håkon Jordet
Location	2263
Time of meeting	27th January 2020 at 11:15

27th January 2020

Discussion

Process model

Exact details of our project model do not matter that much. What matters is that the model as a whole works for us and is well-documented.

Documentation

- It's better to document too much than too little.
- We must document our choices for the first presentation.
- In the weekly reports, it should be visible who did what.
- We should document how we handle administrative matters. If something goes wrong, we can backtrack on it and explain how it happened.

The binder

The binder should contain every piece of documentation which we think is important to read. Any other pieces of documentation should be placed on the flash drive, and might not be read. It should have a front page with our logo and a presentation of us. Having some kind of contents listing is also a good idea. It should contain an overview of the flash drive as well, to make it easy to find documents.

The presentation

- Appear confident.
- Make the room focus on us.
- We need to show that we have a good understanding of the project.
- Should talk about what we are going to work on next.

Meeting

LAPS Group

Meeting with internal supervisor

Participants	Henrik Thue Strocka Håkon Jordet Vetle André Hoffmeyer Neumann Even Thonhaugen Røraas Henning Gundersen
Recipients	Henning Gundersen
Secretary	Even Thonhaugen Røraas
Location	Online
Time of meeting	15th May 2020 at 13:00

15th May 2020

Discussion

0.1 What we are working on

- Al members are currently working on documentation.

0.2 Things we worked on

Our documentation is generally good, but we need to get everything documented.

We are still lacking figures and sources.

We should also remember to make clear who did what. As we are only four and the task is relative separated, it should be to hard to do but still important.

In source code we should consider adding name of the author/authors, a short description and a version of the file. We already have source control on github, we will probably not use time on this.

The poster should be finished by the 25th of may.

Time is set of for the next meeting the 19 of may 2 o'clock, and if not comes up should be the time of the next meeting.

Meeting

LAPS Group

Meeting with Ext. Supervisor 7

Participants	Håkon Jordet Vetle André Hoffmeyer Neumann Even Thonhaugen Røraas Jan Dyre Bjerknes
Recipients	Jan Dyre Bjerknes
Secretary	Vetle André Hoffmeyer Neumann
Location	Skype
Time of meeting	08.04.2020 at 11:00

24th May 2020

Discussion

The group's progress was brought up to supervisor. Henrik has been working on graph neural networks as a proposed solution to the problem. External supervisor pointed out that the resulting path has to be maneuverable, which means it has to be decoupled from the grid. Henrik mentioned that this will hopefully not be a problem, as there are graph neural network implementations which use spline algorithms, which are able to smooth out the path and its turns.

Progress on the reinforcement learning (RL) and the Deep Q-Learning approach, along with using Dijkstra's algorithm for the reward function and insights from that investigation was discussed. Supervisor discussed how RL agents are quick to find flaws with reward functions and utilize those flaws to get high rewards. Alternative ways of using Dijkstra in the reward function was brought up and discussed, which will hopefully yield better results. The formal mathematical definition of the problem is also being worked on.

On the frontend progress has been incremental, and a bug involving listing up multiple algorithms was discussed, where supervisor weighed in with possible workarounds for the problem.

Regarding the backend progress, the feature of uploading modules was announced to supervisor. This allows one to simply upload a .tar file containing the model and its files, and it will be added to the service. Adding support for map metadata was also brought up as it is needed for the pathfinding modules, particularly the point density of the map itself. No unit was given for these, but supervisor mentioned that the user uploading map should be given the opportunity to specify which unit, and in the case that there is no specified unit then it is safe to assume it's in meters. In addition to this, supervisor discussed map projections, and that this was something Håkon could spend some time on learning about and developing around if he found it interesting.

Because of next week being exam week, progress will be slowed down temporarily. A supervisor had a flexible schedule next week, the group was given opportunity to schedule the meeting on either late Wednesday, Thursday or Friday.

Meeting

LAPS Group

Meeting with internal supervisor

Participants	Henrik Thue Strocka Håkon Jordet Vetle André Hoffmeyer Neumann Even Thonhaugen Røraas Henning Gundersen
Recipients	Henning Gundersen
Secretary	Even Thonhaugen Røraas
Location	Online
Time of meeting	5th May 2020 at 14:00

5th May 2020

Discussion

0.1 What we are working on

- Vetle - Work to make the algorithms take into account turn radius. The drone needs to be able to turn early enough. Focus working on the turn short
- Henrik - Stopped working on graph network to get turn short done. Work is also done towards getting a dijkstra module that can take into account turn radius.
- Håkon - Worked on documentation and added so that multiple modules can run parallel.
- Even - Finished work on interactive map that lets the user place markers by mouse. Work started on getting map data.

0.2 Feedback on chapter 4 the project

Most criticism was of a lack of sources and in some places figures would be helpful to help explain some things. The content seems mostly fine, and the writing is good enough.

0.3 misc

Final presentation is on 11th of june.
We should talk to karoline about what should be on the USB and when it needs to be there, especially the expo poster.
We have 1 week left of product development 2 weeks of documentation, and roughly 2 weeks of presentation prep.

Meeting

LAPS Group

Meeting with ext. Supervisor 3

Participants	Henrik Thue Strocka Håkon Jordet Vetle André Hoffmeyer Neumann Even Thonhaugen Røraas Jan Dyre Bjerknes
Recipients	Jan Dyre Bjerknes
Secretary	Håkon Jordet
Location	2263
Time of meeting	27th January 2020 at 13:00

27th January 2020

Discussion

First presentation

We got feedback on our first presentation draft. We talked about dealing with handling pressure, and went through each slide in the draft.
We talked about needing to get the following points into the presentation:

- Time management
- Explain the problem:
  - Weaknesses using current methods
  - Potential strengths of a machine learning-based approach
- Presenting the members of the group
- That the problem is from KDA
- We have to define interfaces between components

Meeting with Kongsberg Digital

We discussed the logistics of our meeting with Kongsberg Digital. We need to present our problem to them in the shape of a short presentation. It does not need to be a particularly good presentation, we just have to get the problem across. They are apparently quite familiar with how to deploy machine-learning based software, and therefore probably have a number of good tips for us.

Minimum viable product

We discussed that our minimum viable product is actually rather large. We discussed shrinking it down a little, such that it does not include all requirements which are currently set to priority level 1(highest).

What constitutes a good path?

We talked about figuring out what makes one path better than others. There are many different approaches to this, and pitfalls in every direction. Which version we pick may change during the course of the project. In certain cases, it will be hard to compare this approach to the traditional approach, so measuring deviation from the ideal line using Dijkstra is not a good method.

Userstories

We discussed our userstories. They were pretty solid, except for the fact that they lack any mention of machine-learning, which is something we will be adding in some form or another.

Misc

- Wishes of a dark theme for the website were put forward
- We're starting to see a slight scope creep. We need to make sure we don't start working on too much, and have no chance of finishing it.
- Server update: There is a server we can use for training on campus. All we need to do to get the ball rolling is talk to Dag Samuelsen or Henning Gundersen.

Second Presentation

We discussed possible times for the second presentation. We decided that the best time for it would be on Thursday, 26th of March.

Meeting

LAPS Group

Initial Meeting with External Supervisor

Participants	Even Thonhaugen Røraas Henrik Thue Strocka Håkon Jordet Vetle André Hoffmeyer Neumann Jan Dyre Bjerknes
Recipients	Jan Dyre Bjerknes
Secretary	Vetle André Hoffmeyer Neumann
Location	2263
Time of meeting	27th January 2020 at 14:00

27th January 2020

Discussion

Nature of the project, why we're making it - Software that generates flightpaths usually use normal pathfinding algorithms on grids that generates paths between the nodes, with instant 90 degree turns. As a result the flightpath will have to be modified before being used by anything flying which results in the real path that is taken no longer being the optimal path. This project is therefore meant as an investigation to explore machine learning solutions as a way of avoiding this problem, in addition to making the calculation possibly faster and more constant than that of pathfinding algorithms.
Requirements - As this is more of an investigation than a product development at its core, there will be less customer requirements than that of a normal project, and the group will therefore have more say in the derived requirements and design choices.
Server solution - Server needs will be dealt with and considered in the future once more is known about how demanding the software will be. A discussed possibility is using a heavy server for training of the machine learning, and then later deploying a lightweight pre-trained software on a less powerful server.
Code repository - As this project is not confidential, the group can freely choose code repository service themselves.
Future meetings with External Supervisor - Meetings at about an hour long will be held once a week, with a list of questions or topics that will be brought up in the meeting, written and sent to external supervisor at least 24 hours in advance.
First presentation date - First presentation should preferably be the 30th or 31st of January.

Meeting Request

Participants	Vetle André Hoffmeyer Neumann Henrik Thue Strocka Even Thonhaugen Røraas Håkon Jordet Henning Gundersen
Location	Group room 2263
Date and Time	28th April 2020 14:00
Expected duration	1 Hour
Today's date	24th May 2020

Subjects to discuss

- Oppdatering på fremdrift.
- Tilbakemelding på dokumentasjon.

Meeting Request

Participants	Vetle André Hoffmeyer Neumann Henrik Thue Strocka Even Thonhaugen Røraas Håkon Jordet Jan Dyre Bjerknes
Location	Group room 2263
Date and Time	17st January 2020 13:00
Expected duration	1 Hour
Today's date	27th January 2020

Subjects to discuss

- We present our progress
- Budget, to servers and other
- Contract. We have printed out the standard contract, and dicussing signing it.
- Dicussing meeting Kongsberg Automasjon
- We present our project model

Meeting Request

Participants	Vetle André Hoffmeyer Neumann Henrik Thue Strocka Even Thonhaugen Røraas Håkon Jordet Henning Gundersen
Location	Skype
Date and Time	17th March 2020 13:15
Expected duration	1 Hour
Today's date	24th May 2020

Subjects to discuss

- What to do going forward.
- How to handle second presentation.

Meeting Request

Participants	Vetle André Hoffmeyer Neumann Even Thonhaugen Røraas Håkon Jordet Jan Dyre Bjerknes
Location	Group room 2263
Date and Time	3rd April 2020 11:00
Expected duration	One Hour
Today's date	24th May 2020

Subjects to discuss

- Documentation
- Progress update

Meeting Request

Participants	Vetle André Hoffmeyer Neumann Henrik Thue Strocka Even Thonhaugen Røraas Håkon Jordet Jan Dyre Bjerknes
Location	Group room 2263
Date and Time	24st January 2020 13:00
Expected duration	1 Hour
Today's date	27th January 2020

Subjects to discuss

- Our Presentation
- Meeting with Kongsberg Digital
- Our Progress
- Minimum Viable Product
- Framework Updates
- KDA Logo

Meeting Request

Participants	Vetle André Hoffmeyer Neumann Henrik Thue Strocka Even Thonhaugen Røraas Håkon Jordet Henning Gundersen
Location	Group room 2263
Date and Time	21st April 2020 14:00
Expected duration	1 Hour
Today's date	24th May 2020

Subjects to discuss

- Ukentlig dokumentasjon fram til innlevering.
- Hva vi har gjort fram til nå.
- Planen fram til innlevering.

Meeting Request

Participants	Vetle André Hoffmeyer Neumann Even Thonhaugen Røraas Håkon Jordet Jan Dyre Bjerknes
Location	Group room 2263
Date and Time	31st January 2020 12:00
Expected duration	One Hour
Today's date	30th January 2020

Subjects to discuss

- Discuss documentation
- The question sent on email

Meeting Request

Participants	Vetle André Hoffmeyer Neumann Henrik Thue Strocka Even Thonhaugen Røraas Håkon Jordet Henning Gundersen
Location	Group room 2263
Date and Time	3th March 2020 11:15
Expected duration	1 Hour
Today's date	24th May 2020

Subjects to discuss

- Gå over avtalt dokumentasjon.

Meeting Request

Participants	Vetle André Hoffmeyer Neumann Even Thonhaugen Røraas Håkon Jordet Jan Dyre Bjerknes
Location	Group room 2263
Date and Time	27th February 2020 13:00
Expected duration	One Hour
Today's date	24th May 2020

Subjects to discuss

- Server
- Our progress since last time

Meeting Request

Participants	Vetle André Hoffmeyer Neumann Even Thonhaugen Røraas Håkon Jordet Jan Dyre Bjerknes
Location	Group room 2263
Date and Time	24th April 2020 11:00
Expected duration	One Hour
Today's date	24th May 2020

Subjects to discuss

- Progress update
- Documentation to look over to next meeting.

Meeting Request

Participants	Vetle André Hoffmeyer Neumann Even Thonhaugen Røraas Håkon Jordet Jan Dyre Bjerknes
Location	Group room 2263
Date and Time	31st January 2020 12:00
Expected duration	One Hour
Today's date	24th May 2020

Subjects to discuss

- Discuss documentation
- The question sent on email

Meeting Request

Participants	Vetle André Hoffmeyer Neumann Even Thonhaugen Røraas Håkon Jordet Jan Dyre Bjerknes
Location	Group room 2263
Date and Time	19th February 2020 13:00
Expected duration	One Hour
Today's date	24th May 2020

Subjects to discuss

- Server
- Our progress since last time

Meeting Request

Participants	Vetle André Hoffmeyer Neumann Even Thonhaugen Røraas Håkon Jordet Jan Dyre Bjerknes
Location	Group room 2263
Date and Time	8th April 2020 11:00
Expected duration	One Hour
Today's date	24th May 2020

Subjects to discuss

- Progress update

Meeting Request

Participants	Vetle André Hoffmeyer Neumann Even Thonhaugen Røraas Håkon Jordet Jan Dyre Bjerknes
Location	Group room 2263
Date and Time	5th February 2020 12:00
Expected duration	One Hour
Today's date	24th May 2020

Subjects to discuss

- Thoughts on report if you had time to read it.
-

Meeting Request

Participants	Vetle André Hoffmeyer Neumann Henrik Thue Strocka Even Thonhaugen Røraas Håkon Jordet Henning Gundersen
Location	Group room 2263
Date and Time	20th January 2020 11:15
Expected duration	45 min
Today's date	27th January 2020

Subjects to discuss

- Process model, comments on Kanban.
- Administrative documentation, how much.
- Important things for first presentation.
- How much documentation for first presentation.

Meeting Request

Participants	Vetle André Hoffmeyer Neumann Henrik Thue Strocka Even Thonhaugen Røraas Håkon Jordet Henning Gundersen
Location	Group room 2263
Date and Time	10th February 2020 11:15
Expected duration	45 min
Today's date	24th May 2020

Subjects to discuss

- Thoughts on first presentation
- Thoughts on the report

Meeting Request

Participants	Vetle André Hoffmeyer Neumann Henrik Thue Strocka Even Thonhaugen Røraas Håkon Jordet Henning Gundersen
Location	Group room 2263
Date and Time	27st January 2020 13:15
Expected duration	1 Hour
Today's date	27th January 2020

Subjects to discuss

- Our Presentation
- Documentation

Meeting Request

Participants	Vetle André Hoffmeyer Neumann Henrik Thue Strocka Even Thonhaugen Røraas Håkon Jordet Henning Gundersen
Location	Group room 2263
Date and Time	17th February 2020 11:15
Expected duration	1 Hour
Today's date	24th May 2020

Subjects to discuss

- Discuss how we would like to review documentation in the future.
- Project update
-

Technical

LAPS Group

Starting point for the product

27th January 2020

Contents

1 Introduction

To start writing the service, a few things should be in place first. We should have a few user-stories which cover the most basic functionality of the service. We have a few of these already, and have derived features from them. From those, we have derived some requirements. We use this as a starting point.

Because we have these in place, as well as an idea of what our general architecture should be, we will be able to start development after the first presentation.

Our project is very open-ended. We can use the technologies we think are best to solve the problem at hand. We're writing the service from scratch. This means we have zero legacy code to deal with, which gives us a lot of freedom in our technology- and architecture choices.

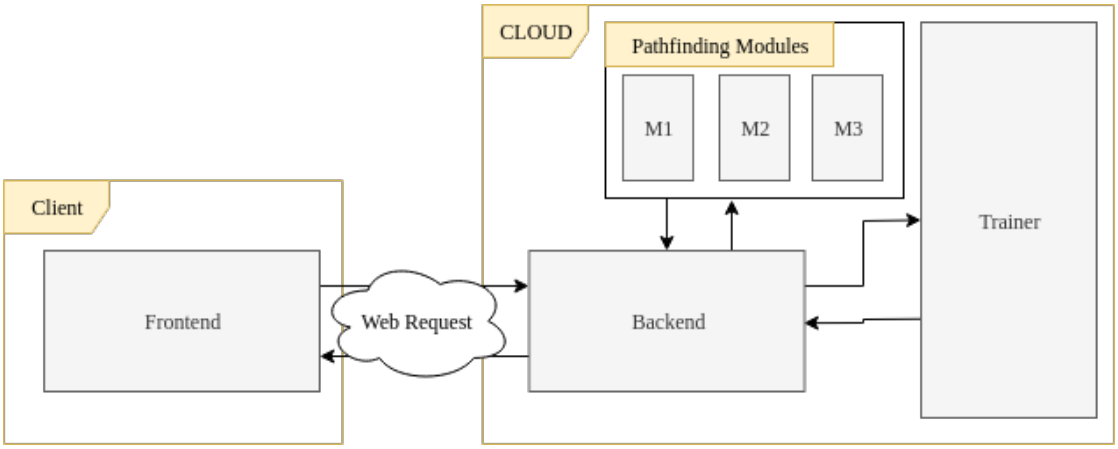


Figure 1: High-level architecture

The general architecture can be seen in Figure 1. It emphasises a separation of front-end and back-end. The pathfinding modules are hot-swappable. That is, we can use any of the available modules to generate a path. These are all going to be handled as a single service. This service will not be available to the user, but will instead be communicated with by the back-end.

The back-end's responsibilities are:

- Powering the Front-end
- Collect and pass on path data from pathfinding service
- (Possibly) Coordinating the training of different pathfinding modules

2 Architecture: Server-side («the cloud»)

As stated in Section 1, the server-side part of the service has numerous responsibilities and consists of multiple components. In order to achieve this, there needs to be some sort of interface between the two. Defining these interfaces is part of the systems engineering process, and is vital to our success.

It is not clear right now what this interface should be, because we do not have requirements for it yet. It will likely fluctuate as our needs change. While stability is important in this case, we know from experience that forcing it to be too stable too quickly is a bad decision. Thus, we will try our best to keep it stable, while at the same time adapting it to our needs.

Speaking of interfaces, the interface between the front-end and the back-end will take the form of a RESTful API. This means using HTTP methods in the correct manner, and changing the request path to indicate different functionality. Compared to the internal server API, this is going to be more volatile. The front- and back-end are deployed as one, and we can afford to change it much more as we go along. In addition, a number of front-end features depend directly on functionality in the backend and may require more data in the future.

2.1 Programming language

A programming language is a tool: We need to pick the right tool for the job. For the pathfinding subsystem, the choice is between C# on .NET core and Python. Both have solid machine-learning frameworks which make it easy to define and use models. Both of them also run in Linux environments, which is important for the deployment of our service. Both of them have server capabilities which allows us to communicate with the back-end easily.

There are many libraries and programming languages suited for back-end development. We want to write our back-end in Rust. Rust is a systems programming language with a focus on safety. It has an ownership and borrowing model which will ensure that your code is memory-safe, without the use of a garbage collector[1, Chapter 4].

Rust's safety guarantees also include thread safety[1, Chapter 16]. All of this allows us to write and deploy a backend which is free of data races, buffer overflows other soundness issues.

2.2 HTTP Frameworks

We need some kind of HTTP library or framework in order to write the backend. There are many available for Rust, and we have narrowed it down to two choices. Before we start working, we have to decide which one to use.

2.2.1 Rocket

Rocket[1] is a very solid HTTP framework. It has many features which make writing web applications using it a breeze, and it's powerful request guards provides a solid abstraction for authentication, database connections and more.

The downsides of Rocket are that it depends on unstable Rust features, and that the latest stable version is about a year old with outdated dependencies. Work is on-going to get a new version out supporting asynchronous I/O. An alternative could be to use this development version of Rocket, but that means relying on an API which is subject to change.

2.2.2 Warp

Like Rocket, Warp[2] is a Rust HTTP server framework. It is a lot simpler than Rocket in design and features, but is a good starting point to build a solid architecture. Advantages of warp:

1. Support for asynchronous I/O.
2. Works on stable Rust 1.39 or newer.
3. It's filter system makes code re-use easy in each endpoint handler.
4. Supports Websockets and multipart forms natively.

3 Machine learning approaches

Because of the nature of the project being focused on research and the external sensor not limiting our choice we did not have many requirements for frameworks. Because of this we stood free to choose what frameworks from the many there was to choose from. So a list was compiled from all compatible frameworks on AWS(Amazon web cloud) because they are the biggest distributor of cloud computing. From there the group assigned pros and cons for each framework to decide which one was an option. Torch was chosen because of its ease of use and its leaning towards researching instead of a final product and was suitable for rapid prototyping.

As discussed in our process model the group had to be really agile when developing because machine learning is notoriously hard, and coupled with the research aspect and very little previous research done on the matter there is no sure way to accomplish the task. Therefore there was a brainstorming session where the group came up with three plans that each had a different way to approach the problem via machine learning.

[1]Link: `https://rocket.rs`
[2]Link: `https://github.com/seanmonstar/warp`

3.1 Plan A

Plan A is the plan that the group felt had the biggest chance of succeeding and was conceived with the help of the external supervisor. The plan focus on using a unsupervised approach called Deep Q Learning. This works by using a neural network to generate a set of weights for all actions the agent can preform given the current state he is in, in this case the state will probably be the agent position on the grid. The agent then chooses the action with the smallest weight.

This approach has two main advantages, the first is that this approach does not require any pre-labelled data hence the unsupervised learning. The second advantage is that if convolution is use in the neural network it should have the ability to extract features from the height map, in a similar way to how humans can look at a map and quickly draw a path around a big mountain.

3.2 Plan B

Plan B is in many was similar to Plan A because it aims to exploit the same feature from Convolutional Neural Networks. But instead of using it to generate weights for a Q Learning algorithm it will instead directly generate the path. This in turn negates one of the advantages of Plan A because now the model needs pre-labelled data to train from. Which means the group has to find a way to generate sufficient data and label it.

3.3 Plan C

Plan C is to use what is called a generational adversarial neural network, this is a complex name for a simple concept. It revolves around having two neural networks, a generator which tries to generate fake paths to fool the discriminator which tries to distinguish between the fake paths and the real paths. After the training the end result is that the group hopefully has a generator network that can generate paths that accurate enough to be indistinguishable from real paths.

4 Architecture: Client side

On the client side, we want to have a mostly single-page web application. In order to do this, we will be utilising a RESTful API. The site will be almost exclusively JavaScript-driven. This implies that we do not render HTML on the server. This is a sensible choice for building singe-page applications, and makes the distinction between front-end and back-end greater, making it easier to work together on the site.

To power the UI, we will be using the JavaScript framework Vue.js. We mainly chose Vue because none of us have much frontend development experience on the web, and Vue is simpler

than many of the other frameworks. Vue has a large following developing components which we will be able to re-use in our project. simple yet powerful framework which allows us to split our code up into components, which helps keep the code clean.

Without having tried writing applications in the other frameworks out there, there is a risk that our choice of Vue is a mistake and will cost us in the long-run.

5 Where to go from here

Now that we have a general architecture design in mind, we can start working on the project. Feature F.1 is essentially our minimum viable product. So to start with we set up the skeleton to build the rest of our architecture on. It is important that we do not stick to the plan too much. Excessive planning means the team is no longer agile, and does not fit in with our project model.

Because of our agile model, we will be doing continuous, incremental improvements to the website. This allows the customer to comment on our progress and give us a better explanation of what they want.

References

[1] Steve Klabnik and Carol Nichols. *The Rust Programming Language*. No Starch Press, 2019. ISBN: 9781718500440. URL: https://nostarch.com/Rust2018.

Technical

LAPS Group

Next steps after the 2nd presentation

24th May 2020

1 Introduction

Now that we have had some weeks to work on the final product, we have achieved most of the milestones to create a minimum viable product. In this document, we explain where we are in relation to our milestones and what our next steps are following the 2nd presentation.

2 Milestones

As a reminder, Figure 1 displays what our milestones are. We have achieved the bottom-most milestone: defining the pathfinding API. On the pathfinding side, we have implemented a simple pathfinding module which uses an implementation of Dijkstra's algorithm, so we have achieved the first two milestones there.

On the website side, progress is somewhat split. The backend is further along than the frontend. This is because there is less that needs doing on the implementation side. The backend supports submission and retrieval of pathfinding jobs, while allowing the selection of pathfinding algorithms. It allows for listing of algorithms such that the user can know which ones are available. Finally, it allows for retrieval of map data. Therefore, we can see that we can technically start working on features required for the admin panel in the backend.

legg til hvorfor backenden er lengre foran

The frontend have as of presentation 2 a few of the core features implemented. Primarily the ability to take user input in the form of a start and stop and sending it to the backend, where it will be used further. The user can select maps as long as they are currently uploaded into the backends database. The frontend supports display and selecting pathfinding modules. All expected outcomes of a pathfinding should currently be useable in the frontend, and therefore adding new pathfinding modules should be easy. The first two milestones for the frontend/backend have been accomplished and work towards the next one have begun.

3 Next steps

In the backend, there are a lot of small tasks which need doing. Up until this point, the focus has been on creating a minimum viable product, and working on documenting everything as well as we can. There are a lot of smaller things which really should be done. Some of which are relevant to features later on, while some improve existing functionality in subtle ways.

After doing many of the improvements we want to do in the backend, we can start working on features we need for the admin panel. The main thing is having some way to manage pathfinding modules, being able to start and stop them from code as needed, while keeping them running if the backend exits. We also need some way to automatically import map data. When these are in place, we can design an initial version of the API to manage the modules, and start implementing it. Afterwards, we can working towards the other milestones down the line.

As the frontend have reached a point where all expected pathfinding modules are supported, work can be focused on moving features currently done in the backend to the frontend and the expansion of new features. Some things that can should added after backend support is created, is user uploaded maps, a login feature, and an admin panel. Purely on the frontend, a function to display end and stop points before they are sent to the backend should be added. Also making the start/end points placeable with a mouse. Currently the map is displayed at the resolution it is received, this is because the coordinates system is based on pixels. Large map will therefore be impractical to use, and almost unusable on small screens like phones or tablets. A map should rather be scaled to a default size, and match the coordinates accordingly. Because most of the work have gone into development of the basic functionality, the website isn't as aesthetically pleasing it should be.

For the pathfinding modules, we want to have some kind of build system to make deployment a lot easier. Currently we have to manually convert the Jupyter notebooks into plain Python files and manually manage dependencies. It would be very convenient to have some kind of build system which would do this automatically for us.

We will continue to investigate different machine learning solutions and network layouts. Our focus will probably shift away from convolution and more towards reinforcement learning and graph neural networks, as they show more promise going forward.

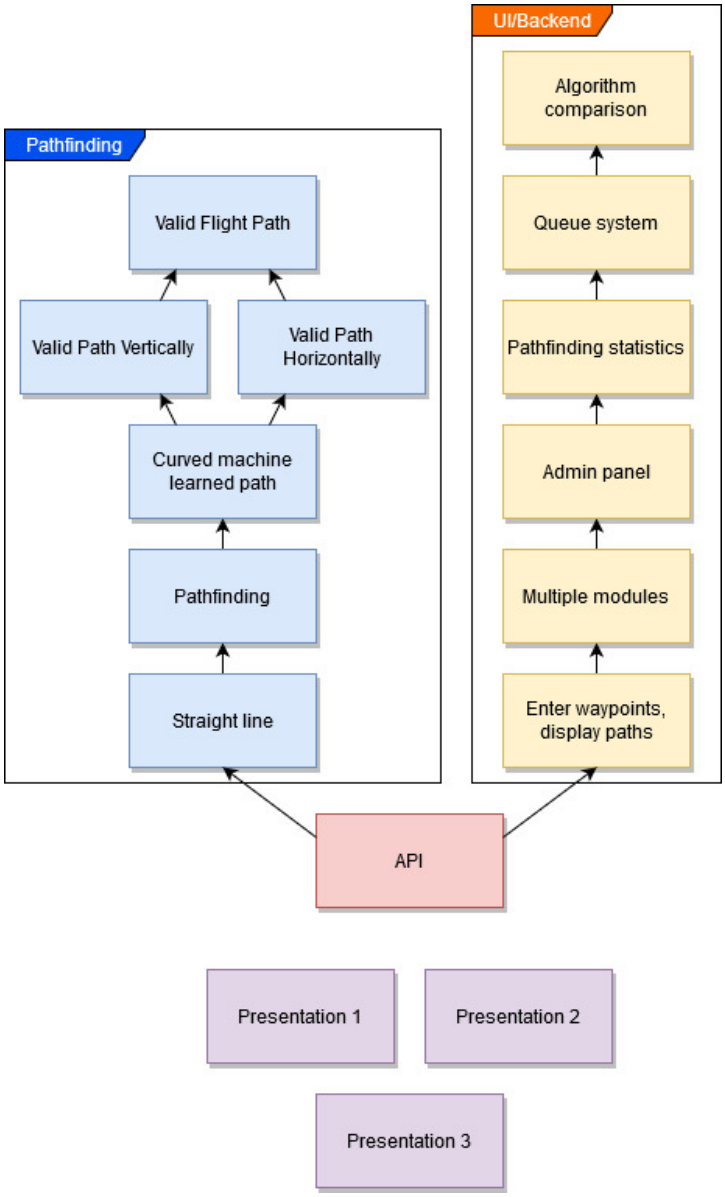


Figure 1: Milestones Diagram

Weekly Report

LAPS Group

Summary of project week 1

6th January⟶ 13th January

13 January 2020

Contents

1. Completed Tasks

1.1. Documents

This week we built up some document templates in order to make our documents have a distinct look and feel to them. We do this in order to make them easily recognizable, and for consistency. This consists of a LaTeX class file with a number of definitions and a custom title page.

This class file is by no means final and will likely keep evolving as the project continues.

We also created template files in order to ensure that every document of the same type will have roughly the same structure. This helps us ensure that we write about everything we agreed on.

1.2. Proccess

This week we selected our process model for the project. We've decided to use Kanban, because we think that will work the best with how we work. We were looking for an agile model, and we were torn on using SCRUM or Kanban. We figured that Kanban would work best for our group.

1.3. Meeting with external advisor

We also had our first meeting with our external advisor, Jan Dyre Bjerknes. He cleared up a lot of questions we had about the project, namely about our requirements. We learned that we can do things the way we think is best. We also don't have a large amounts of requirements, and are free to choose the requirements that we think is the best for the project.

We talked about what the best time for the first presentation would be. It will be the 30th or 31st of January.

1.4. Meeting with internal supervisor

We had our first meeting with our internal supervisor this week. We discussed a few things, and set up a meeting schedule for every Monday. We found out we would have the first presentation on the 31st of January, which we will have to book a room for.

1.5. Structure of thesis

While the final report is months away, we figured it was a good idea to have the general structure of our thesis set up. This allows us to write it over a long time-period. To do this, we looked at a large number of previous year's theses, and looked at what we liked and disliked in each of them. We came up with a structure which we think will be a good starting point. It is almost certainly going to change.

1.6. Name

We debated for a while about what the name of the project should be. Our external advisor gave us free reign of the name, without any preferences. We landed on calling the project **L**ow-**A**ltitude **P**athfinding **S**ervice, or *LAPS* for short. The project group is known as *LAPS Group*.

1.7. Gantt Diagram

We created a setup for our Gantt diagram. This is contained in an Excel file, where we can easily type in our data and it will present a Gantt diagram of this data.

1.8. Requirements presentation

We figured out how to present requirements in technical documents the way we want them to look. It is important to us that this is done consistently across the documents, and that it is fairly quick to do. To do this, we need to write LaTeX code to do this for us. This will most likely be done next week.

1.9. Logo

Because we were given free reign over our logo, we have asked a friend who professional designer to turn our logo idea into a real logo. If he doesn't want to do it, we will probably commission one from somewhere.

1.10. Hour tracking

We set up an hour tracking spreadsheet to track how much we've worked. Appendix A shows a table of our hours worked this week.

2. Tasks for next week

2.1. Requirements

Next week we have to get the first requirements written down properly. We want to have layered requirements, where the essential requirements are the top-level ones. As development progresses, we break these requirements down into many smaller requirements, to the point where every feature has it's own set of requirements. It requires some indication of which requirements are given by the customer, and which are design decisions on our part.

2.2. Improve document templates

We will keep on improving our document templates. In particular, we want to add our logo somewhere, and have some kind of decoration on every page of text. There are also templates for a few types of documents which are missing and have to be added.

2.3. Document process model

Having decided on what we want our process model to be, we want to formalise it. This document will contain an in-depth explanation of why we think the model we have chosen is right for us and our project. It will describe how we intend to document who did what and our progression throughout the weeks.

2.4. Meeting with Jan

We will be meeting with our external supervisor again this week.

A. This week's working hours

Date	Even	Henrik	Håkon	Vetle
Mon. Jan 6	1.75	1.75	3.75	1.75
Tue. Jan 7	3.5	4	4	4
Wed. Jan 8	4	4.5	4.5	4.5
Thu. Jan 9	1.5	5	5	4.5
Fri. Jan 10	3.5	5	5	5
Sat. Jan 11	0	0	0	0
Sun. Jan 12	0	0	0	0
Total	14.25	20.25	22.25	19.75

: Person was sick : Weekend

Weekly Report

LAPS Group

Summary of project week 8

22th February⟶ 28st February

28 February 2020

Contents

LAPS
LOW-ALTITUDE PATHFINDING SERVICE

1. General

This week we continued to work towards a minimum viable product. Our current goal is to be able to present it on our second presentation on the 26th of march. This week focus has also begun to shift from prototyping to documenting as we need most documentation done by next week because the exam period will occupy the next two weeks after that.

1.1. Håkon Jordet

This week, I finished up a couple Trello cards which had failed QA, mostly having to change documentation. I also did a little behind-the-scenes re-vamp of the backend logging system, set up our brand new server, and continuous delivery of the backend + frontend.

1.1.1. Backend

I had originally used *tracing* instead of normal logging. Tracing is a type of logging built around spans. When a span is in scope, the span becomes part of the logged message. A span can have information associated with it, for instance an IP. This makes log output easier to read when using asynchronous IO. When using asynchronous IO, the running thread will jump to working on another task when it has to wait for IO. This massively increases the number of requests which can be processed at once, and improves CPU utilization. This makes it hard to tell which event is related to what request, as an example.

However, tracing does not play well with Rocket. Rocket has it's own logging facilities, using the Rust crate *log*[1]. I was initially using a compatibility layer which allows crates using log to work with a tracing subscriber. This caused issues while unit testing, where it felt much more natural to depend on Rocket for logging(and therefore printing of any useful information if the test fails). In general, it was not pretty and the choice was made to change to the log crate instead.

1.1.2. Architectural

We finally got a server and domain up and running. Our domain is `https://laps.website`. I spent a lot of time getting the server up and running. I configured the HTTP server, iptables chains(firewall), encryption(HTTPS) and a deployment scheme in general.

We have two deployment environments: Staging and production. Production is the latest version of the site which we will show off to the world. It will be what the machine learning developers use when testing out the newest versions of their algorithms. They are running on the same server, but are isolated from each other. We do this with normal administration techniques, i.e running staging and production as different users with different home directories.

[1]link: `https://crates.io/crates/log`

The other one, staging, will be mostly used for frontend development. The idea is that a frontend developer can have a testing environment for their latest and greatest work. This environment will be set up and ready to go automatically, removing the need for any setup on the frontend developer's local machine. It also allows them to use VueJS' hot reloading features, which requires one to run a specialized web server.

URLs will have to be changed to work in this mode. Every relative path must be changed into an absolute one which points to the staging instance. A request that looks like `GET /maps/1` has to be changed to `GET https://staging.laps.website`.

To facilitate running both staging and production versions of the site at once, I will find a way for the two backends to not interfere with each other.

1.1.3. Continuous Delivery(CD)

I have successfully integrated CD into our workflow. It is a modern pattern where we continuously release new versions of our software. This is done by automating the deployment process. If we are on a git branch we deploy, in our case, staging and production, it will first automatically run all tests and ensure that they are working. If everything works, we can safely move on to deployment. Deployment executes a script which can be used to deploy the application. Many CD service providers have integration with cloud services like AWS and Azure and can deploy directly to them.

In our case, we are using Travis CI, which is completely free to use as long as one's project is open source. Any commit to any branch will trigger the build, but only the production and staging branches will trigger deployment. When making a pull request on Github, it will automatically build and test that as well, which serves as a way to verify the new code automatically.

When a deployment is triggered, we firstly decrypt an SSH key which can be used to log in to the deployment and staging users on our server. It is encrypted in order to only allow Travis' servers from reading the key and logging in to our system. Then, the backend is compiled in release mode, suitable for production. It bundles up the frontend code. Finally, it stops the running staging service and copies the built files to the server, and restarts the staging service. When that is done, it has successfully built, deployed and restarted the application for us.

1.2. Even Thonhaugen Røraas

A path is sent to the front end in form of array with points. Each points have an x and y value, these values are the same as the pixel values in the received. Therefore marking the points on the map is quite easy. A new vue component request the coordinate values, and places a red dot according to the points. To send data between the components, such as sending the coordinates to the component that places them, a store to store variables was created. In the store each component can request or update values such as all other components can use

them. As vue focus on splitting up function in the website into components, having good communication between them is important.

1.3. Henrik Thue Strocka

Much like last week my focus started with figuring out a solution to using a CNN to predict the midpoint of a path. This week my goal was to convert it from a continous problem to a labelling problem. This week i managed to create a labeling CNN, i took inspiration from how AlexNet was structured. Sadly the results was very dissepointing, not much better than the results gotten last week.

However this got me thinking about why my solutions where not working. And i think the essence of it is that i have an additional data source for the network to consider which all of the networks i have drawn inspiration from has not had. That is the start and stop point for the path, for the other networks, object detection and face keypoint detection all the data has been in the images.

So my hypothesis is that because the convolutional layer only has access to half the data in the form of a grayscale heightmap, i belive it has a hard time converging on some proper values for what features to extract from the image. In Fig 1 you can see a image representation of the first 10 convolution filters from each layer, as you can see they are pretty much random suggesting that it is indeed struggeling to extract any features.

The way i plan to solve this is to use what is known as an auto encoder for first train the convolution part of the network to be able to extract features from the image. Then after that layer has been trained i can then train the fully connected layer at the end using those feature maps and the start and stop point for the path. After reading this paper on positional properties in convoltuonal networks, https://openreview.net/forum?id=rJeB36NKvB, my hope is that the posistions of each feature will be kept and the network can output a good geuss for the midpoint.

1.4. Vetle André Hoffmeyer Neumann

In retrospect I forgot to fill this in but it is explained in the next week summary.

A. This week's working hours

Date	Even	Henrik	Håkon	Vetle
Sat. Feb 22	0	0	0	0
Sun. Feb 23	0	0	0	0
Mon. Feb 24	6.75	6.5	7	5
Tue. Feb 25	0	0	0	0
Wed. Feb 26	6.75	7	6.5	7
Thu. Feb 27	6.5	8	7.5	9
Fri. Feb 28	7.15	8.25	7	7
Total	27.15	29.75	28	28
Overall Total	224.9	227.4	228.75	243.75

: Had lectures : Not part of our normal working hours : Person was sick

Weekly Report

LAPS Group

Summary of project week 6

8th February⟶ 14th February

14 February 2020

Contents

LAPS
LOW-ALTITUDE PATHFINDING SERVICE

1. Completed Tasks

Technical work has officially begun. We had a functional analysis meeting in addition to a replenishment meeting on Wednesday, without the customer as he is out for the weekend, and work on the service has begun. Focus has been on the requirements with priority 1, as they are the intended minimum viable product, which we are hoping to have completed within the customer gets back.

Within the pathfinding part of the task, it was decided on the functional analysis meeting that the chosen approaches we want to investigate first, is Deep Q-Learning in addition to Convolutional Neural Networks in a simplified scenario.

As a group, we worked on defining the API between pathfinding modules and the backend. Our main focus was on ensuring that it is as simple as possible while still being useful. We decided to base everything around Redis, and currently it only has 5 types of messages being passed from end to end.

We discussed with Dag about using a server located at Krona with 4 GPUs for training the machine learning models, but were required to come up with an estimation of teraFLOPS before using it.

1.1. Håkon Jordet

After we had defined our pathfinding module API, I set out to document it as well as I could. I wrote about it in detail in it's own document, at least for now. Afterwards, I wrote an initial python library in order to make it quick and easy for the machine-learning guys to get started. I tested this with an initial implementation of the back-end. The library I wrote has several issues, but it was intended for them to mold to suit their needs.

Speaking of back-end, I did a lot of work to go towards the minimum viable product. Namely, I started to work on the backend part of the project. First of all, I wrote about which framework and language we use for it, Rust with Warp. Then I got to work on RQ.1.1, namely by adding endpoints for the user to submit a path. As of right now they are not integrated to the actual website at all, but I have been testing using the curl utility from the command line.

I also looked into RQ.6.1: Height data must be available across the system. I did a lot of test exporting of data and trying to display this data using a python script. We were considering making our own format which gets rid of all geographical data, but we discovered that we could export the data in GeoTIFF format, which is just an image format with some extra geographical data. This is a better option than creating our own format and converting a point cloud into said format.

1.2. Even Thonhaugen Røraas

As i never used Vue and very little javascript, most of the time where spent on learning these. Some time where also set of to settings up a developing environment, such as installing Vue dependencies. Vue also have premade plugins created by the community for common features. Taking advantage of this could save development time.

Some code was also written, but limited. A simple application that takes a coordinate from the user converts into JSON for transfer to the back end for further use. The intention is not as permanent solution, but rather to see that a connection can be established between the frontend and backend. Hopefully if we can establish a channel and format early such as development can progress in both FE and BE. The code is not yet test against the backend, and will be done next week.

1.3. Henrik Thue Strocka

The task i was assigned to first is to investigate the option of using convolutional neural networks directly to generate a path. My hope is that convolution part of the network will be able to extract features that the normal network can then use to find the midpoint of the path.

Because neural network does not deal well with variable inputs or outputs all training data had to be of the same size. Therefore this week i also wrote a data generator that generated a random heightmap using perlin noise and using dijkstra found the best path with the least height difference. This program does not tak into account how well the drone can turn but its a good starting point to see if the network can produce some decent results.

Because of the inability to output variable outputs i decided to have the network generate the midpoint of the path and then run it recursively however many times i saw fit generate a proper path.

I was able to create the model and start training it and the initial results where very promising. From just around 1k training images i was able to train the network on my shitty laptop to produce pretty accurate results. The network output points had an average distance from the correct point by just 5 pixels in a 256 by 256 image.

1.4. Vetle André Hoffmeyer Neumann

The standard way of doing machine learning via Python is to use a Jupyter Notebook via a Docker setup. Dockers are similar to Virtual Machines, in that they are containers with processes for running things, which makes it so as long as you've set up the docker correctly the application running inside it will be easily portable and work with little to no extra effort on another system.

The planned setup we will be using which I've been working on is having a Jupyter Notebook docker, with all the tools for developing the machine learning models. Docker containers are

only supposed to host one process at a time, and as we are currently working without a dedicated server for the backend (hosting the service on our local computers) I've temporarily set up a Redis docker container, so that we can communicate with the server running locally from the Jupyter Notebooks.

As we are as of now not using the server at Krona, and instead on our own computers, having a docker setup for training the machine learning models will make setting it up on the server extremely easy.

I've also started working on the simplified Deep Q-Learning approach, starting on implementing the Q-Learning environment, though that's still in the early stages.

2. Not Completed Tasks

3. Tasks for next week

Work on minimum viable product continues, and it will hopefully be achieved before the meeting with the customer on Wednesday.

A. This week's working hours

Date	Even	Henrik	Håkon	Vetle
Sat. Feb 8	0	0	0	0
Sun. Feb 9	0	0	0	4
Mon. Feb 10	7.25	7	7	7
Tue. Feb 11	2	0	0	0
Wed. Feb 12	7	9	7	7
Thu. Feb 13	6	7	6	8
Fri. Feb 14	7	7	7	7
Total	29.25	30	27	29
Overall Total	166.75	164.15	172.45	180

: Had lectures : Not part of our normal working hours

Weekly Report

LAPS Group

Summary of project week 10

21th March⟶ 27th March

23rd March 2020

Contents

LAPS
LOW-ALTITUDE PATHFINDING SERVICE

1. General

This week we did our second presentation and as a result we will not include our individual contributions in this week report, anything of notice will be included in next weeks report. The week started with the group finalizing our documentation making sure the relevant people received it on time. The second part of our week was spent wirting the presentation, recoding it and editing it all together, we did underestimate the time it would take to do this and for our next presentation we will definitely allocate more time for presentation preparation.

A. This week's working hours

Date	Even	Henrik	Håkon	Vetle
Sat. 21th March	1	0	0	5
Sun. 22th March	6	5	3	10
Mon. 23th March	8	7	7	6
Tue. 24th March	9	7	7	7
Wed. 25th March	10	7	7	7
Thu. 26th March	6.5	10	7	7
Fri. 27th March	0	0	0	0
Total	40.5	35	31	42
Overall Total	319.15	320.4	325.45	351.75

: Not part of our normal working hours : Person was sick

Weekly Report

LAPS Group

Summary of project week 12

4th April⟶ 10th April

24th May 2020

Contents

LAPS
LOW-ALTITUDE PATHFINDING SERVICE

1. General

This week was cut somewhat short by our impending exams, where most of the group decided to spend some of our time working on assignments.

2. Håkon Jordet

I decided to spend almost all of my time this week working on the project, because there were things I really wanted to get done. I was quite successful. In the last report I mentioned working on uploading pathfinding modules and running them. I fully implemented everything we need to make that all work. You can now start and stop modules individually. You can get a list of modules with their name, version and running status.

Next, I worked on administrator authentication. I did a simple implementation earlier, just to make sure that I would ensure that the endpoints which require authentication actually do. I had chosen a bad hashing algorithm for the password storage, using a general-purpose hashing function instead of a password hashing one. After some research I decided to replace it with Argon2.

Until now, there was no way to actually register admins. In the unit tests, I would simply hash the passwords and write them to the database manually, not so anymore. I added two admin registration endpoints. One of them requires no authentication. This one is intended to be used when initialising the backend for the first time, creating a super admin from scratch. It will only allow one to register if there are no admins registered from before. The second registration endpoint requires one to be an authenticated super admin. It allows the admin to easily register a new administrator.

3. Even Thonhaugen Røraas

4. Henrik Thue Strocka

The first three days of this week was spent working on the Graph Neural Network and the progress has gone suprisiningly smoothly. After struggeling with the library installation and cuda errors i was able to get prebuilt graph based neural network called SplineConv working. With this working i feel it is time i go into detail as to why i wanted to use graph neural networks a bit more, and this implementation more especially.

So as stated in our second presentation convolution is not good a remembering and extracting positional information from images. So because pathfinding is an inherently graph based problem and by representing the problem on graph from you remove the positional element,

i feel this network suits this problem uniquely well. The downside is graph based convolutional networks are significantly more complex and there is so much to learn. But they are not too different to be incorperated into vetls RL solution should i come upon som promising solutions.

The next stage for me right now is to generate a custom dataset and use that in my own custom implementation to try and find a solution. Hopefully this will not take too long.

5. Vetle André Hoffmeyer Neumann

A. This week's working hours

Date	Even	Henrik	Håkon	Vetle
Sat. 4th April	0	0	0	0
Sun. 5th April	0	0	0	0
Mon. 6th April	0	0	9	0
Tue. 7th April	0	0	10	0
Wed. 8th April	0	0	7	0
Thu. 9th April	0	0	8	0
Fri. 10th April	0	0	5	0
Total	0	0	39	0
Overall Total	0	0	406.45	0

: Not part of our normal working hours : Person was sick

Weekly Report

LAPS Group

Summary of project week 9

14th March⟶ 20th March

23rd March 2020

Contents

LAPS
LOW-ALTITUDE PATHFINDING SERVICE

1. General

Thanks to the rapid spread of COVID-19 in Norway, USN has been closed, and exams postponed. We had initially planned to not work this week, but because the exams were postponed, we carried on.

Because we have to self-isolate, we all worked from home. We find that our model works pretty well for this, and we substitute face-to-face contact with having voice calls. We have our usual standup meeting this way.

1.1. Håkon Jordet

1.1.1. Admin panel

I worked on the admin panel this week. I wrote a conversion library for the system which uses the GDAL library internally. It was extracted into a Rust crate such that it could be used both in the backend and in a standalone CLI tool. I then integrated this into the backend.

This took quite a long time because I had to handle multipart forms manually. Rocket does not support these, so I had to roll my own implementation to get it working. I did get it to work in the end, and now there's a POST method at `/map` which will upload any map, as well as unit tests to verify that it works.

1.1.2. Building pathfinding modules

Next, I got to work on building pathfinding modules. For the admin panel, we need some way to build the pathfinding modules and easily integrate it with the system. It was decided to use Docker for this, because each module could then be built as a Docker image, and we can use the Docker API to create containers and run them on the fly depending on which pathfinding modules should be active at a time. Because the machine-learned pathfinding modules haven't been integrated into the system yet, I only wrote a packaging script for our simple Dijkstra module.

1.1.3. Admin authorisation

After making the map upload tool, I realised that if we were to put the map uploader out onto the open internet, there would be problems. As such I decided that I needed to work on admin authorisation next. This would prevent someone from spamming the service and bringing it offline, as it has to accept quite large files. For reference, some of the GeoTiff files we have exported have been upwards of 200MB, so we have to configure the web server to accept files which are that big.

1.2. Even Thonhaugen Røraas

The website now request information against the backend server rather than a locally hosted backend. Most work have gone towards fixing problems with updating the request format, that was updated for the backend up not locally.

Because the jobs could only be tested against the server a few issues are first now coming into view. One error was that the documentation for the backend didn't match the actual backend, which caused some issues. Progress towards mvp is mostly done and if no major setback happens should be done soon.

1.3. Henrik Thue Strocka

1.3.1. Graph Neural Networks

This week was spent working on a new potential solution for our machine learning algorithms. A new type of neural network called Graph Neural Networks, as the name implies they work on graphs instead of euclidean data.

The first part of the week i spent trying to get the library torch-geometric to work. This is a library for developing and using graph neural networks in pytorch. There is possible to write graph neural networks from scratch in vanilla pytorch but it requires alot of boilerplate code and is suboptimal.

However this was easier said than done and because of some CUDA errors i was unable to get it working. And with our appointment with dag on thursday i decide to put the graph neural network on hold until we had the server up and running.

For the remainder of the week is spent some time trying different approaches and improvement to our existing convolutional approach.

1.4. Vetle André Hoffmeyer Neumann

These last weeks I've been working on the reinforcement learning agent, and the environment. The environment itself is a representation of a problem in which you are in a certain state, and you have a set of actions to choose from to take you to a new state. The state of the environment is all the required information for solving the problem that should be accessible for the agent. For example if you made an environment for blackjack, the state would be all the visible cards, and you would avoid giving it information it shouldn't have, like the value of cards that aren't visible. The agent is the part of the reinforcement learning that makes the decisions regarding which action to take. The environment should also give a number reward, to tell the agent how good the choice was, and its the agents job to figure out how to maximize the reward via reinforcement learning.

The reinforcement learning approach I've been trying to implement is double deep Q-learning, which uses neural networks to approximate how well each action is, without keeping a table of every action and its states. This approach or something similar is necessary for us, as our goal is to remove the pre-processing which means training the network on each map to find a path on would be missing the point, which in turn means the map has to be a part of the state. This exponentially increases the amount of possible states, and it would not be feasible to keep a table with all these states, both in terms of resources and the time it would take to train it.

So I've spent some time learning reinforcement learning, in addition to learning PyTorch. One of the problems with machine learning is that if you've done something wrong, it will just give you wrong output, but it is extremely hard to pin down what causes the wrong output.

In addition to these things we recently got access to the server at USN, and I've moved and set up the Jupyter Notebook development environment there instead of at my own computer.

A. This week's working hours

Date	Even	Henrik	Håkon	Vetle
Sat. 14th March	0	0	0	0
Sun. 15th March	0	0	0	0
Mon. 16th March	2.5	5	7.5	6
Tue. 17th March	4	6	6	7
Wed. 18th March	6	5	7	7
Thu. 19th March	7	7	8	10
Fri. 20th March	6	7	9	8
Total	25.5	30	37.5	38
Overall Total	271.65	278.4	273.45	302.15

: Not part of our normal working hours : Person was sick

Weekly Report

LAPS Group

Summary of project week 11

28th March⟶ 3rd April

24th May 2020

Contents

LAPS
LOW-ALTITUDE PATHFINDING SERVICE

1. General

This week was our first week after our second presentation, and therefore we focussed on developing our product. As such we did not do too much as a group other than settling into our new work routine working from home. We decided to double our meetings to twice a day to compensate for working from our homes.

2. Håkon Jordet

This week I implemented a few things which came up after the second presentation. Namely, I vastly improved caching of jobs. Before, we kept the results of a pathfinding job for quite a while, but didn't really do anything with it unless someone requested the same exact job ID back again. I made it such that if a submitted pathfinding job is identical to a previous one, it will return the ID of that job. If the job is complete, the client will receive the result immediately.

I also worked on improved validation. Before, a pathfinding module would crash if someone input invalid coordinates for the path. Now the server will reject the job if the coordinates are outside the map instead of passing it along to the pathfinding module, probably taking it down.

Other than that, I have been working on uploading modules. They've taken on many forms over the week, but now I have a format which I'm happy with. A user only has to provide a main.py Python file, and a requirements.txt which is a list of pip packages which are needed to run the module. These are uploaded together as a tarball, which allows multi-file modules. After being uploaded, we build a runtime Docker image on the server which executes the module.

3. Even Thonhaugen Røraas

After the second presentation one of the feedback was our documentation, the frontend was definitely lacking. Some time have been spent on document features that where implemented but not documented. Sometime was also spent given better comments of what the code does in the code itself. Clutter and things no longer used was removed. While going through code mistakes and bugs where found. Some time was spent fixing these.

4. Henrik Thue Strocka

This i spent continuing to investiga Graph Neural Networks and how to design a structure for our problem. The idea of a graph neural network in itself is not too complex and is not

too hard to implement, except all the boilerplate described before. But the big problem is how to implement convolution on graphs. Because there is no implicit topological structure for a graph defining the kernel is hard.

However there have been done research on this topic and there are solutions out there to solve this problem of bringing convolution to the graph domain. And this week i was able to install the needed libraries in order to run these solutions. It was some struggle with some cuda version compatability but i can now run the library without any errors. So for next week i hope to get an Graph Convolution implementation running.

5. Vetle André Hoffmeyer Neumann

This week I have continued on the reinforcement learning agent, and have finally got it working with Double Q-Learning. The agent does not seem to learn the environment very well, although it is learning it is just not learning how to get a good reward.

The current cost function gives negative reward ("punishment") on every step, where the amount of negative reward is the height at that point, with the only way to end the episode being reaching the goal. Based on this reward function it is learning that the lowest points on the map, and staying in those points, grants it highest reward. As a result of this it fails to reach the goal.

In the next week I will focus on trying to use Dijkstra to assess its reward instead. I will also fix the environment rendering, as it's very makeshift at the moment and not in the Jupyter Notebook which makes it hard to show to other people.

A. This week's working hours

Date	Even	Henrik	Håkon	Vetle
Sat. 28th March	0	0	0	3
Sun. 29th March	0	0	0	2
Mon. 30th March	3	0	9	7
Tue. 31st March	5	0	8	5
Wed. 1st April	6	0	8	7
Thu. 2nd April	6	0	8	6
Fri. 3rd April	5.5	0	9	7
Total	25.5	0	42	37
Overall Total	0	0	367.45	388.75

: Not part of our normal working hours : Person was sick

Weekly Report

LAPS Group

Summary of project week 14

18th April⟶ 24th April

24th May 2020

Contents

LAPS
LOW-ALTITUDE PATHFINDING SERVICE

1. General

This week is our first week where all members of the group can solely focus on the bachelor project. However the corona situation continues to affect us and we have decided as a group to make some changes to our project model. Because before the outbreak we would sit in the same room and interact with each other for at least 7 hours a day we would get much more exposure to each other's code. With our only scheduled communication being reduced to our daily meetings the divided nature of our task has been exacerbated. Therefore the decision was made to reduce the importance of QA and instead have the two teams approve the code internally.

This will reduce our understanding of each other code but we feel this change is needed to keep us agile in this uncertain future. We still plan on updating each other on our progress during our daily meetings.

2. Håkon Jordet

This week I spent a lot of time on the documentation. I looked through most of the documentation for my parts of the system, and rewrote it. I also made sure that as many things as possible have been properly documented, in particular the module handling code.

While going through the documentation and documenting things I've done before, I noticed that there were several places where the code could be better, and I made sure to spend the time to improve these places. In particular, I completely changed how job polling works. I had misunderstood how the Redis command RPOPLPUSH works, which meant I could use it to poll for job results again. I thought it would reset the expiry of the list if popping and pulling from the same list when that list only has one element. Turns out it doesn't, so I could remove the rather complex polling with this command. This means less CPU usage.

3. Even Thonhaugen Røraas

Leftover work from before the exam, was finished. Mainly adding a dropdown menu to show selectable maps that are available. Work on the frontend implementation of the admin panel has begun. This will include letting the user upload maps, modules and starting and stopping modules. To limit access to unwanted users a authentication feature is being worked on as well. Work initially started on the upload map feature but had to be redirected towards the login feature as all request made had to be done from an admin account which meant the feature could not be tested before the login feature. There where problems with implementing the login feature. Mostly coming from the development being done locally and request being made towards the server. But when the login feature was made it was test on the server towards the server. This caused issues developing the login feature.

4. Henrik Thue Strocka

The first part of this week was spent creating a data generator that converted our dataset into graph form. This was quite labor intensive but it had to be done and after struggling with PyTorch's tensors for a couple of days I had a working program. The dataset itself is pretty much identical to the last ones but it employs no blurring and is entirely node based. The node being as described by Vetle in our first presentation.

With the data generator done I left it on overnight to generate some sufficient data I went to work creating a custom network. The network I created was an extension of the pre built SplineConv network that did not have self-loops. Self loops are simply edges on a graph that connect to itself, this was removed so the pathfinding could not be stuck in a loop.

However initial results was disappointing and its clear that more tweaking is needed, that is my aim for next week.

5. Vetle André Hoffmeyer Neumann

This week after investigating cost functions for the Reinforcement Learning, I discovered that although it was learning the cost function somewhat, but would perform many steps in the wrong direction, often taking up to a thousand extra steps for getting from start to stop on a 40x40 grid. To solve this the problem was simplified even further, by ignoring heightmap and only focusing on making the RL agent move in the correct direction.

Small things were changed as well, such as the way the images are rendered for the neural network, as the images were being rescaled to fit the convolutional network, and this rescaling in the case that it was not a 1:1 rescale would blur the image.

The way the environment rendered the gym was also changed to allow the rendering to be a subplot instead of a main plot, and as a result a better graph plot of the environment could be made.

A. This week's working hours

Date	Even	Henrik	Håkon	Vetle
Sat. 18th April	0	0	0	0
Sun. 19th April	0	0	0	0
Mon. 20th April	7.5	8	7	5
Tue. 21st April	6.5	7	7	6
Wed. 22nd April	7	7	8	7
Thu. 23th April	7	9	8	7
Fri. 24th April	5	10	8	5
Total	33	41	38	30
Overall Total	392.65	432.4	462.95	435.75

: Not part of our normal working hours : Person was sick

Weekly Report

LAPS Group

Summary of project week 2

13th January⟶ 20th January

20 January 2020

Contents

1. Completed Tasks

1.1. Documents

This week, we finished setting up our document macros. This allows us to quickly easily, and more importantly consistently typeset tests, requirements, risks and user-stories. The color scheme is not final and is likely to change in the future.

T.1	Requirements	RQ.1	Related	T.2
Passing	Description	Verify milking is managable		
Not Done	Criteria	Passing criteria for the test		

Figure 1: Example test

Figure 1 is an example of a test. *Status* refers to whether or not the test has been fully made. The state of the test is displayed. It displays whether the test is passing, failing or isn't tested yet. The testers are the people responsible for writing and performing the tests. In our case, many tests are going to be automated code tests, and therefore the responsibility is more about ensuring that we have written tests which can be executed automatically as the project goes on, ensuring things do not break.

RQ.1.2	User stories	U.2	Tests	T.1
Discarded	Category	UI		
Priority	Origin	Jan Dyre Bjerknes, Henrik Thue Strocka		
3	Description	A very good description of the requirement. Can be several lines long.		

Figure 2: Example requirement

			Related		
RI.1	Responsible	Henrik Thue Strocka	RI.3		
	Description	A description of the risk at hand	Impact	Prob.	Risk
	Mitigation	How to avoid this risk	3	6	18

Figure 3: Example risk

Figure 3 is an example of a risk. We have

1.2. User stories

We have written a list of user stories for the service in general. They will probably fluctuate a bit between now and the first presentation, but the gist is this:

- The service must consist of a pathfinding algorithm and a user interface.

U.1			
	This is a short and concise user story.		
	Features	Related	Origin
	F.1.69, F.1.1337, F.1.420	U.2	Håkon Jordet

Figure 4: Example risk

- The pathfinding algorithm must produce a flight path between two points in space, using height data. This path must be flyable by the drone, with tweakable drone parameters.
- A calculated path must be displayed on a map in the user interface.

1.3. Discuss possible pathfinding algorithm sthick

1.4. Writing documentation

We worked on documentation and stuff

- Risk
- Features
- User stories
- Tests

1.5. Meeting with Jan

2. Tasks from last week which aren't completed

While a lot of time was spent working on typesetting risks etc., not much time was spent making the document templates look better. We hope to have a logo up soon, so we will be revisiting it when we have to add our new logo.

3. Tasks for next week

3.1. Writing more documentation

We now have a better idea of what our documents should be. Now that we have an initial version of documentation templates and utilities, it will be much easier to sit down and write the documentation we need.

3.2. Define numbering systems

We have talked a little bit about what the identifiers of risks, requirements, user stories and such should be. We have yet to define a system for what the naming should be. We should define a set of rules for how it should be done. This would make them consistent.

In addition, there are more numbering systems which we have to define. What scale does risk probability follow? How many levels of priority should we define? What will our formula for risk factor be? Questions like these need to be answered before we can use this system.

3.3. Create plans for starting the product

We haven't worked on the product yet, which means we have to lay out a plan for starting to work on it. We want to show off our plan forward on the first presentation.

3.4. Going forward with pathfinding

Going forward we have created 4 plans A-D going forward. This is so we always have a backup option if one plan fails, because of the nature of machine learning we will end up in a lot of dead ends. We have also looked into multiple choices for ML frameworks, we have not decided yet but we are leaning towards Torch or Tensor flow.

A. This week's working hours

Date	Even	Henrik	Håkon	Vetle
Mon. Jan 13	6.75	7	7.25	4
Tue. Jan 14	0	0	0	0
Wed. Jan 15	7	7	7	7
Thu. Jan 16	7	7	5.25	7
Fri. Jan 17	7	6.75	8.2	7
Total	27.75	27.75	27.7	25
Overall Total	42	48	49.25	44.75

: Had lectures : Not part of our normal working hours : Sick

Weekly Report

LAPS Group

Summary of project week 7

15th February⟶ 21st February

21 February 2020

Contents

LAPS
LOW-ALTITUDE PATHFINDING SERVICE

1. General

This week work has continued on the minimum viable product. The Frontend/Backend team has working on the most vital of functions for the product, whereas the Machine Learning team has mostly continued their work on the Convolutional Neural Network approach.

1.1. Håkon Jordet

This week, I decided to change the HTTP framework of the backend. It was decided that while Rocket does not yet have a stable asynchronous release, we should build our application around it anyway, as it provides a much better development experience. The time we may(but likely won't) spend fixing breakages is saved by being faster to develop for.

After that, I figured out how we want to share map data in the application. I realized that working with the raw GeoTiff files was cumbersome at best, and since we can disregard most of the geographical data anyway, we could convert it into a more commonly used image format. It was decided to convert any GeoTiff file into a normalized PNG, where the lowest point in the height data is a black pixel, and a fully white pixel being the highest point. This has another benefit in that it makes 150+MiB .tif files into 2.4MiB .png files.

I found out how to store the image data in the database, which the rest of the application can now build upon. As of yet there is no automated importer tool to perform the .tif → .png conversion and load it into the system. So for a while forward, we have to manually convert them and load them up into the database.

Next, I added backend API endpoints to list and retrieve map data in the frontend. They simply return the raw .PNG files.

Finally, I started designing a new job submission scheme. Currently, it is extremely simplistic in that it will return the calculated path immediately. I have split this up into two API calls: One to submit the job, and one to poll for it's status. After the user has submitted a job, the API returns a token. This token is then used to poll for a job's status.

There are a few benefits to this. The main one being the ability to keep previous jobs for a certain time. I store every piece of data about the job in Redis, and I ensure that the job result expires after a given time frame. This is essentially a type of caching.

1.2. Even Thonhaugen Røraas

Finished coordinate sender function of the front end. Started taking advantage of vues component system. Vue allows Html, javascript and css to be placed in a single file rather than the three typical layers they are normally put into. Components are therefore pieces of html, javascript and css that belongs together and fulfill a function. The coordinate sender code was moved into a component. Work have started on map display. So far the service is able to fetch a map from the backend and display it.

1.3. Henrik Thue Strocka

This week was focused on fining the viability of our Convolutional Neural Network for me. As stated in the previous week report it showed signs of promise and i was able to improve the accuracy down to around 7 pixels. This however was not good enough as i discorded even small innacuracies in the midpoint amplify themselves and makes the path useless.

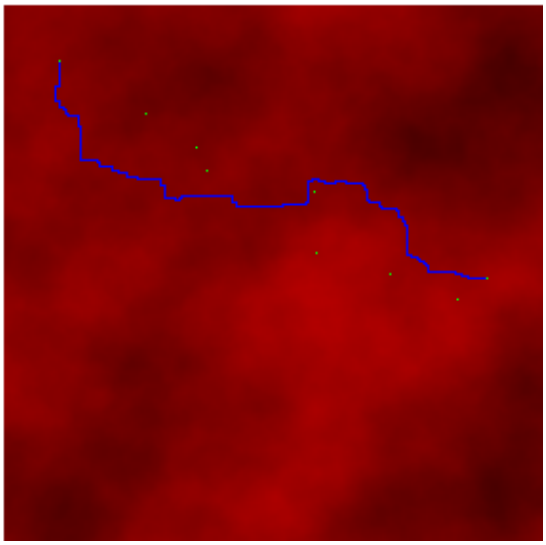


Figure 1: Image showing a ML generated path. Each dot is a midpoint. The blue line is the optimal dijkstra path.

The paths would usually start okay with the first midpoint being okay, but as the next step would use a wrong midpoint it would just get worse and worse. First i belived this was because of a lack of training data, we did have a data generator but it used an un-optimized version of dijkstra and was very slow. So i spent the day and used an optimized version i found on github. This version was around 100x quicker and now i was able to generate up to 10k training data points per hour. However we are now meeting the problem of limited computing power to train the Network.

One interesting thing about the midpoints is that you can see there are some paths that are more often used by the algorithm. The network itself also underwent major changes this week. The only change i found that ended up being benificial for the network was to pass the start and endpoints directly to the fully connected neural network layers, instead of passing them in as separate color channels. This also had the benefit of reducing weights and decreasing training time.

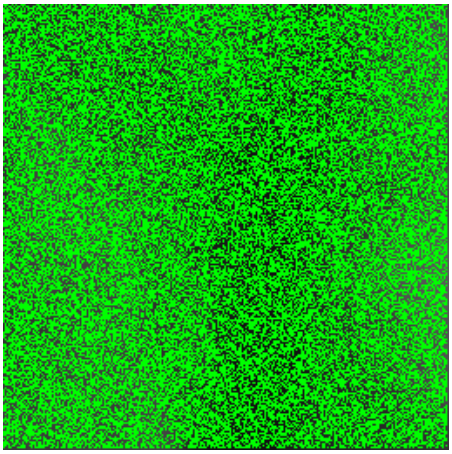


Figure 2: Image showing all start points distributed on the map.

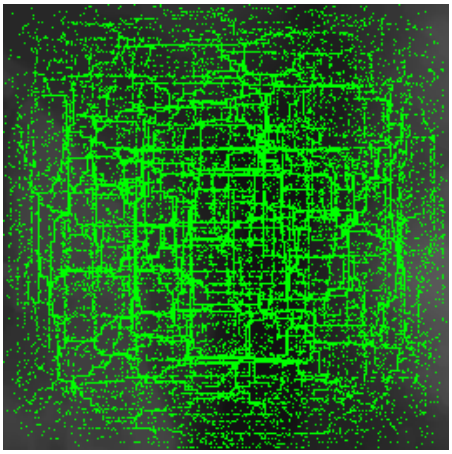


Figure 3: Image showing the midpoint of all paths.

1.4. Vetle André Hoffmeyer Neumann

This week I have continued on machine learning infrastructure, as well as learning how Redis work so that I could make the Python Dijkstra's Algorithm implementation work with the backend, which I did. This is then our first complete pathfinding module.

With infrastructure out of the way, I've began reading up on Reinforcement Learning, and will hopefully have the environment implemented by next week.

2. Tasks for next week

Minimum viable product was not achieved within this week. We are still working on map display in regards to frontend, as well as (backend minimum viable product here).

As work with reinforcement learning has been started on and is now focused, we will hopefully have a simple implementation of that within next the end of next week.

A. This week's working hours

Date	Even	Henrik	Håkon	Vetle
Sat. Feb 15	0	1	0	2
Sun. Feb 16	2	2	0	4.5
Mon. Feb 17	6.5	7	7	7
Tue. Feb 18	1.75	3	0	0
Wed. Feb 19	6.75	7	7	7
Thu. Feb 20	7	6.5	6.5	7.25
Fri. Feb 21	7	7	7	7
Total	31	33.5	27.5	34.75
Overall Total	197.75	197.65	200.95	215.75

: Had lectures : Not part of our normal working hours

Weekly Report

LAPS Group

Summary of project week 4

25th January⟶ 31st January

31 January 2020

Contents

LAPS
LOW-ALTITUDE PATHFINDING SERVICE

1. Completed Tasks

This weekend and the monday the team spent quite some time preparing the documentation in preparation for the first presentation delivery. Afterwards, the Thursday and Wednesday were spent working on the presentation itself, and External Supervisor got us an External Sensor. On the wednesday evening, the day before the presentation, one of the team members got ill and the presentation was postponed to the 7th of February as a result. The Thursday was then mostly spent working on the documentation again as we realised some flaws we had overlooked.

With the documentation mostly in place, technical planning was begun. As the first milestone and the first thing to establish before the parts of the team can begin working seperately is the API, we begun researching API design. We quickly realised that the API we need depends on the software architecture, and we moved over to researching this topic instead. As we will be meeting with Kongsberg Digital on monday where we will discuss deployment of machine learning, we are hoping they might have some thoughts on the subject.

2. Not Completed Tasks

The parts in the documentation which are written as of now have, though subject to change, been completed. The only exception is parts of the background, which might require a bit more reworking.

3. Tasks for next week

The tasks for the next week is primarily meeting Kongsberg Digital, and having the presentation. Other than that we will continue working on the architecture, as well as the API if we get that far.

A. This week's working hours

Date	Even	Henrik	Håkon	Vetle
Sat. Jan 25	5	5.5	7	5
Sun. Jan 26	2.5	8	7.5	5
Mon. Jan 27	7.5	7.5	7.5	7.5
Tue. Jan 28	4.5	4	0	4.5
Wed. Jan 29	5.5	4.5	4.5	4.5
Thu. Jan 30	6	0	6.5	9
Fri. Jan 31	6.5	0	6.5	6.5
Total	37.5	29.5	39.5	42.5
Overall Total	72.75	77.65	78.95	75.5

: Had lectures : Not part of our normal working hours : Sick

Weekly Report

LAPS Group

Summary of project week 5

1st February⟶ 7th February

8 February 2020

Contents

LAPS
LOW-ALTITUDE PATHFINDING SERVICE

1. Completed Tasks

Monday 3rd of February we had plans with our customer to take a trip to Kongsberg Digital in Asker. They have experience researching and deploying machine learning based solutions in a professional capacity. The hope was that our bachelor group could learn a bit about what it takes to set up machine learning based services.

The trip started with all parties having a small presentation about themselves and what they did, in addition our group presented our problem and our ideas for how to solve them. Then following the presentation we discussed Kongsberg Digital's five point plan to decide if machine learning is a good solution to a project. The trip was a success and the group learned a lot about what it takes to solve problems using machine learning and deploy it.

Because our first presentation was moved to this Friday because of sickness the rest of the week was spent preparing for the presentation and modifying our documentation. As our goal was to be able to start working once the first presentation was completed.

2. Not Completed Tasks

Because of sickness no progress was done on writing machine learning modules or writing the service.

3. Tasks for next week

For next week the customer is out of town and we cannot have a meeting with him. However we have decided on a meeting the 19th of February, so our goal is to have something concrete to show him at that meeting. So the goal for next week is to have at least some progress in both machine learning and the service aspect of our project.

A. This week's working hours

Date	Even	Henrik	Håkon	Vetle
Sat. Jan 25	0	0	0	0
Sun. Jan 26	0	0	0	4
Mon. Jan 27	7	7	7	7
Tue. Jan 28	0	0	0	0
Wed. Jan 29	6	7	7	7
Thu. Jan 30	8.25	7	7	9
Fri. Jan 31	6	6	6	6
Total	27.25	27	27	33
Overall Total	137.5	134.15	145.45	151

: Had lectures : Not part of our normal working hours

Weekly Report

LAPS Group

Summary of project week 3

13th January⟶ 20th January

20 January 2020

Contents

LAPS
LOW-ALTITUDE PATHFINDING SERVICE

1. Completed Tasks

This week has been a full-on documentation week to get everything done for the first presentation next week. Because of this we have not been working too much on features but instead documenting and defining how everything will be documented.We started the week with agreed on the last details on how our identification system is supposed to work and documented it.

We also did a bit of work on the way forward after the presentation. For the machine learning part of our project we formulated 4 rough plans for what to do going forward. These plans are very broad and will only serve as guidelines, we decided to not go into too much detail as documentation is our focus this week.

The frontpage and structure for our binder was also finalized this week. We settled on a frontpage with a table of contents with each category of documents. And at each category we start with an explanation of how this is documented.

This week we started on our presentation by laying out a basic structure and deciding on what to say where. We did not go into too much detail but we wanted a general layout to compare when we went to see the first presentation.

2. Tasks for next week

Next week we have the deadline for our documentation handing before our first presentation. So before Thursday documentation will have first priority. Then for the rest of the week we will finalize our presentation and prepare for the to present it.

A. This week's working hours

Date	Even	Henrik	Håkon	Vetle
Sat. Jan 18	0	0	0	0
Sun. Jan 19	0	0	0	0
Mon. Jan 20	6.75	7	7	7
Tue. Jan 21	1	0	0	0
Wed. Jan 22	7.25	7.15	7	6.75
Thu. Jan 23	8.75	8.5	8	10
Fri. Jan 24	7	7	7	7
Total	30.75	29.65	29	30.75
Overall Total	72.75	77.65	78.95	75.5

: Had lectures : Not part of our normal working hours : Sick

Weekly Report

LAPS Group

Summary of project week 14

2nd May⟶ 8th May

24th May 2020

Contents

LAPS
LOW-ALTITUDE PATHFINDING SERVICE

1. General

This is the first week of our two week long focus on documentation before the hand-in date. So we have decided to stop writing our individual contributions for this week and next since we are all working mostly on documentation.

A. This week's working hours

Date	Even	Henrik	Håkon	Vetle
Sat. 2nd May	0	0	3	3.5
Sun. 3rd May	4	11	3	6
Mon. 4th May	6	9	9	11
Tue. 5th May	6	12	9	10
Wed. 6th May	9	8	8	6
Thu. 7th May	8.5	12	9	9
Fri. 8st May	6	8	9.5	7
Total	45.5	56	44.5	50.5
Overall Total	495.15	566.4	574.45	566.75

: Not part of our normal working hours : Person was sick

Visualization

May 24, 2020

Author: Henrik Strocka

1 Deep Star V1 Visualization

This notebook is a collection of all methods used to visualise DeepStar V2

[1]:
```
import os, sys

module_path = os.path.abspath(os.path.join('../..'))
if module_path not in sys.path:
    sys.path.append(module_path)

import loader, torch, torchvision

from torchvision import transforms
from PIL import Image, ImageDraw

from DeepStar.V2.Model import DeepStar
from Dijkstra.Dijkstra import *
```

```
0.00934910774230957
(357.7999999999999, ((0, 29), ((7, 14), ((5, 5), ()))))
```

1.1 Visualization Settings

Img Size: Dimensions of the map to be used. **Save File**: Pytorch weight dictionary to load the network weights from. **Map FIle**: Image of the map to be used for visualization. **Seed**: GPU seed to use for computation.

[2]:
```
img_size = (256, 256)
save_file = "models/deep_star_v2.pt"
map_file = "data/images/map.png"
seed = 0
```

1.2 Load Model

Load the specified deepstar model and set the seed to ensure a determenistic output.

[40]:
```
torch.manual_seed(seed)

net = DeepStar()
net.load_state_dict(torch.load(save_file))
```

[40]:
```
<All keys matched successfully>
```

1.3 Data Visualization

1.3.1 Helper Functions

Guess To Point

Converts a normalized guess between [0 - 1] to a pixel value.

G: Guess tensor.

Img Size: Size of image used to guess.

Return: Pixel coordinate of the guess.

[4]:
```
to_tensor = transforms.ToTensor()

def guess_to_point(g, img_size):
    return (round(g[0][0].item() * img_size[0]), round(g[0][1].item() *␣
 ↪img_size[0]))
```

Guess Point

Uses the loaded network to guess the midpoint between a start and a stop point. Start and Stop position will be normalized in the function.

Net: Network to use for the guess.

Img: Map to feed the network.

Start: Path start point.

Stop: Path stop point.

Img Size: Map size.

Return: A normalized guess at the midpoint on the map.

[5]:
```
def guess_point(net, img, start, stop, img_size):
    img_tensor = to_tensor(img).unsqueeze(0)
    sx, sy = start
    ex, ey = stop
```

```
    sx = sx / img_size[0]
    sy = sy / img_size[1]

    ex = ex / img_size[0]
    ey = ey / img_size[1]

    pos_tensor = torch.FloatTensor([sx, sy, ex, ey]).unsqueeze(0)
    return guess_to_point(net(img_tensor, pos_tensor), img_size)
```

Load World

Load a normalized heightmap world of nodes from a grayscale image.

Img: Pillow image to be converted.

Return: 2D Normalized world array.

[6]:
```
def load_world(img):
    world = []
    pixels = img.load()

    for x in range(img_size[0]):
        world.append([])
        for y in range(img_size[1]):
            if isinstance(pixels[x, y], int):
                value = pixels[x, y]
            else:
                value = pixels[x, y] if len(pixels[x, y]) == 1 else pixels[x,␣
 ↪y][0]

            world[x].append(Node(float(value) / 255, (x, y)))

    return world
```

Add Point

Add a point to a pillow image with the given color.

Img: Pillow image to use.

Point: Pos for the point.

Color Space: Index of color channel to add the point to.

[7]:
```
def add_point(img, point, color_space):
    x, y = point
    img.paste(color_space, (x - 1, y - 1, x, y))
```

Add Point

Add the start and endpoint to the image in their separate color channel.

Img: Pillow image to use.

Start: Start point to add to the Green channel.

End: Stop point to add to the Red channe.

[8]:
```
def add_points(img, start, end):
    sx, sy = start
    ex, ey = end

    img.paste((0, 255, 0), (sx - 1, sy - 1, sx, sy))
    img.paste((0, 0, 255), (ex - 1, ey - 1, ex, ey))
```

Get Midpoints

Get a midpoint guess betwean each index pair in the path list.

Net: Netowrk to use for the guess.

Path: List of points, a guess will be made between each pair of points.

Return: A path with the midpoint for each index par inserted between them.

[ ]:
```
def get_midpoints(net, path, start, stop):
    new_path = []

    for i in range(len(path) - 1):
        with Image.open(map_file) as img:
            g = guess_point(net, img.convert('L'), path[i], path[i + 1],␣
 ↪img_size)
            new_path.append(g)

    return_path = []
    for i in range(len(path)):
        return_path.append(path[i])

        if (i < len(new_path)):
            return_path.append(new_path[i])

    return return_path
```

1.4 Test Loaded Network

Gues midpoint between a start and a stop point, using the image specified in the settings at the beginning.

[52]:
```
start = (52, 25)
stop = (200, 200)

with Image.open(map_file) as img:
```

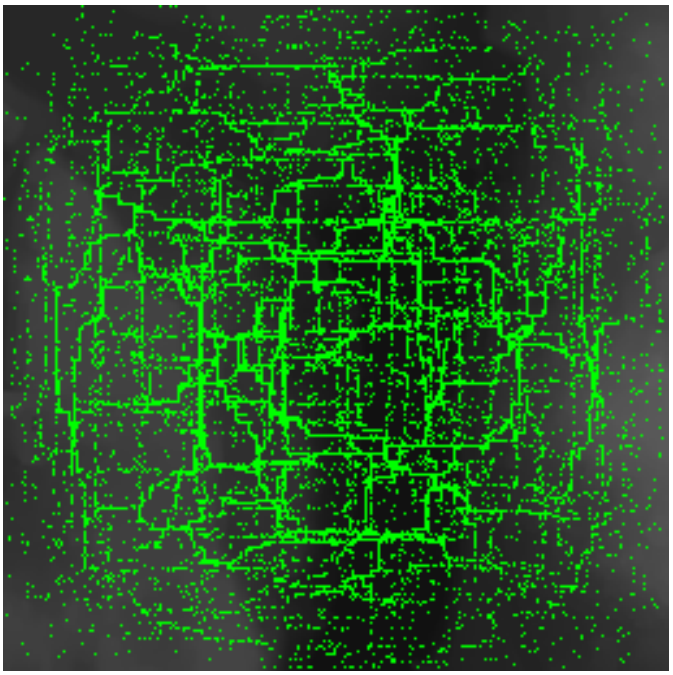
```
    img = img.convert('L')
    guess = guess_point(net, img, start, stop, img_size)

    world = load_world(img)
    edges = get_edges(world)
    dijkstra_path = dijkstra(edges, start, stop)
    path = get_path(dijkstra_path)
    midpoint = get_path_midpoint(path)

    print(midpoint)
    print(guess)
    print(math.sqrt(math.pow(midpoint[0] - guess[0], 2) + math.pow(midpoint[1]␣
↪- guess[1], 2)))
```

```
(168, 70)
(120, 114)
65.11528238439882
```

1.5 Visualize Data Distribution

Visualize a data column in the csv file. Change the column name to view different data. Currently there is only three columns, Start, End and Midpoint.

[11]:
```
column_name = "Midpoint"

import pandas as pd
csv = pd.read_csv("data/images/data.csv", encoding = "UTF-8")

def to_tuple(t):
        return tuple(map(int, t.replace('(','').replace(')', '').split(', ')))

with Image.open(map_file) as img:
    img = img.convert("RGB")
    for p in csv[column_name]:
        add_point(img, to_tuple(p), (0, 255, 0))

    display(img)
```

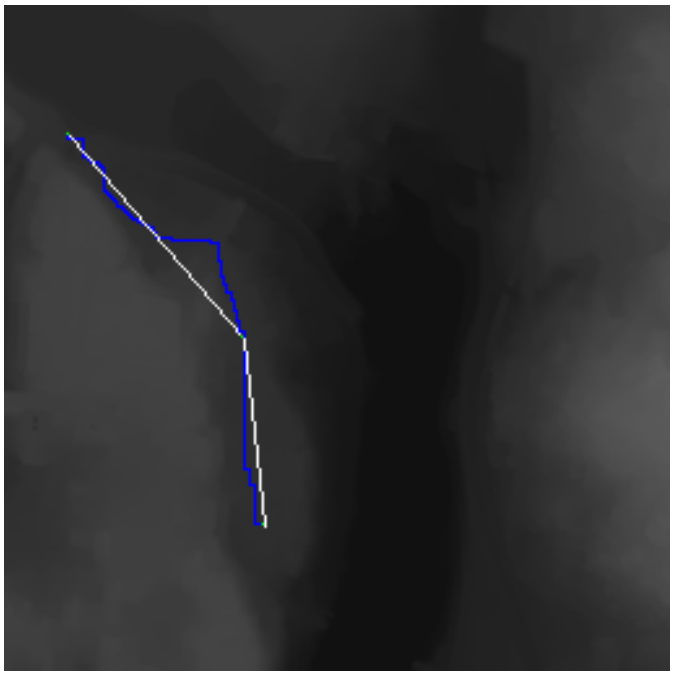


1.6 Test Run On The Network

Test run on the selected mab between the start and stop point.

[45]:
```
start = (25, 50)
stop = (100, 200)

path = [start, stop]

# Calculate the path using DeepStar
for i in range(1):
    path = get_midpoints(net, path, start, stop)

with Image.open(map_file) as img:
    # Load Image
    world = load_world(img)
    pixels = img.load()

    # Run djikstra on world loaded from image.
    img = img.convert("RGB")
    edges = get_edges(world)
    dijkstra_path = dijkstra(edges, start, stop)

    # Add djikstra path to image.
```

```
    for d in get_path(dijkstra_path):
        add_point(img, d, (0, 0, 255))

    # Draw lines between each pixel guessed by deepstar
    draw = ImageDraw.Draw(img)
    draw.line(path)

    #del draw
    for p in path:
        add_point(img, p, (0, 255, 0))

    display(img)
```



DataGeneration

May 24, 2020

Author: Henrik Strocka

1 DeepStar V1 Data Generation

[ ]:
```
import os, sys
module_path = os.path.abspath(os.path.join('../..'))
if module_path not in sys.path:
    sys.path.append(module_path)

import loader, random, noise, csv

from PIL import Image
from Dijkstra.Dijkstra import *
```

1.1 Data Generation Settings

Data Settings **Shape**: Dimensions of the output image.

Output: Output folder for the data.

Validation: Wether to generate validation data or training data.

Image: How many images to create.

[ ]:
```
shape = (256, 256)
output = "data/"
validation = False
images = 60000
```

Noise Settings Note: Noisemap will not be overitten if it already exists

Scale: Noise scale.

Octaves: How many times to apply the noise.

Persistence: How much the noise amplitue changes for each octave.

Lacunarity: How much the noise frequency changes for each octave.

[ ]:
```
scale = 100.0
octaves = 6
persistence = 0.5
lacunarity = 2.0
```

1.2 Data Generation

1.2.1 Helper Functions

Get Height Calculates the perlin noise height at a given point (i, j).

Return: A float value between -1 and 1

[ ]:
```
def get_height(x, y):
    return noise.pnoise2(x/scale, y/scale, octaves=octaves,␣
 ↪persistence=persistence, lacunarity=lacunarity, repeatx=shape[0],␣
 ↪repeaty=shape[1], base=0)
```

Generate Map Calculates noise values for the entire grid with the dimensions given in the shape setting. The heightvalues are normalized to between 0 and 1.

Return: A grid of dimensions (shape.x, shape.y) that contains normalized perline height noise.

[ ]:
```
def generate_map():
    grid = []
    for i in range(shape[0]):
        grid.append([])
        for j in range(shape[1]):
            n = get_height(i, j)
            n += 1
            n /= 2

            grid[i].append(Node(n, (i, j)))

    return grid
```

Get Node Get node at point.

World: World of nodes to use.

Point: Coordinates of the node to load.

Return: Node at point give.

[ ]:
```
def get_node(world, point):
    return world[point[0]][point[1]]
```

Get Random Points Calculates a random start and stop inside the shape given in the settings.

Return: Start and End point (Not node).

[ ]:
```
def get_random_points():
    start = (random.randrange(0, shape[1]), random.randrange(0, shape[1]))
    end = (random.randrange(0, shape[1]), random.randrange(0, shape[1]))

    return start, end
```

Create Image Create a training image for DeepStar. The R axis contains the heightdata from the grid, G and B axies contains the start and stop point respectivly.

Return: RGB training image fro DeepStar.

[ ]:
```
def create_image(grid):
    im = Image.new("L", shape)
    pixels = im.load()

    for i in range(shape[0]):
        for j in range(shape[1]):
            intNoise = int(grid[i][j].value * 256)
            pixels[i, j] = intNoise

    return im
```

1.3 Load Image

Create a world of nodes from a pillow image.

Img: Image to create world from.

Return: Node world that is compatible with dijkstra and deep star.

[ ]:
```
def load_image(img):
    world = []
    pixels = img.load()

    for x in range(shape[0]):
        world.append([])
        for y in range(shape[1]):
            if isinstance(pixels[x, y], int):
                value = pixels[x, y]
            else:
                value = pixels[x, y] if len(pixels[x, y]) == 1 else pixels[x,␣
 ↪y][0]

            world[x].append(Node(value / 256, (x, y)))
```

```
    return world
```

1.4 Create Files

Make sure the necessary files are present, if they are not generate them. This cell also makes sure a world has been created and its edges been retrived.

[ ]:
```
# Handle world generation
world = generate_map()
edges = get_edges(world)

subfolder = "images"
if (validation):
    subfolder = "validation"

# Generate required files.
if (os.path.isfile(f'{output}{subfolder}/map.png')):
    img = Image.open(f'{output}{subfolder}/map.png')
    world = load_image(img)

if (not os.path.isfile(f'{output}{subfolder}/map.png')):
    img = create_image(world)
    img.save(f'{output}{subfolder}/map.png')

if (not os.path.isfile(f'{output}{subfolder}/data.csv')):
    with open(f'{output}{subfolder}/data.csv', 'w') as file:
        writer = csv.writer(file)
        writer.writerow(["Start", "Stop", "Midpoint"])
```

1.5 Generate Data

Generating the actual data and writing it to the csv. As the heightmap is always the same it does not need to write a image for each path and can store only the necessary values in a csv file. The path are calculated using dijkstra to make sure its the optimal path.

[ ]:
```
with open(f'{output}/{subfolder}/data.csv', 'a') as file:
    writer = csv.writer(file)
    for image in range(images):
        # For each path get a random start and stop point.
        start, end = get_random_points()

        # Calculate the optimal path
        path = dijkstra(edges, start, end)
        try:
```

```
            path = get_path(path)
        except:
            continue

        # Sometimes the path will be none, just ignore it if that is the case␣
↪(BUG)
        if path is None:
            continue

        # Calculate the median point of the path.
        midpoint = get_path_midpoint(path)
        if len(path) < 11:
            midpoint = path[len(path) - 1]
        else:
            midpoint = path[10]

        # Save the generated data to the master CSV file.
        writer.writerow([start, end, midpoint])
        print(image)
```

Model

May 24, 2020

Author: Henrik Strocka

1 DeepStar V2 Model Definition

This notebook is responsible for defnining the version network structure and the accompanying data loader.

[1]:
```
import time, torch, torch, csv
import torch.nn as nn
import pandas as pd
import torch.optim as optim
import torch.nn.functional as F
import torch.utils.data as utils

from torchvision import transforms
from PIL import Image
```

1.1 DeepStar Network Definition

DeepStar consists of up to five convolution layers. This can easely be commented out depending on hardware limitations and software needs. It uses a two layer feed forward fully connected classification network to make the predictons.

Get Optim: Currently we are using the SGD optimizer function. https://pytorch.org/docs/stable/optim.html#torch.optim.SGD

Get Loss: Currently we are using the Csross Entry Loss. https://pytorch.org/docs/stable/nn.html#torch.nn.CrossEntropyLoss

[3]:
```
class DeepStar(nn.Module):
    # Network creation
    def __init__(self, prediction_size):
        super(DeepStar, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=4, stride=2, padding=0)
        self.pool1 = nn.MaxPool2d(kernel_size=1, stride=1, padding=0)
```

```
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=2, padding=2)
        self.pool2 = nn.MaxPool2d(kernel_size=1, stride=1, padding=0)

        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, stride=2, padding=1)
        self.pool3 = nn.MaxPool2d(kernel_size=1, stride=1, padding=0)

        self.conv4 = nn.Conv2d(128, 256, kernel_size=3, stride=2, padding=1)
        self.pool4 = nn.MaxPool2d(kernel_size=1, stride=1, padding=0)

        self.conv5 = nn.Conv2d(256, 512, kernel_size=1, stride=2, padding=0)
        self.pool5 = nn.MaxPool2d(kernel_size=1, stride=1, padding=0)

        self.conv_out = 512 * 9 * 9
        px, py = prediction_size

        self.fc1 = nn.Linear(self.conv_out + 4, px*py*2)
        self.dropout1 = nn.Dropout(p=0.25)
        self.fc2 = nn.Linear(px*py*2, px*py + 1)

    def __name__(self):
        return "DeepStar"

    # Loss function this models uses
    def get_loss(self):
        return nn.CrossEntropyLoss()

    # Optimization algorithm used by this model
    def get_optim(self, lr, momentum):
        return optim.SGD(self.parameters(), lr=lr, momentum=momentum)

    # Instructions for how to do the forward pass
    def forward(self, img, points):
        img = self.forward_conv(img)

        l = img.view(-1, self.conv_out)
        l = torch.cat((l, points), 1)

        l = F.relu(self.fc1(l))
        l = self.dropout1(l)

        return self.fc2(l)

    def forward_conv(self, img):
        img = F.relu(self.conv1(img))
        img = self.pool1(img)
```

```
        img = F.relu(self.conv2(img))
        img = self.pool2(img)

        img = F.relu(self.conv3(img))
        img = self.pool3(img)

        img = F.relu(self.conv4(img))
        img = self.pool4(img)

        img = F.relu(self.conv5(img))
        img = self.pool5(img)

        return img
```

1.2 DataLoader

This dataloader is identical to DeepStar V1. See there for more info.

[4]:
```
class PathDataLoader(utils.Dataset):
    def __init__(self, data_dir, prediction_size):
        self.map = f'{data_dir}/map.png'
        self.data_path = f'{data_dir}/data.csv'
        self.to_tensor = transforms.ToTensor()
        self.data = pd.read_csv(self.data_path, encoding = "UTF-8")
        self.size = prediction_size

    def __len__(self):
        return len(self.data["Start"])

    def __getitem__(self, idx):
        with Image.open(self.map) as img:
            imgWidth, imgHeight = img.size
            width, height = self.size

            sx, sy = self.to_tuple(self.data["Start"][idx])
            ex, ey = self.to_tuple(self.data["Stop"][idx])
            mx, my = self.to_tuple(self.data["Midpoint"][idx])

            label = round((mx / imgWidth) * width) + (width - 1) * round((my /␣
↪imgHeight) * height)

            img_tensor = self.to_tensor(img)
            pos_tensor = torch.FloatTensor([sx / imgWidth, sy / imgHeight, ex /␣
↪imgWidth, ey / imgHeight])
            label_tensor = torch.LongTensor([label])
```

```
            #print(f'{img_tensor.size()}-{pos_tensor.size()}-{label_tensor.
↪size()}')

            return img_tensor, pos_tensor, label_tensor

    def to_tuple(self, t):
        return tuple(map(int, t.replace('(','').replace(')', '').split(', ')))
```

Model

May 24, 2020

Author: Henrik Strocka

1 DeepStar V1 Model Definition

This notebook is responsible for defnining the version network structure and the accompanying data loader.

[ ]:
```
import time, onnx, torch, torch, csv
import torch.nn as nn
import pandas as pd
import torch.optim as optim
import torch.nn.functional as F
import torch.utils.data as utils

from torchvision import transforms
from PIL import Image
```

1.1 DeepStar network definition

Version one of DeepStar consists of three convolutional layers. These layers are max pooled with a kernel size of one, this was because if max pooling was removed it preformed significantly worse. The convolution layers currently gives 256 layers of 20x20 feature maps. These are then fed into a fully connected neural network for classification. The output of the model is two normalized values between [0-1] one for X and one for Y.

Get Loss: The function used by this model, currently the MSE loss function is used. Read more at https://pytorch.org/docs/stable/nn.html#mseloss.

Get Optim: The optimisation algorithm used by the model, currently the Adadelta algorithm is used. Read more at https://pytorch.org/docs/stable/optim.html#torch.optim.Adadelta

[ ]:
```
class DeepStar(nn.Module):
    # Network creation
    def __init__(self, do_pool = True):
        super(DeepStar, self).__init__()
```

```
        #Three convolution layers with their accompanying pooling layer.
        self.conv1 = torch.nn.Conv2d(1, 64, kernel_size=7, stride=3, padding=0)
        self.pool1 = torch.nn.MaxPool2d(kernel_size=1, stride=1, padding=0)

        self.conv2 = torch.nn.Conv2d(64, 128, kernel_size=4, stride=2,␣
↪padding=0)
        self.pool2 = torch.nn.MaxPool2d(kernel_size=1, stride=1, padding=0)

        self.conv3 = torch.nn.Conv2d(128, 256, kernel_size=3, stride=2,␣
↪padding=0)
        self.pool3 = torch.nn.MaxPool2d(kernel_size=1, stride=1, padding=0)

        self.conv_out = 256 * 20 * 20
        self.do_pool = do_pool

        # Three feed forward fully connected classification layer.
        self.fc1 = torch.nn.Linear(self.conv_out + 4, 64)
        self.fc4 = torch.nn.Linear(64, 32)
        self.fc5 = torch.nn.Linear(32, 2)

    def __name__(self):
        return "DeepStar"

    # Loss function this models uses
    def get_loss(self):
        return torch.nn.MSELoss()

    # Optimization algorithm used by this model
    def get_optim(self, rho, lr, weight_decay):
        return optim.Adadelta(self.parameters(), rho=rho, lr=lr,␣
↪weight_decay=weight_decay)

    # Instructions for how to do the forward pass
    def forward(self, img, points):
        img = self.forward_conv(img)

        l = img.view(-1, self.conv_out)
        l = torch.cat((l, points), 1)

        l = F.relu(self.fc1(l))
        l = F.relu(self.fc4(l))
        return self.fc5(l)

    # Convolution layers forward pass
    def forward_conv(self, img):
        img = F.relu(self.conv1(img))
```

```
        img = self.pool1(img)

        img = F.relu(self.conv2(img))
        img = self.pool2(img)

        img = F.relu(self.conv3(img))
        img = self.pool3(img)

        return img
```

1.2 DataLoader

To be able to train the network as defined above it needs a way to load the data into memory. This is the purpose of this custom data loader. Currently we only use one map and one csv file for all paths generated on that map. The map is a grayscale heightmap normalzied to between 0 and 1. The data is a list of start and stop points.

[ ]:
```
class PathDataLoader(utils.Dataset):
    def __init__(self, data_dir):
        self.map = f'{data_dir}map.png' # Path to map used to train on
        self.data_path = f'{data_dir}data.csv' # Path to csv file containing␣
↪all generated data
        self.to_tensor = transforms.ToTensor() # Functino for converting pillow␣
↪images to pytorch tensors
        self.data = pd.read_csv(self.data_path, encoding = "UTF-8") # Read CSV␣
↪data file
        self.img_tensor = self.to_tensor(Image.open(self.map)) # Convert map to␣
↪pytorch tensor

    # Get the length of this dataset
    def __len__(self):
        return len(self.data["Start"])

    # Get data at index
    def __getitem__(self, idx):

        # Read start, stop and path midpoint.
        sx, sy = self.to_tuple(self.data["Start"][idx])
        ex, ey = self.to_tuple(self.data["Stop"][idx])
        mx, my = self.to_tuple(self.data["Midpoint"][idx])

        img_tensor = self.img_tensor # Map converted to a pytorch tensor
        input_tensor = torch.FloatTensor([sx / 256, sy / 256, ex / 256, ey /␣
↪256]) # Input start and stop normalise to [0-1] and converted to tensor
        expected_tensor = torch.FloatTensor([mx / 256, my / 256]) # Expected␣
↪output converted to tensor
```

```
        return img_tensor, input_tensor, expected_tensor

    # Parse csv file tupels to python tupels
    def to_tuple(self, t):
        return tuple(map(int, t.replace('(','').replace(')', '').split(', ')))
```

Visualization

May 24, 2020

Author: Henrik Strocka

1 Deep Star V2 Visualization

This notebook is a collection of all methods used to visualise DeepStar V2

[ ]:
```
import os, sys

module_path = os.path.abspath(os.path.join('../..'))
if module_path not in sys.path:
    sys.path.append(module_path)

import loader, torch, torchvision, matplotlib, time

import numpy as np
import torch.nn as nn
import matplotlib.pyplot as plt

from PIL import Image, ImageDraw
from torchvision import transforms
from IPython.display import clear_output

from Dijkstra.Dijkstra import *
from DeepStar.V3.Model import DeepStar
```

1.1 Visualization Settings

Img Size: Dimensions of the map to be used.

Save File: Pytorch weight dictionary to load the network weights from.

Map FIle: Image of the map to be used for visualization.

Seed: GPU seed to use for computation.

[ ]:
```
img_size = (256, 256)
prediction_size = (32, 32)

load_file = False
save_file = "Trained Models/deep_star_v2.pt"
map_file = "../V2/data/images/map.png"
seed = 0
```

1.2 Load Model

Load the specified deepstar model and set the seed.

[ ]:
```
torch.manual_seed(seed)

net = DeepStar(prediction_size)

if load_file:
    net.load_state_dict(torch.load(save_file))
```

1.3 Visualize Layer Filters

Loop through the layers and display each filter as an image. Each pixel represents the weight in that filter. This image is just the raw weights and are not meant to be human readable, see next visualisation for a better overview.

[ ]:
```
layers = 5 # How many layers of convolution there is
conv_start = 0
conv_stop = 10 # How deep you want to visualize the filters

# Create a numpy figure.
fig, ax = plt.subplots(layers, conv_stop - conv_start)
visited = 0
for i, layer in enumerate(net.state_dict()):
    if "bias" in layer:
        continue

    if visited >= layers:
        break

    visited += 1
    for a, filt in enumerate(net.state_dict()[layer][conv_start:conv_stop]):
        axis = ax[visited - 1, a]
        axis.imshow(filt[0, :, :])
        axis.axis('off')
```

1.3.1 Get Activation

Hooks into a forward pass, when the event is invoked the weights will be added to the activation list.

Return: Hook that will be called.

[ ]:
```
activation = {}
def get_activation(name):
    def hook(model, input, output):
        activation[name] = output.detach()
    return hook
```

1.4 Visualize Convolution Activation

Combine the weights and a forward pass to create an activation image. The higher the value in a stop the more the filter is triggered at that stop.

[ ]:
```
layer_name = "conv2" # Layer to hook into for visualisation.

# Start and Stop point to use for this visualizaiton
start = (20, 20)
end = (200, 200)

to_tensor = transforms.ToTensor()
net.conv1.register_forward_hook(get_activation(layer_name)) # Register a hook␣
 ↪to get the image to display

with Image.open(map_file) as img:
    imgWidth, imgHeight = img.size

    sx, sy = start
    ex, ey = end

    img_tensor = to_tensor(img).unsqueeze(0)
    pos_tensor = torch.FloatTensor([sx / imgWidth, sy / imgHeight, ex /␣
 ↪imgWidth, ey / imgHeight]).unsqueeze(0)
    net(img_tensor, pos_tensor)

    act = activation[layer_name].squeeze()
    fig, axarr = plt.subplots(1, act.size(0))
    for idx in range(act.size(0)):
        axarr[idx].imshow(act[idx], interpolation='nearest')
        axarr[idx].axis("off")
```

1.5 Visualize Network Progress

Loop over all layers and create a animation that displays how the network has learned over time.

[ ]:
```
%%capture
layers = 5
conv_start = 0
conv_stop = 10

figs = []
for x in range(5):
    net = DeepStar(prediction_size)

    fig, ax = plt.subplots(layers, conv_stop - conv_start)
    figs.append(fig)

    visited = 0
    for i, layer in enumerate(net.state_dict()):
        if "bias" in layer:
            continue

        if visited >= layers:
            break

        visited += 1
        for a, filt in enumerate(net.state_dict()[layer][conv_start:conv_stop]):
            axis = ax[visited - 1, a]
            axis.imshow(filt[0, :, :])
            axis.axis('off')
```

[ ]:
```
for i, fig in enumerate(figs):
    clear_output(True)
    display(fig)
    time.sleep(1)
```

Model

May 24, 2020

Author: Henrik Strocka

1 DeepStar V3 Model Definition

This notebook is responsible for defnining the version network structure and the accompanying data loader.

[1]:
```
import time, torch, torch, csv
import torch.nn as nn
import pandas as pd
import torch.optim as optim
import torch.nn.functional as F
import torch.utils.data as utils

from torchvision import transforms
from PIL import Image
```

1.1 DeepStart network definition

Get Optim: Currently we are using the SGD optimizer function. https://pytorch.org/docs/stable/optim.html#torch.optim.SGD

Get Loss: Currently we are using the Cross Entropy Loss. https://pytorch.org/docs/stable/nn.html#crossentropyloss

[2]:
```
class DeepStar(nn.Module):
    def __init__(self, prediction_points):
        super(DeepStar, self).__init__()
        self.max_pool = nn.MaxPool2d(kernel_size=2, stride=2)

        self.conv1 = nn.Conv2d(1, 32, kernel_size=5, stride=1, padding=0)
        self.bn1 = nn.BatchNorm2d(32)

        self.conv2 = nn.Conv2d(32, 64, kernel_size=5, stride=1, padding=1)
        self.bn2 = nn.BatchNorm2d(64)
```

```
        self.conv3 = nn.Conv2d(64, 128, kernel_size=5, stride=1, padding=0)
        self.bn3 = nn.BatchNorm2d(128)

        self.conv_out = 128 * 29 * 29
        self.dropout = nn.Dropout(p=0.2)

        self.fc1 = nn.Linear(self.conv_out, 256*13)
        self.fc2 = nn.Linear(256*13, 256*13)
        self.fc3 = nn.Linear(256*13, prediction_points*2)

    def __name__(self):
        return "DeepStar"

    def get_loss(self):
        return nn.CrossEntropyLoss()

    def get_optim(self, lr, momentum):
        return optim.SGD(self.parameters(), lr=lr, momentum=momentum)


    def forward(self, img):
        img = self.forward_conv(img)

        l = img.view(-1, self.conv_out)

        l = F.elu(self.fc1(l))
        l = self.dropout(l)

        l = F.elu(self.fc2(l))
        l = self.dropout(l)

        return self.fc3(l)

    def forward_conv(self, img):
        img = F.elu(self.bn1(self.conv1(img)))
        img = self.max_pool(img)

        img = F.elu(self.bn2(self.conv2(img)))
        img = self.max_pool(img)

        img = F.elu(self.bn3(self.conv3(img)))
        img = self.max_pool(img)

        return img
```

1.2 DataLoader

[5]:
```
class PathDataLoader(utils.Dataset):
    def __init__(self, data_dir):
        self.data_path = f'{data_dir}/data.csv'
        self.image_path = f'{data_dir}/images/'
        self.to_tensor = transforms.ToTensor()
        self.data = pd.read_csv(self.data_path, encoding = "UTF-8")

    def __len__(self):
        return len(self.data["Start"])

    def __getitem__(self, idx):
        with Image.open(f'{self.image_path}{self.data["Map"][idx]}') as img:
            img_tensor = self.to_tensor(img)
            key_tensor = torch.FloatTensor(self.data.iloc[idx, 1:])

            return img_tensor, key_tensor
```

Model

May 24, 2020

Author: Henrik Strocka

1 DeepStar Auto Encoder Model Definition

This notebook is responsible for defnining the version network structure and the accompanying data loader.

[1]:
```
import time, torch, torch, csv
import torch.nn as nn
import pandas as pd
import torch.optim as optim
import torch.nn.functional as F
import torch.utils.data as utils

from torchvision import transforms
from PIL import Image
```

1.1 DeepStar Network Definition

This auto encoder currently cosists of a three layerd decoder and a three layered encoder. It is designd to be trained as an auto-encoder, but by changing decode to False it will only encode the data and output the feature map.

Get Optim: Currently we are using the Adam optimizer function. https://pytorch.org/docs/stable/nn.html#mseloss

Get Loss: Currently we are using the MSE Loss. https://pytorch.org/docs/stable/optim.html#torch.optim.Adam

Use Decoder: Set wether to decode the feature map back to its original image.

Flatten Data: Set wther to flatten the output image, this works with both the feature map and the output image.

[4]:
```
class DeepStarEncoder(nn.Module):
    def __init__(self, decode=True, flatten=False):
        super(DeepStarEncoder, self).__init__()
        self.decode = decode
```

```
        self.flatten = flatten

        self.conv1 = nn.Conv2d(1, 16, 3, padding=1)
        self.conv2 = nn.Conv2d(16, 4, 3, padding=1)
        self.conv3 = nn.Conv2d(4, 2, 3, padding=1)

        self.pool = nn.MaxPool2d(2, 2)

        self.t_conv1 = nn.ConvTranspose2d(2, 4, 2, stride=2)
        self.t_conv2 = nn.ConvTranspose2d(4, 16, 2, stride=2)
        self.t_conv3 = nn.ConvTranspose2d(16, 1, 2, stride=2)

    def __name__(self):
        return "DeepStar"

    def get_loss(self):
        return nn.MSELoss()

    def get_optim(self, lr):
        return optim.Adam(self.parameters(), lr=lr)

    def use_decoder(self, use):
        self.decode = use

    def flatten_data(self, flatten):
        self.flatten = flatten

    def forward(self, img):
        x = F.relu(self.conv1(img))
        x = self.pool(x)

        x = F.relu(self.conv2(x))
        x = self.pool(x)

        x = F.relu(self.conv3(x))
        x = self.pool(x)

        if self.decode:
            x = F.relu(self.t_conv1(x))
            x = F.relu(self.t_conv2(x))
            x = self.t_conv3(x)

        if self.flatten:
            x = x.view(-1, 2048)

        return x
```

1.2 DataLoader

Because we use a custom dataset we have to have a custom data loader. The training data for an auto-encoder is very simple and currently consits of a simple directy of heightmaps it has to recreate.

[ ]:
```
class ImageDataLoader(utils.Dataset):
    def __init__(self, map_folder, data_length):
        self.map_folder = map_folder
        self.data_length = data_length
        self.to_tensor = transforms.ToTensor()

    def __len__(self):
        return self.data_length

    def __getitem__(self, idx):
        with Image.open(f'{self.map_folder}{idx}.png') as img:
            return self.to_tensor(img)
```

DeepStar

May 24, 2020

Author: Henrik Strocka

1 Training Notebook

This is a general purpose model training notebook. This notebook has evolved over time and is used by all version to train their models.

[ ]:
```
import os, sys

module_path = os.path.abspath(os.path.join('../..'))
if module_path not in sys.path:
    sys.path.append(module_path)

# Uncomment this to better debug CUDA errors.
#os.environ['CUDA_LAUNCH_BLOCKING'] = "1"

import loader, time, torch
from DeepStar.V2.Model import DeepStar, PathDataLoader

import torch.utils.data as utils
import matplotlib.pyplot as plt
```

2 Network Training Settings

All settings related to training and model creation is setup here.

[ ]:
```
# Saving/Loading
save_name = "models/deep_star_v2" # Save name for the savefiles, _{epoch} will␣
 ↪be appended to the save name.
load = False # Whether to load the save_name at startup.
save = True # Wether to auto save for each epoch.

#Tensor device configuration
use_gpu = True # Wether to run the model on the gpu.
```

```
gpu_device = 0 # What gpu to run the model on.
seed = 0 # GPU prng seed to ensure reproduceability

# Training Configuration
n_epochs = 20000 # How many epochs to run this program for.
validate_network = True # Wether to run a validation batch after each epoch.

# Data Loading
data_folder = "data" # Folder where data can be loaded fra.
num_workers = 1 # Number of gpu workers to use for data loading.
batch_size = 10 # How much data to train the model at once.
shuffle_data = True # Wether to randomize the data order.

# Custom properties
paths_per_image = 100 # How many paths there is per heightmap.
do_pool = False

#Critation
lr=1
rho=0.9
eps=1e-06
weight_decay=0

#Loss
weight=None
reduction='mean'
```

2.1 Model setup

Here the model is created and pretrained weights are loaded. The model is also moved to the GPU here if that setting is enableled.

[ ]:
```
net = DeepStar(do_pool)

if (load):
    net.load_state_dict(torch.load(f'{save_name}.pt'))

if (use_gpu):
    torch.cuda.empty_cache()

    device = torch.device(f'cuda:{gpu_device}' if torch.cuda.is_available()␣
 ↪else "cpu")
    torch.cuda.set_device(device)
    net.to(device)
```

3 Train Confguration

These cells specifies how the data from the custom data loader will be put into the network, and what the network is expected to output. This is used as an intermidiate layer to glue together the dataloader and the network model. This is also the only part of this notebook that is specific for each implementation, the example shown here is DeepStar V1.

[ ]:
```
# Setup training data from training data folder
train_data = PathDataLoader(f'{data_folder}/images/')
train_loader = utils.DataLoader(train_data, batch_size=batch_size,␣
↪shuffle=shuffle_data, num_workers=num_workers)

# If validation data has been enabled setup validation loader from validation␣
↪data folder.
if validate_network:
    val_data = PathDataLoader(f'{data_folder}/validation/')
    val_loader = utils.DataLoader(val_data, batch_size=batch_size,␣
↪shuffle=shuffle_data, num_workers=num_workers)
```

[ ]:
```
loss = net.get_loss()
optimizer = net.get_optim(rho, lr, weight_decay)

# Glue together the data and the network
def execute(net, data):
    img, points, e = data
    return net(img, points), e
```

4 Network Training

4.1 Training Setup

This is the method that runs one batch of training, meaning it iterates through all the data in the given data loader and trains the network on it. It executes the network via the execute command setup in the train configuration cell.

Run Batch

Data Loader: The data loader to use for this batch.

Net: The model to use for this batch. As specified in Model Setup.

Optim: The critation algorithm to use to modify the network weights.

Loss: The loss function to use for error calculation.

Use GPU: Wether to use GPU for thi batch.

Do Print: If you want the method to print the progress throughout the run.

```
[ ]: def run_batch(data_loader, net, optim, loss, use_gpu, do_print=False):
         running_loss = 0.0
         print_every = n_train_batches // 10
         start_time = time.time()
         total_train_loss = 0

         for i, data in enumerate(data_loader):
             # Load data into the GPU if it is enabeled
             if (use_gpu):
                 for a,d in enumerate(data):
                     data[a] = data[a].to(device)

             # Reset optim algorith, if this is not done the weights will be summed␣
         ↪and become to big
             optim.zero_grad()
             output, expected = execute(net, data)

             # Calculate the loss
             loss_size = loss(output, expected)

             # Run backpropergation algorithm as defined in model
             loss_size.backward()
             optim.step()

             # Calculate and print statisitcs
             running_loss += loss_size.data.item()
             total_train_loss += loss_size.data.item()

             if do_print and (i + 1) % (print_every + 1) == 0:
                 print(f'Epoch {epoch+1}, {int(100 * (i+1) / n_train_batches):d}% \t␣
         ↪train_loss: {(running_loss / print_every):.4f} took: {(time.time() -␣
         ↪start_time):.2f}s')

                 running_loss = 0.0
                 start_time = time.time()

         return total_train_loss
```

4.2 Training Loop

The main loop that runs the notebook and trains the model. It will run for the set amount of epochs and save once a epoch. This cell also deals with visualizing the performance, currently there is no graph visualisation only simple print statements. For future work it might be worth investing time into creating a proper graph.

[ ]:
```
# Calculate the amount of training an validation batches
n_train_batches = len(train_loader)
if validate_network:
    n_val_batches = len(val_loader)
else:
    n_val_batches = 0

training_start_time = time.time()
print(f'Training with {n_train_batches} training batches and {n_val_batches}␣
 ↪validation batches.')
for epoch in range(n_epochs):
    train_loss = run_batch(train_loader, net, optimizer, loss, use_gpu,␣
 ↪do_print=True)

    # Run a validation batch if its enabeled
    if validate_network:
        val_loss = run_batch(val_loader, net, optimizer, loss, use_gpu)
        print(f'Validation loss = {(val_loss / len(val_loader)):.4f}')

    # Save once a epoch
    if save:
        torch.save(net.state_dict(), f'{save_name}.pt')

print(f'Training finished, took {(time.time() - training_start_time):.2f}s')
```

4.3 Save

This cell is just so you can manually save if you want to.

[ ]:
```
if save:
    torch.save(net.state_dict(), f'{save_name}_0.pt')
    print("Saved")
```

Normal Q-Learning-Copy1

May 25, 2020

Author: Vetle A. H. Neumann

1 DQN Agent

Code taken from PyTorch DQN tutorial, then modified to work on our environment.

Source of original code: https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html

[ ]:
```
import gym
import gym_drone
import math
import random
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from collections import namedtuple
from itertools import count
from PIL import Image

import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import torchvision.transforms as T

# set up matplotlib
is_ipython = 'inline' in matplotlib.get_backend()
if is_ipython:
    from IPython import display

plt.ion()

# if gpu is to be used
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
torch.cuda.empty_cache()
#device = "cpu"
```

Intialize environment along with the matplotlib figure. - DroneCardinal-v0 is the name of the environment to initialize - Rows and columns determine the shape of the environment area - Memory capacity says how many of the last steps shall be remembered and displayed when rendering with 'notebook' mode. - Ax takes a matplotlib plot, which it will plot to if given, otherwise plot to the default pyplot. This makes it so we can have the rendering of the playing as a matplotlib subplot.

[ ]:
```
fig, ax = plt.subplots(nrows=2, ncols=2)
fig.set_figwidth(30)
fig.set_figheight(14)

grid_shape = (40, 40)
env = gym.make('DroneCardinal-v0',
               rows=grid_shape[0],
               columns=grid_shape[1],
               memory_capacity=50,
               ax=ax[1][0]).unwrapped
env.reset()
```

[ ]:
```
import os, sys

module_path = os.path.abspath(os.path.join('../..'))
if module_path not in sys.path:
    sys.path.append(module_path)

import laps, redis, loader
from Dijkstra.Dijkstra import *

grid = env._grid

grid = []
width, height = grid_shape
for x in range(width):
    grid.append([])
    for y in range(height):
        grid[x].append(Node(env._grid[x, y], (x, y)))

edges = get_edges(grid)
```

Set up the ReplayMemory object type.

It is how transitions in the environment is stored, and randomly given back with the sample function, to train the neural network.

[ ]:
```
resize = T.Compose([T.ToPILImage(),
                    #T.Resize(40, interpolation=Image.CUBIC),
                    #T.Resize(40),
                    T.Resize(40),
                    T.ToTensor()])
```

```
pos_blur_size = 10
goal_blur_size = 10
white_channel = np.zeros(grid_shape)

screen = env.render(mode='rgb_array')
last_drone_pos = tuple(env._get_obs()[:2])
last_goal_pos = tuple(env._get_obs()[2:])
def get_screen(new_episode=False):
    global screen, last_drone_pos, last_goal_pos
    drone_pos = tuple(env._get_obs()[:2])
    goal_pos = tuple(env._get_obs()[2:])

    if new_episode:
        screen = env.render(mode='rgb_array')
        screen[:, :, 0] = white_channel
        deblur_position(last_goal_pos, goal_blur_size, 2)
        blur_position(goal_pos, goal_blur_size, 2)
        last_goal_pos = goal_pos

    deblur_position(last_drone_pos, pos_blur_size, 1)
    blur_position(drone_pos, pos_blur_size, 1)

    last_drone_pos = drone_pos
    last_goal_pos = goal_pos

    # blur grid
    #screen = blur_grid(screen)
    # recenter around drone
    #screen = recenter_grid_on_position(screen)
    # set the red channel (height) to zero

    output = screen.transpose((2, 0, 1))
    _, screen_height, screen_width = output.shape
    output = np.ascontiguousarray(screen, dtype=np.float32) / 255
    output = torch.from_numpy(output)
    output = output.type(torch.FloatTensor)
    return resize(screen).unsqueeze(0).to(device)

def blur_position(position, blur_radius, channel):
    global screen
    size_x, size_y = screen[:, :, channel].shape
    for d_x in range(blur_radius * 2 + 1):
        d_x -= blur_radius
        for d_y in range(blur_radius * 2 + 1):
            d_y -= blur_radius
            # If the distance to the point is greater than the
```

```
            # distance to be blurred, skip this cell.
            distance = abs(d_x) + abs(d_y)
            if distance > blur_radius:
                continue
            x, y = position
            x += d_x
            y += d_y
            # Check if index is inside grid
            if x < 0 or x >= size_x or y < 0 or y >= size_y:
                continue
            screen[x, y, channel] = 255 / (distance + 1)

def deblur_position(position, blur_radius, channel):
    global screen
    size_x, size_y = screen[:, :, channel].shape
    for d_x in range(blur_radius * 2 + 1):
        d_x -= blur_radius
        for d_y in range(blur_radius * 2 + 1):
            d_y -= blur_radius
            x, y = position
            x += d_x
            y += d_y
            # Check if index is inside grid
            if x < 0 or x >= size_x or y < 0 or y >= size_y:
                continue
            screen[x, y, channel] = 0

def recenter_grid_on_position(rgb_array):
    rows, columns = grid_shape
    drone_pos = env._drone_pos
    drone_x, drone_y = drone_pos
    offset_x = (columns - 1) - drone_x
    offset_y = (rows - 1) - drone_y

    new_grid = np.zeros((rows * 2 - 1, columns * 2 - 1, 3)) + 255
    for x, column in enumerate(rgb_array):
        for y, row in enumerate(column):
            for channel, cell in enumerate(row):
                #new_grid[x + offset_x, y + offset_y, channel] = #rgb_array[x,␣
 ↪y, channel]
                new_grid[x + offset_x, y + offset_y, channel] = cell
    return new_grid
```

[ ]:
```
plt.imshow(np.swapaxes(np.swapaxes(np.squeeze(np.asarray(get_screen().cpu()),␣
 ↪0), 0, 2), 0, 1))
plt.axis('off')
plt.show()
```

[ ]:
```
env.step(1)
plt.imshow(env.render(mode='rgb_array'))
plt.show()
```

[ ]:
```
BATCH_SIZE = 128
GAMMA = 0.9
#GAMMA = 0.1
EPS_START = 0.9
EPS_END = 0.2
#EPS_END = 0.0
EPS_DECAY = 40000
TARGET_UPDATE = 10

# Get screen size so that we can initialize layers correctly based on shape
# returned from AI gym. Typical dimensions at this point are close to 3x40x90
# which is the result of a clamped and down-scaled render buffer in get_screen()
init_screen = env.render(mode='rgb_array')
screen_height, screen_width, _ = init_screen.shape

# Get number of actions from gym action space
n_actions = env.action_space.n

#policy_net = DQN(screen_height, screen_width, n_actions).to(device)
#target_net = DQN(screen_height, screen_width, n_actions).to(device)
#target_net.load_state_dict(policy_net.state_dict())
#target_net.eval()

#optimizer = optim.RMSprop(policy_net.parameters())
#memory = ReplayMemory(10000)

# Reset these whenever
episode_durations = []
episode_rewards = []
episode_best_reward = []
```

[ ]:
```
steps_done = 0
def select_action(state):
    global steps_done, q_table
    sample = random.random()
    eps_threshold = EPS_END + (EPS_START - EPS_END) * \
        math.exp(-1. * steps_done / EPS_DECAY)
    steps_done += 1
    if sample > eps_threshold:
        state = get_state()
        return np.argmax(q_table[state])
    else:
        return random.randrange(n_actions)
```

[ ]:
```
episode_durations = []
episode_rewards = []
episode_best_reward = []
def plot_durations():
    # Plot episode durations
    ax[0][0].cla()
    durations_t = torch.tensor(episode_durations, dtype=torch.float)
    ax[0][0].set_xlabel('Episode')
    ax[0][0].set_ylabel('Duration')
    ax[0][0].plot(durations_t.numpy())
    # Take 100 episode averages and plot them too
    if len(durations_t) >= 100:
        means = durations_t.unfold(0, 100, 1).mean(1).view(-1)
        means = torch.cat((torch.zeros(99), means))
        ax[0][0].plot(means.numpy())

    # Plot episode rewards
    ax[0][1].cla()
    rewards_t = torch.tensor(episode_rewards, dtype=torch.float)
    best_rewards_t = torch.tensor(episode_best_reward, dtype=torch.float)
    ax[0][1].set_xlabel('Episode')
    ax[0][1].set_ylabel('Reward')
    ax[0][1].plot(rewards_t.numpy())
    ax[0][1].plot(best_rewards_t.numpy())
    if len(rewards_t) >= 100:
        means = rewards_t.unfold(0, 100, 1).mean(1).view(-1)
        means = torch.cat((torch.zeros(99), means))
        ax[0][1].plot(means.numpy())

    # Plot environment
    env.render()

    # Plot environment as seen by the agent
    ax[1][1].imshow(np.swapaxes(np.swapaxes(np.squeeze(np.asarray(get_screen().
↪cpu()), 0), 0, 2), 0, 1))
    ax[1][1].axis('off')

    # pause a bit so that plots are updated
    plt.pause(0.001)
    if is_ipython:
        display.clear_output(wait=True)
        display.display(fig)
```

2 Q-Learning Functions

To reduce amount of possible states, use only 4 different start and stop positions.

[ ]:
```
possible_start_points = [(5,5), (grid_shape[0] - 5, 5), (5, grid_shape[1] - 5),␣
↪(grid_shape[0] - 5, grid_shape[1] - 5)]
LEARNING_RATE = 0.25

def encode(drone_x, drone_y, goal_index):
    state = drone_x
    state *= grid_shape[0]
    state += drone_y
    state *= grid_shape[1]
    state += goal_index
    return state

def decode(state):
    out = []
    out.append(state % grid_shape[1])
    state = state // grid_shape[1]
    out.append(state % grid_shape[0])
    state = state // grid_shape[0]
    out.append(state)
    return list(reversed(out))

def get_state():
    state = env._get_obs()
    goal_index = possible_start_points.index(tuple(state[2:]))
    return encode(state[0], state[1], goal_index)

q_table = np.zeros((encode(grid_shape[0] - 1, grid_shape[1] - 1, 3) + 1, 4)) + 1
```

3 Training Loop

Reward Function: 1 for correct direction, 0 otherwise.

[ ]:
```
num_episodes = 20000
episode_durations = []
episode_rewards = []
episode_best_reward = []

possible_start_points = [(5,5), (grid_shape[0] - 5, 5), (5, grid_shape[1] - 5),␣
↪(grid_shape[0] - 5, grid_shape[1] - 5)]
for i_episode in range(num_episodes):
    # Initialize the environment and state
    env.reset()

    #env._goal_pos = (5, 5)
    env._goal_pos = random.choice(possible_start_points)
```

```
    mistakes = 0

    state = get_state()
    last_state = state
    total_reward = 0
    highest_grid_value = np.amax(env._grid)
    for t in count():
        last_x, last_y, _, _ = env._get_obs()

        # Select and perform an action
        action = select_action(state)
        state_tuple, reward, done, _ = env.step(action)
        drone_x, drone_y, goal_x, goal_y = state_tuple
        drone_point = (drone_x, drone_y)
        goal_point = (goal_x, goal_y)

        # Hijack reward calculation (temporary?)
        if (drone_x, drone_y) == (goal_x, goal_y):
            reward = np.float64(0)
        else:
            diff = abs(goal_x - drone_x) + abs(goal_y - drone_y)
            last_diff = abs(goal_x - last_x) + abs(goal_y - last_y)
            if (last_diff > diff):
                reward = np.float64(1)
            else:
                reward = np.float64(0)

        total_reward += reward
        if t == 0:
            episode_best_reward.append(abs(last_x - goal_x) + abs(last_y -␣
↪goal_y))
        if t > 200:
            done = True
        if reward == 0:
            #done = True
            mistakes += 1
            if mistakes >= 10:
                done = True
            pass

        # Perform one step of the optimization
        state = get_state()
        # Adjust using the Q-Learning algorithm
        q_table[last_state][action] += LEARNING_RATE * (reward + GAMMA * np.
↪amax(q_table[state]) - q_table[last_state][action])

        last_state = state
```

```
        #if done or mistakes > 10:
        if done:
            #print(t + 1)
            episode_durations.append(t + 1)
            # episode_rewards.append(max(total_reward - mistakes, 0))
            episode_rewards.append(max(total_reward - mistakes, 0) /␣
 ↪episode_best_reward[-1])
            episode_best_reward[-1] = 1
            if i_episode % 500 == 0:
                plot_durations()
            break
print('Complete')
```

turn_short_v2

May 25, 2020

Author: Vetle A. H. Neumann

1 DQN Agent

Code taken from PyTorch DQN tutorial, then modified to work on our environment.

Source of original code: https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html

[ ]:
```
import gym
import gym_drone
import math
import random
import time
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from collections import namedtuple
from itertools import count
from PIL import Image

import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import torchvision.transforms as T

# set up matplotlib
is_ipython = 'inline' in matplotlib.get_backend()
if is_ipython:
    from IPython import display

plt.ion()

# if gpu is to be used
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
torch.cuda.empty_cache()
```

```
#device = "cpu"
```

2 Intialize Environment and Matplotlib Figure

- TurnShort-v1 is the name of the environment to initialize
- training_data_dir (required) specificies location of training data
- shape determine the shape of the environment area
- subsampling is how many chunks the grid should be divided into, in both directions
- steps is how many turns
- turn rate is the turn radius of the drone
- ax takes a matplotlib plot, which it will plot to if given, otherwise plot to the default pyplot. This makes it so we can have the rendering of the playing as a matplotlib subplot.

[ ]:
```
fig, ax = plt.subplots(nrows=2, ncols=2)
fig.set_figwidth(30)
fig.set_figheight(14)

grid_shape = (31, 31)
env = gym.make('TurnShort-v1',
               shape=grid_shape,
               training_samples=0,
               subsampling=31,
               steps=1,
               turn_rate=4,
               training_data_dir='/workspace/unsorted/',
               ax=ax[1][0]).unwrapped
env.reset()
```

3 Set up the ReplayMemory object type

It is how transitions in the environment is stored, and randomly given back with the sample function, to train the neural network.

[ ]:
```
Transition = namedtuple('Transition',
                        ('state', 'action', 'next_state', 'reward'))


class ReplayMemory(object):

    def __init__(self, capacity):
        self.capacity = capacity
        self.memory = []
        self.position = 0
```

```
    def push(self, *args):
        """Saves a transition."""
        if len(self.memory) < self.capacity:
            self.memory.append(None)
        self.memory[self.position] = Transition(*args)
        self.position = (self.position + 1) % self.capacity

    def sample(self, batch_size):
        return random.sample(self.memory, batch_size)

    def __len__(self):
        return len(self.memory)
```

4 Network

The network is a convolutional neural network, which uses ReLu activation function.

[ ]:
```
class DQN(nn.Module):

    def __init__(self, h, w, outputs):
        super(DQN, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=5, stride=1)
        self.bn1 = nn.BatchNorm2d(16)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=2, stride=1)
        self.bn2 = nn.BatchNorm2d(32)
        self.conv3 = nn.Conv2d(32, 32, kernel_size=2, stride=1)
        self.bn3 = nn.BatchNorm2d(32)

        # Number of Linear input connections depends on output of conv2d layers
        # and therefore the input image size, so compute it.
        #def conv2d_size_out(size, kernel_size = 5, stride = 2):
        #    return (size - (kernel_size - 1) - 1) // stride  + 1
        #convw = conv2d_size_out(conv2d_size_out(conv2d_size_out(w)))
        #convh = conv2d_size_out(conv2d_size_out(conv2d_size_out(h)))
        #linear_input_size = convw * convh * 32
        #self.head = nn.Linear(linear_input_size, outputs)
        self.head = nn.Linear(25 ** 2 * 32, outputs)

    # Called with either one element to determine next action, or a batch
    # during optimization. Returns tensor([[left0exp,right0exp]...]).
    def forward(self, x):
        x = F.relu(self.bn1(self.conv1(x)))
        x = F.relu(self.bn2(self.conv2(x)))
        x = F.relu(self.bn3(self.conv3(x)))
        return self.head(x.view(x.size(0), -1))
```

5 Get Image of Screen

Resize the image to fit neural network, change order of channels to fit the expected PyTorch input type, and send image to device.

[ ]:
```
resize = T.Compose([T.ToPILImage(),
                    T.Resize(31),
                    T.ToTensor()])

pos_blur_size = 10
goal_blur_size = 10

# create screen array, rows*columns*channels in size
# channels are RGB
screen = np.zeros((grid_shape[0], grid_shape[1], 3), dtype=np.uint8)
def get_screen(new_episode=False, points=None):
    global screen
    state = env._get_obs()
    last_pos = state[:2]
    drone_pos = state[2:4]
    goal_pos = state[4:]
    # these are only done once
    if new_episode:
        screen[:, :, 0] = env._heightmap
        screen[:, :, 1:] = np.zeros((grid_shape[0], grid_shape[1], 2), dtype=np.
 ↪uint8)

        # Scale red channel to be approximately 0 to 255
        max_value = np.amax(env._heightmap)
        min_value = np.amin(env._heightmap)

        # output[:, :, 0] /= max_value
        # output[:, :, 0] *= 255

        # Linear scale, in which every number is moved down
        # by the lowest, so that lowest point lies at 0, and
        # every point is linearly scaled by the factor needed
        # to move the moved highest point to 255. The result is
        # lowest point is at 0 and highest is at 255, and every
        # other is somewhere inbetween
        def scale_linear(highest, lowest, number):
            return (number - lowest) * (255/(highest - lowest))

        for y, row in enumerate(screen[:, :, 0]):
            for x, cell in enumerate(row):
                screen[y, x, 0] = scale_linear(max_value, min_value, cell)
```

```
        # blur goal position
        blur_position(goal_pos, goal_blur_size, screen[:, :, 2])

    # make copy of screen, by value, to modify for current frame
    current_screen = screen[:, :, :]
    if last_pos != (None, None):
        # blur last position
        blur_position(last_pos, pos_blur_size, current_screen[:, :, 1])

        current_screen[env._drone_pos[0], env._drone_pos[1], 1] = 255

    #blur_position(drone_pos, pos_blur_size, current_screen[:, :, 1])
    #blur_position(drone_pos, pos_blur_size, current_screen[:, :, 2])
    if points is not None:
        current_screen[:, :, 1] = np.zeros((grid_shape[0], grid_shape[1]),␣
↪dtype=np.uint8)
        for point in points:
            current_screen[point[0], point[1]] = 255
        pass

    output = current_screen.transpose((2, 0, 1))
    _, screen_height, screen_width = output.shape
    output = np.ascontiguousarray(current_screen, dtype=np.float32) / 255
    output = torch.from_numpy(output)
    output = output.type(torch.FloatTensor)
    return resize(current_screen).unsqueeze(0).to(device)

def lattice_distance(a, b):
    a_x, a_y = a
    b_x, b_y = b
    return abs(a_x - b_x) + abs(a_y - b_y)

def blur_position(position, blur_radius, table):
    size_x, size_y = table.shape
    for d_x in range(blur_radius * 2 + 1):
        d_x -= blur_radius
        for d_y in range(blur_radius * 2 + 1):
            d_y -= blur_radius
            # If the distance to the point is greater than the
            # distance to be blurred, skip this cell.
            distance = abs(d_x) + abs(d_y)
            if distance > blur_radius:
                continue
            x, y = position
            x += d_x
            y += d_y
            # Check if index is inside grid
```

```
            if x < 0 or x >= size_x or y < 0 or y >= size_y:
                continue
            table[x][y] += 255 / (distance + 1)
```

6 For Testing the Get Screen Method

[ ]:
```
env.reset()
env.step(1)
print(env._get_obs())
test = get_screen(True)
plt.imshow(np.swapaxes(np.swapaxes(np.squeeze(np.asarray(test.cpu()), 0), 0,␣
↪2), 0, 1))
plt.axis('off')
plt.show()
```

7 Setup Networks

[ ]:
```
BATCH_SIZE = 128
GAMMA = 0.9
EPS_START = 0.9
EPS_END = 0.01
EPS_DECAY = 35000
TARGET_UPDATE = 10

# Get screen size so that we can initialize layers correctly based on shape
# returned from AI gym. Typical dimensions at this point are close to 3x40x90
# which is the result of a clamped and down-scaled render buffer in get_screen()
#init_screen = env.render(mode='rgb_array')
#screen_height, screen_width, _ = init_screen.shape
screen_height, screen_width = grid_shape

# Get number of actions from gym action space
n_actions = env.action_space.n

policy_net = DQN(screen_height, screen_width, n_actions).to(device)
target_net = DQN(screen_height, screen_width, n_actions).to(device)
target_net.load_state_dict(policy_net.state_dict())
target_net.eval()

optimizer = optim.RMSprop(policy_net.parameters())
memory = ReplayMemory(10000)

# Reset these whenever
```

```
episode_durations = []
episode_rewards = []
episode_best_reward = []
```

8 Select Action Based on Policy Network

[ ]:
```
steps_done = 0
def select_action(state):
    global steps_done
    sample = random.random()
    eps_threshold = EPS_END + (EPS_START - EPS_END) * \
        math.exp(-1. * steps_done / EPS_DECAY)
    steps_done += 1
    if sample > eps_threshold:
        with torch.no_grad():
            # t.max(1) will return largest column value of each row.
            # second column on max result is index of where max element was
            # found, so we pick action with the larger expected reward.
            state = get_screen()
            policy_net.eval()
            return policy_net(state).max(1)[1].view(1, 1)
    else:
        return torch.tensor([[random.randrange(n_actions)]], device=device,␣
 ↪dtype=torch.long)
```

9 Function for Plotting Graphs

Episode_durations is how long each episode lasted.

Episode_rewards is rewards for each episode.

Episode_best_rewards is the best reward for every episode, this way the rewards may be compared to what the theoretical best rewards could've been.

[ ]:
```
episode_durations = []
episode_rewards = []
episode_best_reward = []
def plot_durations():
    # Plot episode durations
    ax[0][0].cla()
    durations_t = torch.tensor(episode_durations, dtype=torch.float)
    ax[0][0].set_xlabel('Episode')
    ax[0][0].set_ylabel('Duration')
    ax[0][0].plot(durations_t.numpy())
    # Take 100 episode averages and plot them too
```

```
    if len(durations_t) >= 100:
        means = durations_t.unfold(0, 100, 1).mean(1).view(-1)
        means = torch.cat((torch.zeros(99), means))
        ax[0][0].plot(means.numpy())

    # Plot episode rewards
    ax[0][1].cla()
    rewards_t = torch.tensor(episode_rewards, dtype=torch.float)
    best_rewards_t = torch.tensor(episode_best_reward, dtype=torch.float)
    ax[0][1].set_xlabel('Episode')
    ax[0][1].set_ylabel('Reward')
    ax[0][1].plot(rewards_t.numpy())
    ax[0][1].plot(best_rewards_t.numpy())
    if len(rewards_t) >= 100:
        means = rewards_t.unfold(0, 100, 1).mean(1).view(-1)
        means = torch.cat((torch.zeros(99), means))
        ax[0][1].plot(means.numpy())

    # Plot environment
    ax[1][0].cla()
    env.render()

    # Plot environment as seen by the agent
    ax[1][1].cla()
    ax[1][1].imshow(np.swapaxes(np.swapaxes(np.squeeze(np.
↪asarray(get_screen(True).cpu()), 0), 0, 2), 0, 1))
    ax[1][1].axis('off')

    # pause a bit so that plots are updated
    plt.pause(0.001)
    if is_ipython:
        display.clear_output(wait=True)
        display.display(fig)
```

Algorithm for Adjusting Network

[ ]:
```
def optimize_model():
    if len(memory) < BATCH_SIZE:
        return
    transitions = memory.sample(BATCH_SIZE)
    # Transpose the batch (see https://stackoverflow.com/a/19343/3343043 for
    # detailed explanation). This converts batch-array of Transitions
    # to Transition of batch-arrays.
    batch = Transition(*zip(*transitions))

    # Compute a mask of non-final states and concatenate the batch elements
    # (a final state would've been the one after which simulation ended)
```

```
    #non_final_mask = torch.tensor(tuple(map(lambda s: s is not None,
    #                                      batch.next_state)), device=device,␣
↪dtype=torch.bool)
    #non_final_next_states = torch.cat([s for s in batch.next_state
    #                                   if s is not None])
    state_batch = torch.cat(batch.state)
    action_batch = torch.cat(batch.action)
    reward_batch = torch.cat(batch.reward)

    # Compute Q(s_t, a) - the model computes Q(s_t), then we select the
    # columns of actions taken. These are the actions which would've been taken
    # for each batch state according to policy_net
    state_action_values = policy_net(state_batch).gather(1, action_batch)

    # Compute V(s_{t+1}) for all next states.
    # Expected values of actions for non_final_next_states are computed based
    # on the "older" target_net; selecting their best reward with max(1)[0].
    # This is merged based on the mask, such that we'll have either the expected
    # state value or 0 in case the state was final.
    #next_state_values = torch.zeros(BATCH_SIZE, device=device)
    #next_state_values[non_final_mask] = target_net(non_final_next_states).
↪max(1)[0].detach()
    # Compute the expected Q values
    #expected_state_action_values = (next_state_values * GAMMA) + reward_batch
    expected_state_action_values = reward_batch

    # Compute Huber loss
    loss = F.smooth_l1_loss(state_action_values,
                            expected_state_action_values.unsqueeze(1))

    # Optimize the model
    optimizer.zero_grad()
    loss.backward()
    for param in policy_net.parameters():
        param.grad.data.clamp_(-1, 1)
    optimizer.step()
```

10 Training Loop

Reward Function: Negative of total height, and -50,000 (chosen because it is lower than most normally encountered values).

[ ]:
```
#so that learning can be stopped, and continiued where it left off
last_episode = 0
```

[ ]:
```
num_episodes = 2500000
EPS_DECAY = 50000
EPS_START = 0.9
EPS_END = 0.15

possible_start_points = [(5,5), (grid_shape[0] - 5, 5), (5, grid_shape[1] - 5),␣
↪(grid_shape[0] - 5, grid_shape[1] - 5)]
for i_episode in range(last_episode, num_episodes):
    last_episode = i_episode
    # Initialize the environment and state
    env.reset()

    current_screen = get_screen(True)
    state = current_screen
    total_reward = 0
    for t in count():
        # Select and perform an action
        action = select_action(state)
        state_tuple, reward, done, info = env.step(action.item())
        last_x, last_y, drone_x, drone_y, goal_x, goal_y = state_tuple
        last_point = (last_x, last_y)
        drone_point = (drone_x, drone_y)
        goal_point = (goal_x, goal_y)

        # Overwrite reward function
        if info['points_traversed'] == []:
            reward = -50000
        else:
            reward = -1 * info['total_height']
        reward = torch.tensor([reward], device=device).type(torch.float)

        # Observe new state
        last_screen = current_screen
        current_screen = get_screen(True)

        if not done:
            next_state = current_screen
        else:
            next_state = None

        # Store the transition in memory
        memory.push(state, action, next_state, reward)

        # Move to the next state
        state = next_state

        # Perform one step of the optimization (on the target network)
```

```
        optimize_model()
        if done:
            episode_rewards.append(reward)
            if i_episode % 200 == 0:
                plot_durations()
            break

    # Update the target network, copying all weights and biases in DQN
    if i_episode % TARGET_UPDATE == 0:
        target_net.load_state_dict(policy_net.state_dict())

print('Complete')
```

10.1 Save Model and Data

[ ]:
```
# convert from tensor to normal list
episode_rewards_list = [i.item() for i in episode_rewards]
episode_best_reward_list = [i.item() for i in episode_best_reward]

file_name = "no_pooling_three_layer"
with open(file_name + ".txt", "w") as output:
    output.write(str(episode_rewards_list))

with open(file_name + "_best"+ ".txt", "w") as output:
    output.write(str(episode_best_reward))

torch.save(policy_net.state_dict(), file_name + "_policy_net.pt")
torch.save(target_net.state_dict(), file_name + "_target_net.pt")
```

Lazy Redraw V2

May 25, 2020

Author: Vetle A. H. Neumann

1 DQN Agent

Code taken from PyTorch DQN tutorial, then modified to work on our environment.

Source of original code: https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html

[ ]:
```
import gym
import gym_drone
import math
import random
import time
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from collections import namedtuple
from itertools import count
from PIL import Image

import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import torchvision.transforms as T

# set up matplotlib
is_ipython = 'inline' in matplotlib.get_backend()
if is_ipython:
    from IPython import display

plt.ion()

# if gpu is to be used
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
torch.cuda.empty_cache()
```

```
#device = "cpu"
```

2 Intialize Environment and Matplotlib Figure

- DroneCardinal-v0 is the name of the environment to initialize
- Rows and columns determine the shape of the environment area
- Memory capacity says how many of the last steps shall be remembered and displayed when rendering with 'notebook' mode.
- Ax takes a matplotlib plot, which it will plot to if given, otherwise plot to the default pyplot. This makes it so we can have the rendering of the playing as a matplotlib subplot.

[ ]:
```
fig, ax = plt.subplots(nrows=2, ncols=2)
fig.set_figwidth(30)
fig.set_figheight(14)

grid_shape = (40, 40)
env = gym.make('DroneCardinal-v0',
               rows=grid_shape[0],
               columns=grid_shape[1],
               memory_capacity=50,
               ax=ax[1][0]).unwrapped
env.reset()
```

3 Load Dijkstra

Load Dijkstra and calculate the edges.

[ ]:
```
import os, sys

module_path = os.path.abspath(os.path.join('../..'))
if module_path not in sys.path:
    sys.path.append(module_path)

import laps, redis, loader
from Dijkstra.Dijkstra import *

grid = env._grid

grid = []
width, height = grid_shape
for x in range(width):
    grid.append([])
    for y in range(height):
        grid[x].append(Node(env._grid[x, y], (x, y)))
```

```
edges = get_edges(grid)
```

4 Set up the ReplayMemory object type

It is how transitions in the environment is stored, and randomly given back with the sample function, to train the neural network.

[ ]:
```
Transition = namedtuple('Transition',
                        ('state', 'action', 'next_state', 'reward'))


class ReplayMemory(object):

    def __init__(self, capacity):
        self.capacity = capacity
        self.memory = []
        self.position = 0

    def push(self, *args):
        """Saves a transition."""
        if len(self.memory) < self.capacity:
            self.memory.append(None)
        self.memory[self.position] = Transition(*args)
        self.position = (self.position + 1) % self.capacity

    def sample(self, batch_size):
        return random.sample(self.memory, batch_size)

    def __len__(self):
        return len(self.memory)
```

5 Network

The network is a convolutional neural network, which uses ReLu activation function.

[ ]:
```
class DQN(nn.Module):

    def __init__(self, h, w, outputs):
        super(DQN, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=5, stride=2)
        self.bn1 = nn.BatchNorm2d(16)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=5, stride=2)
        self.bn2 = nn.BatchNorm2d(32)
```

```
        self.conv3 = nn.Conv2d(32, 32, kernel_size=5, stride=2)
        self.bn3 = nn.BatchNorm2d(32)

        # Number of Linear input connections depends on output of conv2d layers
        # and therefore the input image size, so compute it.
        def conv2d_size_out(size, kernel_size = 5, stride = 2):
            return (size - (kernel_size - 1) - 1) // stride  + 1
        convw = conv2d_size_out(conv2d_size_out(conv2d_size_out(w)))
        convh = conv2d_size_out(conv2d_size_out(conv2d_size_out(h)))
        linear_input_size = convw * convh * 32
        self.head = nn.Linear(linear_input_size, outputs)

    # Called with either one element to determine next action, or a batch
    # during optimization. Returns tensor([[left0exp,right0exp]...]).
    def forward(self, x):
        x = F.relu(self.bn1(self.conv1(x)))
        x = F.relu(self.bn2(self.conv2(x)))
        x = F.relu(self.bn3(self.conv3(x)))
        return self.head(x.view(x.size(0), -1))
```

6 Lazy Redraw of Blur

Arrays are passed by reference, not value. Therefore pre-calculate the blur in every pixel and store it on the GPU, so that it does not have to be calculated again, and only the reference needs to be changed.

This speeds up rendering of the screen by approximately 25%.

[ ]:
```
def blur_position(grid, position, blur_radius):
    size_x, size_y = grid.shape
    for d_x in range(blur_radius * 2 + 1):
        d_x -= blur_radius
        for d_y in range(blur_radius * 2 + 1):
            d_y -= blur_radius
            # If the distance to the point is greater than the
            # distance to be blurred, skip this cell.
            distance = abs(d_x) + abs(d_y)
            if distance > blur_radius:
                continue
            x, y = position
            x += d_x
            y += d_y
            # Check if index is inside grid
            if x < 0 or x >= size_x or y < 0 or y >= size_y:
                continue
            grid[x, y] = 1 / (distance + 1)
```

```
blur_radius = 10
blur_arrays = np.zeros((grid_shape[0], grid_shape[1], grid_shape[0],␣
 ↪grid_shape[1]))
for y, array_row in enumerate(blur_arrays):
    for x, array in enumerate(array_row):
        blur_position(blur_arrays[y][x], (y, x), blur_radius)

blur_tensors = torch.tensor(blur_arrays, dtype=torch.float64, device=device)
```

7 Get Image of Screen

Resize the image to fit neural network, change order of channels to fit the expected PyTorch input type, and send image to device.

[ ]:
```
resize = T.Compose([T.ToPILImage(),
                    T.Resize(40),
                    T.ToTensor()])

screen = None
white_channel = np.zeros(grid_shape)
def get_screen(new_episode=False):
    global screen
    drone_pos = tuple(env._get_obs()[:2])
    goal_pos = tuple(env._get_obs()[2:])

    if new_episode:
        screen = env.render(mode='rgb_array')

        # ignore heightmap
        #screen[:, :, 0] = white_channel

        screen = screen.transpose((2, 0, 1))
        _, screen_height, screen_width = screen.shape
        screen = np.ascontiguousarray(screen, dtype=np.float32) / 255
        screen = torch.from_numpy(screen)
        screen = screen.type(torch.FloatTensor)
        screen = resize(screen).unsqueeze(0).to(device)

    # blur drone position
    screen[0][1] = blur_tensors[drone_pos[0], drone_pos[1]]
    # blur goal position
    screen[0][2] = blur_tensors[goal_pos[0], goal_pos[1]]

    return screen
```

8 For Testing the Get Screen Method

[ ]:
```
test = get_screen(True)
plt.imshow(np.swapaxes(np.swapaxes(np.squeeze(np.asarray(test.cpu()), 0), 0,␣
 ↪2), 0, 1))
plt.axis('off')
plt.show()
```

9 Setup Networks

[ ]:
```
BATCH_SIZE = 128
GAMMA = 0.9
#GAMMA = 0.1
EPS_START = 0.9
EPS_END = 0.01
#EPS_END = 0.0
EPS_DECAY = 20 * 4000
TARGET_UPDATE = 10

# Get screen size so that we can initialize layers correctly based on shape
# returned from AI gym. Typical dimensions at this point are close to 3x40x90
# which is the result of a clamped and down-scaled render buffer in get_screen()
init_screen = env.render(mode='rgb_array')
screen_height, screen_width, _ = init_screen.shape

# Get number of actions from gym action space
n_actions = env.action_space.n

policy_net = DQN(screen_height, screen_width, n_actions).to(device)
target_net = DQN(screen_height, screen_width, n_actions).to(device)
target_net.load_state_dict(policy_net.state_dict())
target_net.eval()

optimizer = optim.RMSprop(policy_net.parameters(), lr=0.1, momentum=0.9)
memory = ReplayMemory(10000)

# Reset these whenever
episode_durations = []
episode_rewards = []
episode_best_reward = []
```

10 Select Action Based on Policy Network

[ ]:
```
steps_done = 0
def select_action(state):
    global steps_done
    sample = random.random()
    eps_threshold = EPS_END + (EPS_START - EPS_END) * \
        math.exp(-1. * steps_done / EPS_DECAY)
    steps_done += 1
    if sample > eps_threshold:
        with torch.no_grad():
            # t.max(1) will return largest column value of each row.
            # second column on max result is index of where max element was
            # found, so we pick action with the larger expected reward.
            state = get_screen()
            policy_net.eval()
            return policy_net(state).max(1)[1].view(1, 1)
    else:
        return torch.tensor([[random.randrange(n_actions)]], device=device,␣
↪dtype=torch.long)
```

11 Function for Plotting Graphs

Episode_durations is how long each episode lasted.

Episode_rewards is rewards for each episode.

Episode_best_rewards is the best reward for every episode, this way the rewards may be compared to what the theoretical best rewards could've been.

[ ]:
```
episode_durations = []
episode_rewards = []
episode_best_reward = []
def plot_durations():
    # Plot episode durations
    ax[0][0].cla()
    durations_t = torch.tensor(episode_durations, dtype=torch.float)
    ax[0][0].set_xlabel('Episode')
    ax[0][0].set_ylabel('Duration')
    ax[0][0].plot(durations_t.numpy())
    # Take 100 episode averages and plot them too
    if len(durations_t) >= 100:
        means = durations_t.unfold(0, 100, 1).mean(1).view(-1)
        means = torch.cat((torch.zeros(99), means))
        ax[0][0].plot(means.numpy())
```

```
    # Plot episode rewards
    ax[0][1].cla()
    rewards_t = torch.tensor(episode_rewards, dtype=torch.float)
    best_rewards_t = torch.tensor(episode_best_reward, dtype=torch.float)
    ax[0][1].set_xlabel('Episode')
    ax[0][1].set_ylabel('Reward')
    ax[0][1].plot(rewards_t.numpy())
    ax[0][1].plot(best_rewards_t.numpy())
    if len(rewards_t) >= 100:
        means = rewards_t.unfold(0, 100, 1).mean(1).view(-1)
        means = torch.cat((torch.zeros(99), means))
        ax[0][1].plot(means.numpy())

    # Plot environment
    ax[1][0].cla()
    env.render()

    # Plot environment as seen by the agent
    ax[1][1].cla()
    ax[1][1].imshow(np.swapaxes(np.swapaxes(np.squeeze(np.asarray(get_screen().
↪cpu()), 0), 0, 2), 0, 1))
    ax[1][1].axis('off')

    # pause a bit so that plots are updated
    plt.pause(0.001)
    if is_ipython:
        display.clear_output(wait=True)
        display.display(fig)
```

Algorithm for Adjusting Network

```
[ ]: def optimize_model():
    if len(memory) < BATCH_SIZE:
        return
    transitions = memory.sample(BATCH_SIZE)
    # Transpose the batch (see https://stackoverflow.com/a/19343/3343043 for
    # detailed explanation). This converts batch-array of Transitions
    # to Transition of batch-arrays.
    batch = Transition(*zip(*transitions))

    # Compute a mask of non-final states and concatenate the batch elements
    # (a final state would've been the one after which simulation ended)
    non_final_mask = torch.tensor(tuple(map(lambda s: s is not None,
                                            batch.next_state)), device=device,␣
↪dtype=torch.bool)
    non_final_next_states = torch.cat([s for s in batch.next_state
                                       if s is not None])
```

```
    state_batch = torch.cat(batch.state)
    action_batch = torch.cat(batch.action)
    reward_batch = torch.cat(batch.reward)

    # Compute Q(s_t, a) - the model computes Q(s_t), then we select the
    # columns of actions taken. These are the actions which would've been taken
    # for each batch state according to policy_net
    state_action_values = policy_net(state_batch).gather(1, action_batch)

    # Compute V(s_{t+1}) for all next states.
    # Expected values of actions for non_final_next_states are computed based
    # on the "older" target_net; selecting their best reward with max(1)[0].
    # This is merged based on the mask, such that we'll have either the expected
    # state value or 0 in case the state was final.
    next_state_values = torch.zeros(BATCH_SIZE, device=device)
    next_state_values[non_final_mask] = target_net(non_final_next_states).
↪max(1)[0].detach()
    # Compute the expected Q values
    expected_state_action_values = (next_state_values * GAMMA) + reward_batch

    # Compute Huber loss
    loss = F.smooth_l1_loss(state_action_values,
                            expected_state_action_values.unsqueeze(1))

    # Optimize the model
    optimizer.zero_grad()
    loss.backward()
    for param in policy_net.parameters():
        param.grad.data.clamp_(-1, 1)
    optimizer.step()
```

12 Training Loop

Reward Function: 1 for correct direction, 0 otherwise.

12.1 Save Model and Data

[ ]:
```
num_episodes = 20000000
episode_durations = []
episode_rewards = []
episode_best_reward = []

def lattice_distance(a, b):
    a_x, a_y = a
```

```
    b_x, b_y = b
    return abs(a_x - b_x) + abs(a_y - b_y)

for i_episode in range(num_episodes):
    # Initialize the environment and state
    env.reset()
    mistakes = 0

    current_screen = get_screen(True)
    state = current_screen
    total_reward = 0
    highest_grid_value = np.amax(env._grid)

    for t in count():
        last_x, last_y, _, _ = env._get_obs()

        # Select and perform an action
        action = select_action(state)
        state_tuple, reward, done, _ = env.step(action.item())
        drone_x, drone_y, goal_x, goal_y = state_tuple
        drone_point = (drone_x, drone_y)
        goal_point = (goal_x, goal_y)

        # Hijack reward calculation
        if (drone_x, drone_y) == (goal_x, goal_y):
            reward = np.float64(0)
        else:
            diff = abs(goal_x - drone_x) + abs(goal_y - drone_y)
            last_diff = abs(goal_x - last_x) + abs(goal_y - last_y)
            if (last_diff > diff):
                reward = np.float64(1)
            else:
                reward = np.float64(0)

        total_reward += reward
        reward = torch.tensor([reward], device=device).type(torch.float)
        if t > 200 or lattice_distance(drone_point, goal_point) < 2:
            done = True
        if reward == 0:
            #done = True
            mistakes += 1
            if mistakes >= 10:
                done = True

        # Observe new state
        last_screen = current_screen
        current_screen = get_screen()
```

```
        if not done:
            next_state = current_screen
        else:
            next_state = None

        # Store the transition in memory
        memory.push(state, action, next_state, reward)

        # Move to the next state
        state = next_state

        # Perform one step of the optimization (on the target network)
        optimize_model()

        # Append max reward possible for graphing
        if t == 0:
            episode_best_reward.append(abs(last_x - goal_x) + abs(last_y -␣
↪goal_y))
        if done:
            # The amount of steps the episode took
            episode_durations.append(t + 1)
            # Total reward gained divided by maximum, to get value between 0␣
↪and 1 of performance
            episode_rewards.append(max(total_reward - mistakes, 0) /␣
↪episode_best_reward[-1])
            episode_best_reward[-1] = 1
            # Only plot graphs every hundreth episode for performance
            if i_episode % 100 == 0:
                plot_durations()
            break

    # Update the target network, copying all weights and biases in DQN
    if i_episode % TARGET_UPDATE == 0:
        target_net.load_state_dict(policy_net.state_dict())

print('Complete')
```

12.2 Save Model and Data

[ ]:
```
# convert from tensor to normal list
episode_rewards_list = [i.item() for i in episode_rewards]
episode_best_reward_list = [i.item() for i in episode_best_reward]

file_name = "drone_cardinal"
with open(file_name + ".txt", "w") as output:
    output.write(str(episode_rewards_list))
```

```
with open(file_name + "_best"+ ".txt", "w") as output:
    output.write(str(episode_best_reward))

torch.save(policy_net.state_dict(), file_name + "_policy_net.pt")
torch.save(target_net.state_dict(), file_name + "_target_net.pt")
```