Sensur av hovedoppgaver
Høgskolen i Buskerud
Avdeling for Teknologi

HØGSKOLEN
i Buskerud

Prosjektnummer: 2010-7
For studieåret: 2009/2010
Emnekode: SFHO-3200

Prosjektnavn
LocalHawk – UAV Bakkesystem
LocalHawk – UAV Ground system

Utført i samarbeid med: Kongsberg Defence System

Ekstern veileder: Jon Bernhard Høstmark

Sammendrag: Videreutvikle et bakkesystem til et ubemannet autonomt fly. Det omfattet å utvikle bakkestasjons software, å videreutvikle en utskyter for flyet og GPS nøyaktighets forbedringer.

Stikkord:
- Unmanned aerial vehicle
- Ground Station
- Launcher

Tilgjengelig: DELVIS – Kildekode ikke tilgjengelig

Prosjekt deltagere:

Navn
Jon Bjørnland
Erik Myklebust
Lars Hallvard Sannes
Lars Erik Røise

Dato: 10. Juni 2010

Zoran Dokic	Olaf Hallan Graven	Jon Bernhard Høstmark
Intern Veileder	Intern Sensor	Ekstern Sensor

HØGSKOLEN
i Buskerud

MAIN RAPPORT

LocalHawk

Jon Bjørnland

Erik Myklebust

Lars Erik Røise

Lars Hallvard Sannes

May 20th 2010

Preface

We are a project group from Buskerud University College carried out this project during the spring of 2010. Our group consisted of Jon Bjørnland, Erik Myklebust, Lars Hallvard Sannes and Lars Erik Røise. It was the final project during the third year of our bachelor degree.

We were all excited to take part of this project, as it was interesting in many ways. It focused on incorporating several branches of engineering making one unified system. This gave us insight into cross branch cooperation.

We would like to give our thanks to: Saggrenda bilopphøggeri, for donating two car door latches to our release mechanism, Vestfold University College for lending us a much needed JTAGICE, Paul Arthur Vittersø for lending us his workshop and giving our Lars Hallvard Sannes a crash course in welding and Per Maguns Veierland for RC1240 support.

Would also like to thank our supervisors: Emil Moholt, Arne Goderstad, Magnus Wannebo and Zoran Dokic for feedback, help and advice.

Last but not least, we would like to thank our supervisor Jon Bernhard Høstmark for advice, feedback, test help and some mean flying skills.

Summary

This LocalHawk project report treats development of LocalHawk ground system, primarily the development of the Ground Station software, position accuracy improvement and further development of the Launcher.

Goals accomplished during this project:

- A working Ground Station software application
- Telemetry transfer from the UAV[1] to the new Ground Station
- Implemented DGPS[2] and position filtering
- Siren airframe launch capability, with the use of a new wagon adapter
- Redesigned a durable release mechanism

Guides on how to set up and use the new hardware and software are included in the appendices.

The Ground Station software consists of a computer with a connected GPS and Radio Link and a running application that communicates with the aircraft. The application's task is to receive data from a radio link and a GPS, then displaying this information in a GUI[3].

The information from the radio link consists of telemetry from the aircraft. GPS and IMU[4] data is collected in the aircraft then transmitted down to a receiving unit and into the Ground Station.

We implemented DGPS to improve position accuracy and position filtering possibilities for more correct acquisition of the Ground Station's position to be used in the DGPS calculation.

The Siren airframe wagon adapter consists of two main parts forward and aft stabilizers both are adjustable and can be used for other front powered model aircrafts.

We chose to design a new durable servo operated release mechanism, which is capable of both delayed onsite switch and wireless release with added safety features.

[1] UAV - *Unmanned aerial vehicle*. See chapter *1.*
[2] DGPS - *Differential-GPS*. See chapter *2.3.1.*
[3] GUI – *Graphical User Interface*. See chapter 2.5.
[4] IMU – *Internal Measurement Unit.* See chapter *1.2.*

Conclusion

During this bachelor project the LocalHawk platform has been further developed to encompass; a reworked Ground Station software written in C++, GPS position improving features, a wagon adapter capable of launching the Siren aircraft and a new sturdy wireless release mechanism.

The Ground Station has been totally reworked. The new Ground Station is written in C++. It is divided into two sub programs, the Core and GUI, and utilizes a socket connection between them. The GUI is an interface that shows information collected from the UAV and the Core application that acts like a data relay, which forwards the COM port data to the GUI. Position improvement has been achieved by implementing a DGPS system for the UAV and a position filtering to acquire the Ground Station's stationary position for the DGPS.

The Siren aircraft can now be launched, with use of the new wagon adaptor. This can be fixed firmly to the existing wagon and adjusted to fit the Siren aircraft or other front powered model airplane. Launches can now be preformed wirelessly from the Ground Station; a radio circuit has been connected to the new release mechanism and is able to communicate with the Ground Station. The release mechanism is redesigned to be able to deal with the forces from Launch. It has also greatly improved safety feature, able to eliminate the risk of unintentional launch.

The development of microcontroller-based components became greater tasks than planned, since most of the previously written code would not compile properly and therefore was used primarily for orientation rather than rewritten to achieve our goals.

Contact information

The project group had some questions during the execution of this project but had some problems finding contact information and who worked on what. Therefore contact information is included:

Jon Bjørnland – Navigation and Ground Station serial communication and parsing
Bjornland@gmail.com

Erik Myklebust – Radio Link and release mechanism
erikmykl@gmail.com

Lars Hallvard Sannes – Wagon and release mechanism
larshsann@hotmail.com

Lars Erik Røise – Ground Station
roise88@gmail.com

Contents

Figure Index

Chapter 1

Introduction

Localhawk is the name of a project financed by the Kongsberg based company, Kongsberg Defence Systems referred to as KDS. The project has over the years been worked on by several projects to create one product. Localhawk has evolved throughout the last years from what is defined as the beginning in 2007

The goal of the LocalHawk is to create a fully functional Autonomous Unmanned Aerial Vehicle or AUAV. An AUAV is a remote control aircraft drone capable of autonomous take off, stable flight, navigation and landing

The LoclHawk platform started in 2007 with the Cyberswan master thesis done by Jon Bernhard Høstmark, Mikael Kristian Eriksen and Edgar Bjørntvedt at NTNU. This project resulted an aircraft platform capable of stable autonomous flight.

The LocalHawk project itself started the summer of 2008 with cooperative summer internship, which ended with a reworked development platform, internal electronics and sensors.

This project has since then culminated in a platform capable of autonomous take-off, navigation and landing as well as video recognition. A wind tunnel tested carbon fiber airframe has been developed and simulation models for the entire system have been created.

This project will surround the development of the ground system that is the ground station and launcher.

1.1 Goals

The problem statement of this project:

Improve the LocalHawk ground system to be more reliable and modular for later expansions, by creating a new version of the Ground Station software, improving the GPS and radio link reliability, making the Launcher wagon capable of taking other airframes, and creating a reliable launch release mechanism.

The ground system is divided into two sub systems, the Ground Station and the Launcher.

1.1.1 Ground Station

One of the overall goals to this system is that everything is kept fail safe and modular.

The ground station is divided into:

Core application

- Make an application in C++ that reads from the GPS and radio link module and relays that information to the GUI.

Graphical user interface

- Make a GUI that displays information sent from the Core[5].

Radio Frequency communication

- Implement electronics in the aircraft to gather GPS information.
- Use a long range radio link to send telemetric data down to a receiving unit that sends strings of data to the Ground Station.

Navigation

- Add a GPS module to the Ground Station.
- Improve the accuracy of the GPS position of the UAV.

1.1.2 Launcher

Make improvements to the launcher, giving it more possibilities and more usability.

The Launcher is divided into:

Wagon

- Make the wagon capable of launching Siren airframe (front mounted thrust).

Release mechanism

- Make a semi automatic release mechanism that works.
- Improve the safety regarding premature launch.

[5] Core – The communication layer between GPS, radio link and GUI. See chapter 2.4.

1.2 The used circuit boards

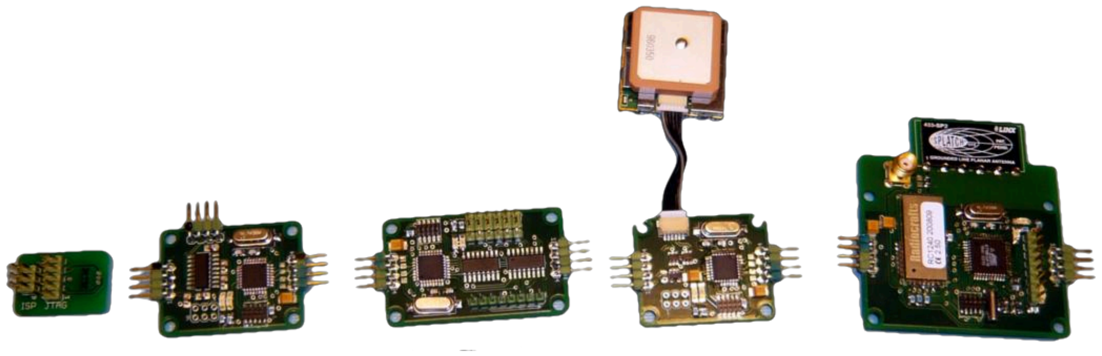


Figure 1: The circuit boards

In this project, circuit boards created by KDA in 2008 were used. There have been made configuration to these boards to suit our goals. This will be described into the document. For more information on these boards and their circuit diagrams, see (ref #).

The cards used in this project:

- Two radio circuit boards (RF boards) used for low frequency wireless communication.
- Two GPS circuit boards used to collect position data from GPS module
- One IMU circuit board created for collecting data on orientation and gravitational forces, but used in this project for its I^{2}C to UART since the all the cards only have one hardware UART. Software programmed UART turned out to be unstable
- One servo circuit board used for writing servo data
- Programming adaptor circuit to allow standard development tools to program the microcontrollers.

The boards are all built basically similar, with peculiar elements to perform their individual functions. They all contain a microcontroller, a JTAG[6] connector, and four pins on opposite ends. Pin 1 and 4 are power and ground respectably. Pin 2 and 3 are used for communication between the boards, using the I^{2}C protocol[7]. Having the I^{2}C pins connected directly to the opposite pins creates a common bus, making a board available for all connected boards, using addressing.

I^{2}C

I^{2}C (Two-Wire-Interface) is a standard for connecting different hardware in a quick and port-saving manner. Data is transferred on one wire and clocking is sent on the other. It is also important to have common ground between the hardware. The components are connected in parallel and can each have a unique address. I^{2}C uses a slave-master system meaning the master(s) request or sends information to the slave(s).

[6] JTAG - A connector to program the microcontroller. See Chapter 3.3.7.
[7] I^{2}C protocol See Chapter 3.3.7.

1.3 The aircrafts used

The main LocalHawk aircraft is a Stryker f27c. It has a wingspan of 94cm, length 69cm, weighs 620g. The other aircraft added in the LocalHawk project is the Siren. It has a wingspan of 200.7 cm, length 97.8 cm and weighs 1200g.

Figure 2: The Siren aircraft [1]

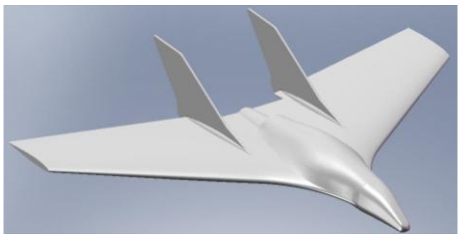


Figure 3: The LocalHawk Stryker aircraft [2]

2 Chapter 2

Ground Station

2.1 System

The Ground Station consists of a computer running an application that communicates with all the other parts of the system. The Ground Station is using an EM-406A GPS module and a GPS circuit board to support the GPS. It also uses a radio transceiver to collect all necessary data from the aircraft. The local GPS data together with data from the aircraft is merged and displayed in a GUI. The Ground Station also uses the transceiver to wirelessly control the Release Mechanism on the Launcher. All the communication between the used circuit boards is done using I^{2}C.

2.2 Communication with UAV

To begin with problems with the two development-kits were found. Attempting setting up a connection between the two development-kits was failed. To solve this problem; Radiocrafts' configuration tools was downloaded. Reaching the configuration options in the module still did not work. Quite some time was spent troubleshooting this problem, but without results. A new concept was created to overcome this barrier; using the one of the radio circuit boards as part of the Ground Station in an embedded platform.

Concept of the embedded platform

The embedded system was designed to contain the three circuit boards shown in figure 4. The IMU circuit board is used because the two other circuit boards are using their only hardware UART ports. This concept's motive was that moving the processes out of the core application and on to an embedded platform; the development of the cross-platform aspect of the core application is easier. Treating the data before entering the Core was thought to reduce processing time, but this was found to be false because of the microprocessor's low performance. This concept was then discarded.

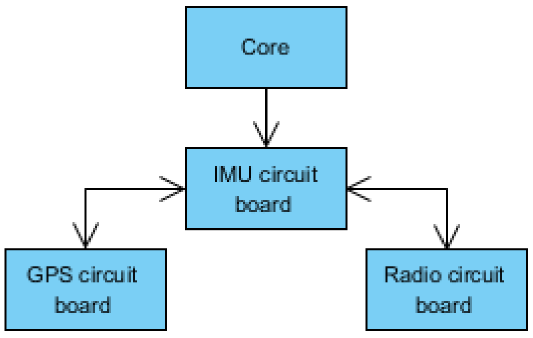


Figure 4: The embedded platform concept

Concept of using the Radiocrafts RC1240 development-kit

Discarding the last concept, the development-kits again had to come into play. The dev.kit was now thought to be the connection between the UAV and the Ground Station. It would receive data; convert this to UART before sending this over a virtual COM-port and into the Ground Station Core where the data would be further treated. The Ground Station's GPS data would be connected through the IMU board and into the Ground Station using a second COM-port. The fact that this would use two COM/USB-ports did not matter to KDA, this concept was chosen. The system design is shown figure 5.

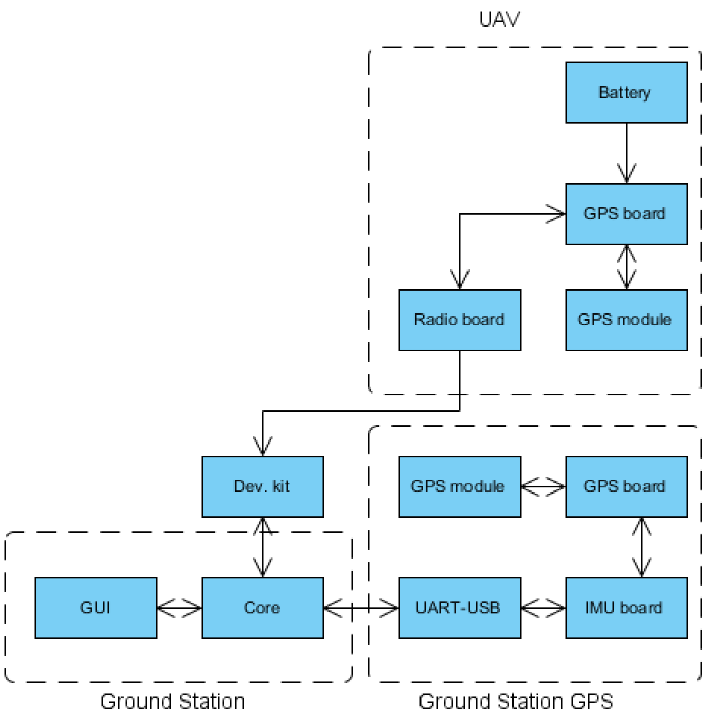
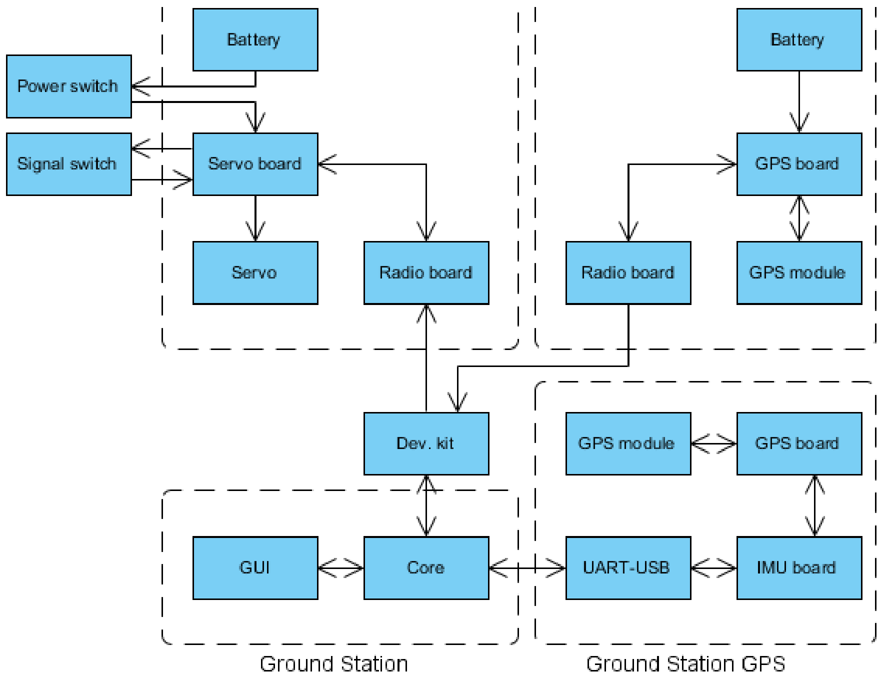


Figure 5: Development kit design

Figure 6: Entire system layout



Concept of remotely releasing the launcher's wagon

Having the launcher wirelessly connected to the Ground Station was would make a convenient feature, making the release mechanism available for the Ground Station. With this, a "Launch" button to trigger the release mechanism remotely could be made in the Ground Station.

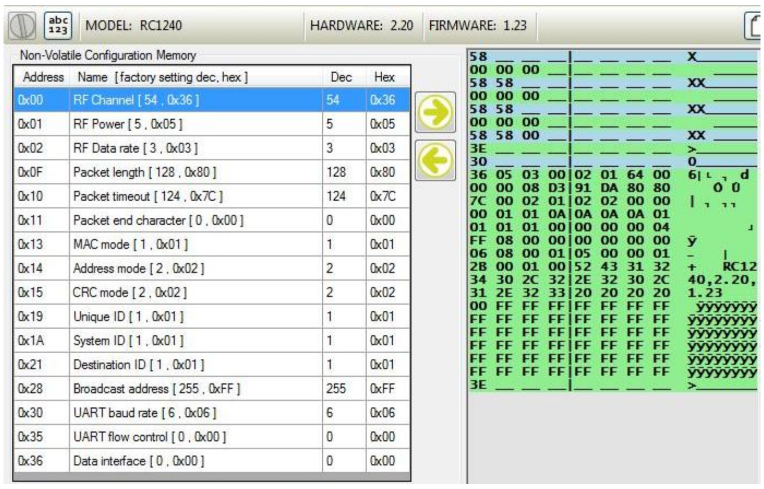
The embedded platform concept resulted in a shortage of radio boards, thus making launching the aircraft difficult. The radio board in the aircraft had to receive the "launch"-signal. From here, a wire was supposed to carry this signal to the servo board. See chapter 3.3.6 for more details on this subject.

After deciding to use the development-kit to be the link between aircraft and Ground Station, a radio board was free to use. Placing this in the Release Mechanism the task of making the ground station wireless suddenly took a great step closer to realization. Figure 6 shows the the final system design.

The radio board in the aircraft is constantly simulating collecting GPS data, sending it to the development-kit and into the Core, where data is treated. The Ground Station should have a function that sends a launch signal to the radio board in the Release Mechanism. For more information about the Release Mechanism, see chapter 3.3.5.

2.2.2 Problems while setting up the system

As mentioned earlier, problems were encountered using the Radiocrafts' development-kit. Since the last project group had used these development-kits, A former project member was contacted for help with troubleshooting. He agreed that setting up the development-kits should not be much of a problem, especially since Radiocrafts' Configuration and Communication Tool (CCT) was used. This still did not work. Radiocrafts' help desk was contacted. Luckily they had experienced this problem before.



MODEL: RC1240 HARDWARE: 2.20 FIRMWARE: 1.23

Non-Volatile Configuration Memory

Address	Name [factory setting dec, hex]	Dec	Hex
0x00	RF Channel [54 , 0x36]	54	0x36
0x01	RF Power [5 , 0x05]	5	0x05
0x02	RF Data rate [3 , 0x03]	3	0x03
0x0F	Packet length [128 , 0x80]	128	0x80
0x10	Packet timeout [124 , 0x7C]	124	0x7C
0x11	Packet end character [0 , 0x00]	0	0x00
0x13	MAC mode [1 , 0x01]	1	0x01
0x14	Address mode [2 , 0x02]	2	0x02
0x15	CRC mode [2 , 0x02]	2	0x02
0x19	Unique ID [1 , 0x01]	1	0x01
0x1A	System ID [1 , 0x01]	1	0x01
0x21	Destination ID [1 , 0x01]	1	0x01
0x28	Broadcast address [255 , 0xFF]	255	0xFF
0x30	UART baud rate [6 , 0x06]	6	0x06
0x35	UART flow control [0 , 0x00]	0	0x00
0x36	Data interface [0 , 0x00]	0	0x00

```
58 __ __ __|__ __ __ __  X_______
00 00 00 __|__ __ __ __   _______
58 58 __ __|__ __ __ __  XX______
00 00 00 __|__ __ __ __   _______
58 58 __ __|__ __ __ __  XX______
00 00 00 __|__ __ __ __   _______
58 58 00 __|__ __ __ __  XX _____
3E __ __ __|__ __ __ __  >_______
30 __ __ __|__ __ __ __  0_______
36 05 03 00|02 01 64 00  6|     d
00 00 08 D3|91 DA 80 80     Ó Ú
7C 00 02 01|02 02 00 00  |
00 01 01 0A|0A 0A 0A 01
01 01 01 00|00 00 00 04
FF 08 00 00|00 00 00 00  ÿ
06 08 00 01|05 00 00 01  -    |
2B 00 01 00|52 43 31 32  +   RC12
34 30 2C 32|2E 32 30 2C  40,2.20,
31 2E 32 33|20 20 20 20  1.23
00 FF FF FF|FF FF FF FF   ÿÿÿÿÿÿÿ
FF FF FF FF|FF FF FF FF  ÿÿÿÿÿÿÿÿ
FF FF FF FF|FF FF FF FF  ÿÿÿÿÿÿÿÿ
FF FF FF FF|FF FF FF FF  ÿÿÿÿÿÿÿÿ
FF FF FF FF|FF FF FF FF  ÿÿÿÿÿÿÿÿ
FF FF FF FF|FF FF FF FF  ÿÿÿÿÿÿÿÿ
3E __ __ __|__ __ __ __  >_______
```

Figure 7: The first memory reading of the development-kit using Radiocrafts' CCT

Apparently, the modules had been configured in internal non-volatile memory (EEPROM), and had become corrupt. This happens if one configures using the entering the EEPROM and exiting without first writing a stop bit. A list of default values had to be written into the module to fix the problem. See the readme-file on the CD. This fixed the problem. Figure 7 shows the memory values of the RC1240 after being fixed. All the values shown are used, except the Destination ID and Unique ID. These were set individually for each module.

To some extent it was tried to do the same with the second development-kit to make a connection work between the development-kits. This did not work and was discarded because of time.

2.2.3 Using the microcontrollers

The GPS board is described in ref 2.3.2 and will not be described here.

Aircraft's radio board

To begin with, Atmel's own UART communication code was used to send data to the RC1240 module. The problem using this code is that the GPS module receives large float and double variables. These are then sent to the radio board where it is formatted using the UART protocol. The UART code can only send single characters to the buffer before sending another.

```
for(int i=0;i<sizeof(data_to_send);i++)
        uart_putchar(data_to_send [i], stdout)
```

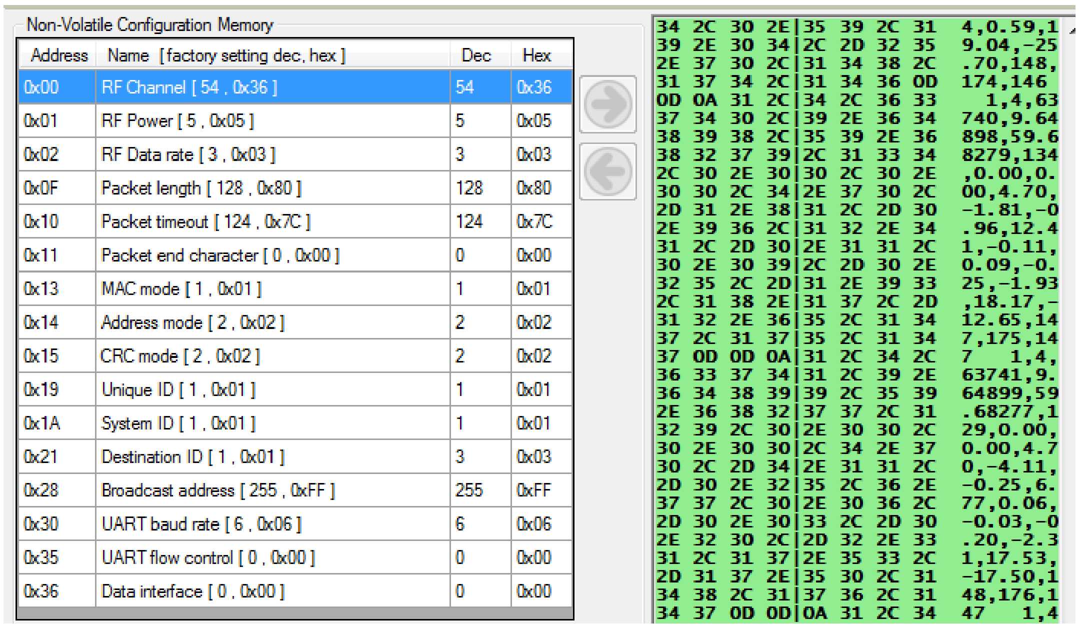
To make this code work, the GPS data had to be casted. The casting of these variables turned out to be more of a hindrance than expected. Instead, a simpler function, printf() was used as shown below. Beware, this still requires the "stdout = uart" line. This function converts all types of data types into a string which then is sent over the UART to the module.

```
printf("%d,",recieved->pos_fix);
printf("%d,",recieved->sat_used);
printf("%.0f,",recieved->utc_time);
printf("%.5f,",recieved->longitude);
(etc.)
```

The next step was initializing the RC1240 module on the radio board. This had be done on the previously using the code from the former project group to program the Atmega16. While trying to transmit data to the Ground Station from the board was clear that something was wrong. Scanning the radio signals in the RC1240's range (333.05-434.79) MHz showed no activity. The EEPROM was corrupt. The "fix-values" received from Radiocrafts were sent to the module as follows:

```
Memorytable[128];

for(i=0;sizeof(Memorytable);i++)

        sendchar(Memorytable[i]);
```

This fixed the problem and the module started transmitting. The configuration function is included in the code, but to avoid problems, this should never be run at every initializing. The development-kit's and the radio circuit's module's addresses were synchronized, and information was transmitted. Figure 8 shows this data using Radiocrafts' CCT. For more information on Ground Station data handling, see chapter 2.5.3.



Non-Volatile Configuration Memory

Address	Name [factory setting dec, hex]	Dec	Hex
0x00	RF Channel [54 , 0x36]	54	0x36
0x01	RF Power [5 , 0x05]	5	0x05
0x02	RF Data rate [3 , 0x03]	3	0x03
0x0F	Packet length [128 , 0x80]	128	0x80
0x10	Packet timeout [124 , 0x7C]	124	0x7C
0x11	Packet end character [0 , 0x00]	0	0x00
0x13	MAC mode [1 , 0x01]	1	0x01
0x14	Address mode [2 , 0x02]	2	0x02
0x15	CRC mode [2 , 0x02]	2	0x02
0x19	Unique ID [1 , 0x01]	1	0x01
0x1A	System ID [1 , 0x01]	1	0x01
0x21	Destination ID [1 , 0x01]	3	0x03
0x28	Broadcast address [255 , 0xFF]	255	0xFF
0x30	UART baud rate [6 , 0x06]	6	0x06
0x35	UART flow control [0 , 0x00]	0	0x00
0x36	Data interface [0 , 0x00]	0	0x00

```
34 2C 30 2E|35 39 2C 31  4,0.59,1
39 2E 30 34|2C 2D 32 35  9.04,-25
2E 37 30 2C|31 34 38 2C  .70,148,
31 37 34 2C|31 34 36 0D  174,146
0D 0A 31 2C|34 2C 36 33    1,4,63
37 34 30 2C|39 2E 36 34  740,9.64
38 39 38 2C|35 39 2E 36  898,59.6
38 32 37 39|2C 31 33 34  8279,134
2C 30 2E 30|30 2C 30 2E  ,0.00,0.
30 30 2C 34|2E 37 30 2C  00,4.70,
2D 31 2E 38|31 2C 2D 30  -1.81,-0
2E 39 36 2C|31 32 2E 34  .96,12.4
31 2C 2D 30|2E 31 31 2C  1,-0.11,
30 2E 30 39|2C 2D 30 2E  0.09,-0.
32 35 2C 2D|31 2E 39 33  25,-1.93
2C 31 38 2E|31 37 2C 2D  ,18.17,-
31 32 2E 36|35 2C 31 34  12.65,14
37 2C 31 37|35 2C 31 34  7,175,14
37 0D 0D 0A|31 2C 34 2C  7   1,4,
36 33 37 34|31 2C 39 2E  63741,9.
36 34 38 39|39 2C 35 39  64899,59
2E 36 38 32|37 37 2C 31  .68277,1
32 39 2C 30|2E 30 30 2C  29,0.00,
30 2E 30 30|2C 34 2E 37  0.00,4.7
30 2C 2D 34|2E 31 31 2C  0,-4.11,
2D 30 2E 32|35 2C 36 2E  -0.25,6.
37 37 2C 30|2E 30 36 2C  77,0.06,
2D 30 2E 30|33 2C 2D 30  -0.03,-0
2E 32 30 2C|2D 32 2E 33  .20,-2.3
31 2C 31 37|2E 35 33 2C  1,17.53,
2D 31 37 2E|35 30 2C 31  -17.50,1
34 38 2C 31|37 36 2C 31  48,176,1
34 37 0D 0D|0A 31 2C 34  47   1,4
```

Figure 8 Received telemetry data using CCT

UART data transmission

This explains in short the principle of using Universal asynchronous receiver/transmitter (UART) and what settings were used. UART is a piece of computer hardware that translates data between parallel and serial communication. UART breaks a byte down into five to eight bits carried by a start bit and a stop bit as shown in figure 9. An UART on the other side of the transmission line rebuilds the bits into one byte. One may also add a parity bit, but with this short low noise travel distance between components, a parity check was not needed.

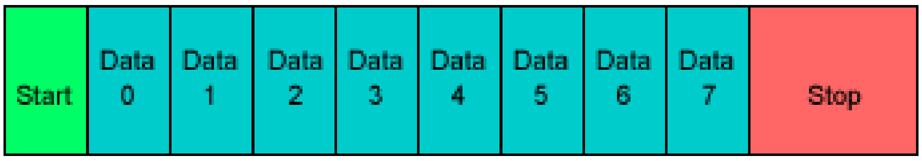


Figure 9: UART byte breakdown

Interfacing the RC 1240 modules is done through UART. To set up an UART connection, one has to specify the UART settings. The following shows the order to set the settings.

$$\text{(Baud rate, byte size, parity, stop bit, flow control)} \qquad (2.1)$$

The settings used to communicate with the RC1240 is (19200, 8, 1, N, no flow control). Up to 128 bytes can be stored in the module's buffer. The module will transmit data when the buffer is either full, the unique end character is sent, or the timeout limit is reached.

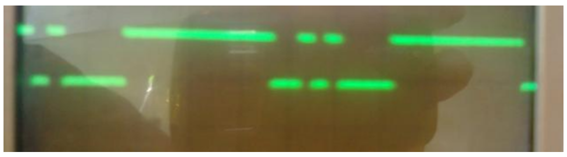


Figure 10: Oscilloscope showing an example on two bytes sent with UART protocol

2.3 Navigation

The navigation part of this assignment was to get GPS position data of the Ground Station and using this to improve the accuracy of the UAV's position. The first thing needed to do here is to get an idea of the amount of work that lies in the *parsing* or formatting of the GPS data.

Parsing is the process of analyzing a text string. Parsing splits a sequence of characters or values into smaller parts. It can be used for recognizing characters or values that occur in a specific order. In addition to this, it provides a powerful, readable, and maintainable approach to regular expression pattern matching.

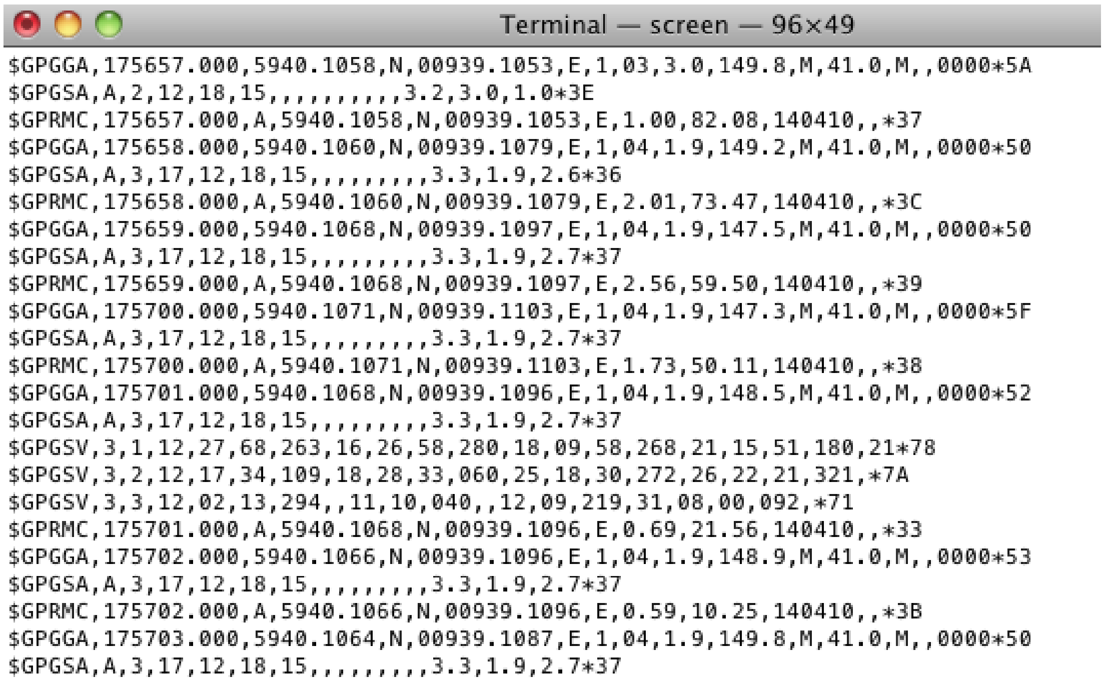
2.3.1 GPS analyses

What the raw GPS data looks like and what parts of the data we need was started. There were connected a FT232 converter, soldered on a break out board[8] to the GPS. The FT232 is a COM port redirector; a chip that converts UART signals to USB signals and the microchips driver emulate a COM port on the computer. Using this makes it available to a computer that does not have an RS232 serial port.

Serial Port Read

To read the data of the serial port on the computer it was used the UNIX terminal in OS x and hyper terminal on windows machines. Hyper terminal is a native app on win XP and earlier, but for vista and windows 7 it was developed a Serial Monitor application see chapter 2.7 (Available on the documentation CD).

When connected to a computer and had acquired satellite lock, gave the following data:



```
$GPGGA,175657.000,5940.1058,N,00939.1053,E,1,03,3.0,149.8,M,41.0,M,,0000*5A
$GPGSA,A,2,12,18,15,,,,,,,,,,3.2,3.0,1.0*3E
$GPRMC,175657.000,A,5940.1058,N,00939.1053,E,1.00,82.08,140410,,*37
$GPGGA,175658.000,5940.1060,N,00939.1079,E,1,04,1.9,149.2,M,41.0,M,,0000*50
$GPGSA,A,3,17,12,18,15,,,,,,,,,3.3,1.9,2.6*36
$GPRMC,175658.000,A,5940.1060,N,00939.1079,E,2.01,73.47,140410,,*3C
$GPGGA,175659.000,5940.1068,N,00939.1097,E,1,04,1.9,147.5,M,41.0,M,,0000*50
$GPGSA,A,3,17,12,18,15,,,,,,,,,3.3,1.9,2.7*37
$GPRMC,175659.000,A,5940.1068,N,00939.1097,E,2.56,59.50,140410,,*39
$GPGGA,175700.000,5940.1071,N,00939.1103,E,1,04,1.9,147.3,M,41.0,M,,0000*5F
$GPGSA,A,3,17,12,18,15,,,,,,,,,3.3,1.9,2.7*37
$GPRMC,175700.000,A,5940.1071,N,00939.1103,E,1.73,50.11,140410,,*38
$GPGGA,175701.000,5940.1068,N,00939.1096,E,1,04,1.9,148.5,M,41.0,M,,0000*52
$GPGSA,A,3,17,12,18,15,,,,,,,,,3.3,1.9,2.7*37
$GPGSV,3,1,12,27,68,263,16,26,58,280,18,09,58,268,21,15,51,180,21*78
$GPGSV,3,2,12,17,34,109,18,28,33,060,25,18,30,272,26,22,21,321,*7A
$GPGSV,3,3,12,02,13,294,,11,10,040,,12,09,219,31,08,00,092,*71
$GPRMC,175701.000,A,5940.1068,N,00939.1096,E,0.69,21.56,140410,,*33
$GPGGA,175702.000,5940.1066,N,00939.1096,E,1,04,1.9,148.9,M,41.0,M,,0000*53
$GPGSA,A,3,17,12,18,15,,,,,,,,,3.3,1.9,2.7*37
$GPRMC,175702.000,A,5940.1066,N,00939.1096,E,0.59,10.25,140410,,*3B
$GPGGA,175703.000,5940.1064,N,00939.1087,E,1,04,1.9,149.8,M,41.0,M,,0000*50
$GPGSA,A,3,17,12,18,15,,,,,,,,,3.3,1.9,2.7*37
```

Figure 11: Screen shot of the GPS serial data feed

[8] Break out board - *Hardware that allows hand-access to densely-spaced pins on a microchip*

This is standard NMEA formatted position data. The UTC time, Latitude, Longitude, GPS Quality Indicator, Number of satellites in view and Altitude is needed from the $GPGGA string [3]. This should cover the most of the Ground Station's needs, but since one of the tasks was to improve the GPS accuracy. The dilution of precision, both position and vertical from the $GPGSA [3] should also be include, all together, eight data sources. The position information is formatted in decimal minutes (ddmm.mmmm), which is difficult when converting to distance; it was therefore chosen to convert this to decimal degrees (dd.dddddd).

The GPS data handling and parsing were originally intended to run in the Core application. Instead it was chosen to parse the necessary data into a new data string on an embedded platform before it was sent to the ground station. This is to create modularity, if it is decided to change too another GPS module in the future. To do this some electronics was needed.

2.3.2 GPS electronics

For the GPS electronics the old LocalHawk electronics have come to good use, mainly the GPS and IMU circuit boards. See chapter 1.2 RF electronics for polarity and more information.

Figure 13: IMU circuit board

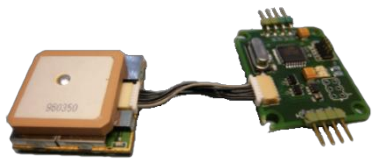


Figure 12: GPS circuit board whit connected GPS module

The GPS module is connected to the GPS circuit board (Figure 12) and then the data is transmitted over I2C to the IMU circuit (Figure 13).

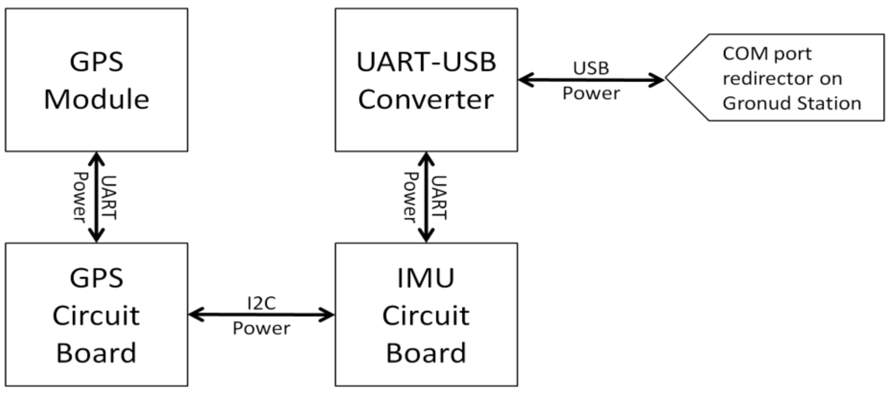


Figure 14: Microcontroller connection schematics

GPS Microcontroller

The microcontroller on the GPS circuit reads the UART that the GPS module connected is sending to. The circuit then checks the start characters of the string received to determine what is to be extracted from that string. It also converts longitude and latitude from decimal minutes to decimal degrees then saves all the values to a struct[9] and makes that struct available on the I^{2}C bus.

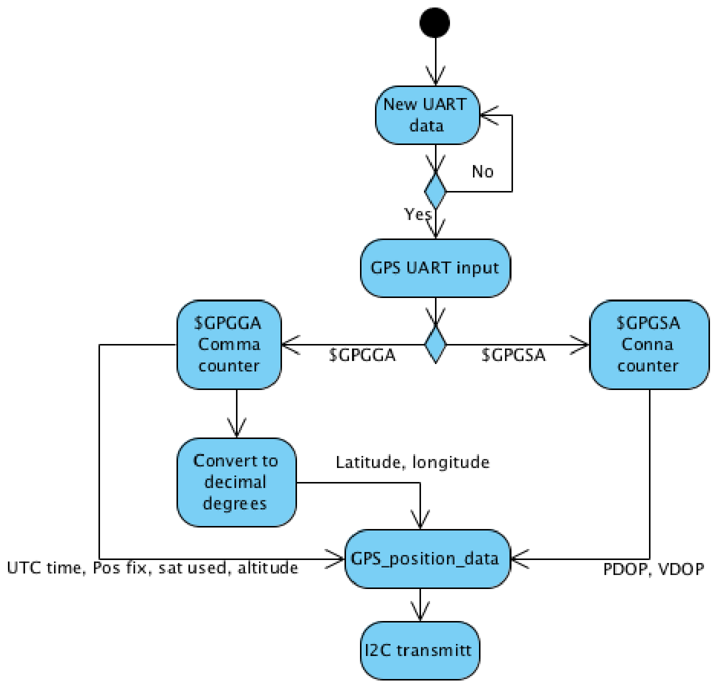


Figure 15: GPS Flowchart

IMU Microcontroller

The microcontroller on IMU board only has one task. When new position data is available on the I2C bus. It takes that data and creates an information string.

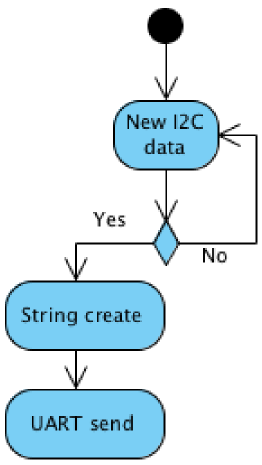


Figure 16: IMU Flowchart

[9] Struct – is a structured record type with a fixed set of labeled variables used in the C language

That string is formatted according to the following comma separated setup

$$\{1, 2, 3, 4, 5, 6, 7, 8\} \quad (2.2)$$

1. Position fix indicator
2. Number of satellites in view 0-10
3. UTC time formatted: hhmmss
4. Longitude in decimal degrees
5. Latitude in decimal degrees
6. Altitude in meters
7. Dilution of precision – (See chapter 2.3.3)
8. Vertical dilution of precision – (See chapter 2.3.3)

The formatted string is then transmitted over UART with following settings (38400, 8, 1, N, no flow control).

To read this data on the Ground Station, the same UART to USB convert chip as was used to read the raw data from the GPS was chosen. This provides a steady 5 volts power source in addition to a virtual COM port. This eliminated the need for a separate power source, which would be required if a conventional RS232 to USB converter was to be used.

As this integrated circuit reads transistor-to-transistor logic UART. The RS232 level converter that is soldered on the IMU board has been bypassed to get only the 0 – 5 volts UART signal.

Any GPS module will work with this system as long as it is set up to transmit the data over a RS232 or a virtual COM port to the Ground Station, according to the aforementioned string configuration (2.2). This is done to keep the modularity of the system as high as possible.

2.3.4 GPS Accuracy

There are many factors that affect the accuracy of any GPS signal.

- Satellite geometry

  Satellite geometry describes the position of the satellites to each other from the view of the receiver. Bad satellite geometry can lead to position errors of 100 – 150 meters. The GPS provides an indication of how good the satellite geometry is in the form of a DOP (Dilution of Precision) value. Ranging from 50 to 0, where 50 is bad.

- Satellite orbit ± 2.5 meters

  Slight shifts in the satellites orbit due to the sun and moons gravitational forces.

- Multipath effect ± 1 meter

  The reflection of satellite signals on objects, such as buildings.

- Atmospheric effect ± 5.5 meters

  Reduced speed of propagation in the troposphere and ionosphere, leading to refraction of the signal waves.

- Clock inaccuracies ± 3 meters

  Difference in the receiver clock and the satellite time during the position determination also calculation, rounding and relativistic effect errors factor inn.

The EM406 navigation satellite receiver has, according to its datasheet, a 10 meters 2D RMS [10]accuracy. The device was tested outdoors on a clear day to see how the accuracy was compared to what the data sheet said. Collected one thousand readings and plotted those in MatLab by converting the log files to a MatLab readable format using a custom-made conversion program (See chapter 2.6). (Both the conversion program and the plotting file is found on the CD).

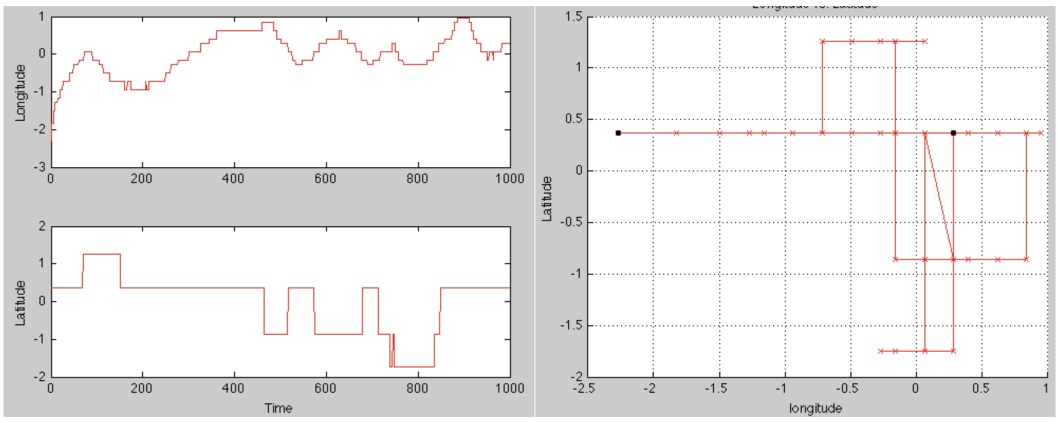


Figure 17: Plotted position data

[10]RMS – Root Means Square, statistical measure of the magnitude of a varying quantity

As the position data was gathered outdoors on a day with clear sky, with maximum number of satellites available (10) and a stationary GPS, these factors make the GPS position is as good as it can get.

This gave an error of ± 1.49 meters in the latitudinal direction and ± 1.59 meters in the longitudinal direction. The error numbers are converted from decimal degrees using the table on the right.

Decimal degree	Distance
1.0	111 km
0.1	11.1 km
0.01	1.11 km
0.001	111 m
0.0001	11.1 m
0.00001	1.11 m
0.000001	11.1 cm

Or 1 minute of latitude (1/60 of a degree) is 1 nautical mile. 1 minute of longitude (at the equator) is 1 nautical mile

2.3.5 Position enhancement

DGSP

Differential GPS is a way to enhance the accuracy of the GPS by using ones known position. It uses the known position and the GPS position of a stationary object to calculate the GPS position error (2.3). This differential is then subtracted from the GPS position you want to improve.

$$x_{error} = x_{known} - x_{GS} \quad (2.3)$$

Where x_{GS} is the Ground Station GPS position, whilex_{known} is the known position.

Position Filtering

As one does not always know ones precise current position it was needed a position accuracy-improving feature that did not rely upon user input information. Since the Ground Station is stationary for the length of the operation, a conclusion was made that some form of smoothing filter, or averaging the position, would stabilize the dispersal of the position and eliminates spikes in the position data. Then we would use the acquired position information in the same manner as known position in the DGPS (2.3)

We developed two different avenging filters, a mean value "filter" and a weighed filter.

Mean average filter

A mean average filter (2.4) uses the *N* last position values and averages them.

$$\hat{x} = \frac{1}{N}\sum_{i=0}^{n} x_{n-i}$$
(2.4)

Where *N* is the number of last positions you want to include and x_n is the last position value. This is dependent on the *N* last readings to get a good result, the higher the *N* value is the better the filter works. The mean value filter is resource heavy compared to the Weighed filter.

Weighted filter

The Weighed filter (2.5) is used to use measurements that are observed over time that contain random variations and other inaccuracies, and produce values that tend to be closer to the true value of the measurement.

$$\hat{x}_n = \frac{k-1}{k}\,\hat{x}_{n-1} + \frac{1}{k}x$$
(2.5)

Where $\hat{x}_n$ is the current filtered position, while $\hat{x}_{n-1}$ is the previous position, *k* is the weight coefficient (how much the last position is weighted). This filter is reclusive and need only the current position value and last filter position, this makes it less resource demanding to run.

To decide which of these two filters to use, simulations were done using MatLab.

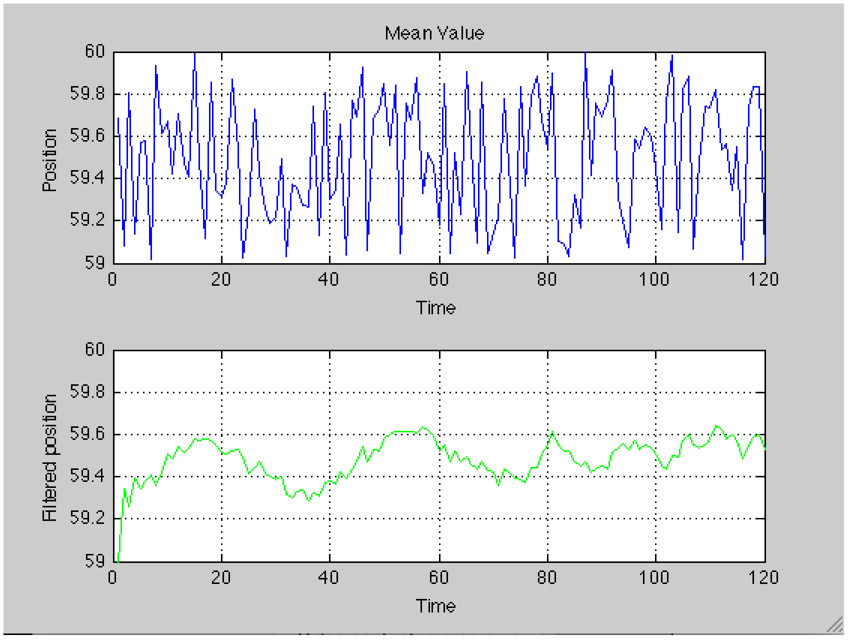


Figure 18: Simulated position (top) and the same position filtered with the mean filter (bottom)

The plot above is a simulation of the mean filter; where the last 10 positions are averaged (*N = 10)*. The target value is 59.5 and one can see that the filtered position (bottom) is a lot better than the unfiltered position (top).

The next plot is of the weighted mean filter, with a *k* value of 50 = 2% with the same original unfiltered values.

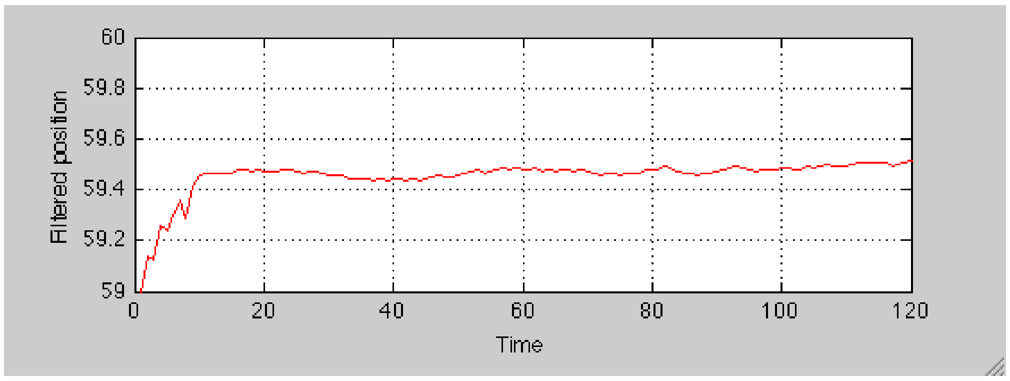


Figure 19: Weighed filtered value plotted over time

This filter is even better than the average filter and a great improvement over the original signal.

The image below shows the two filters superimposed so one can clearly see the difference, red being the weighted filter and green the average.

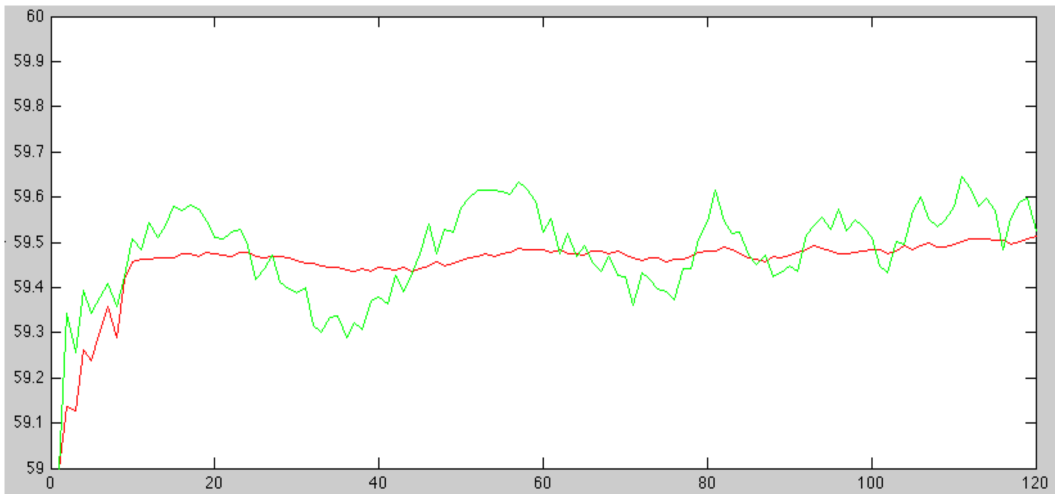


Figure 20: Both filter plots superimposed

Naturally the Weighed filter was chosen, which was implemented into the ground station.

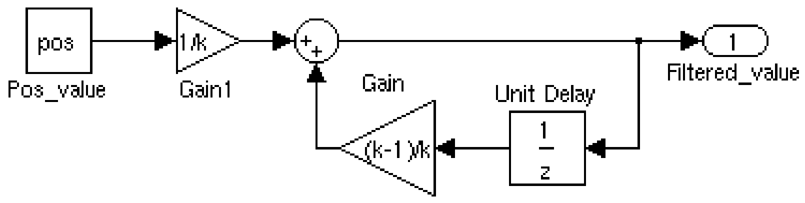


Figure 21: A graphical representation of a weighted filter

The next plot shows the same data (red) as in Figure 22 run through the weighted filter (black).

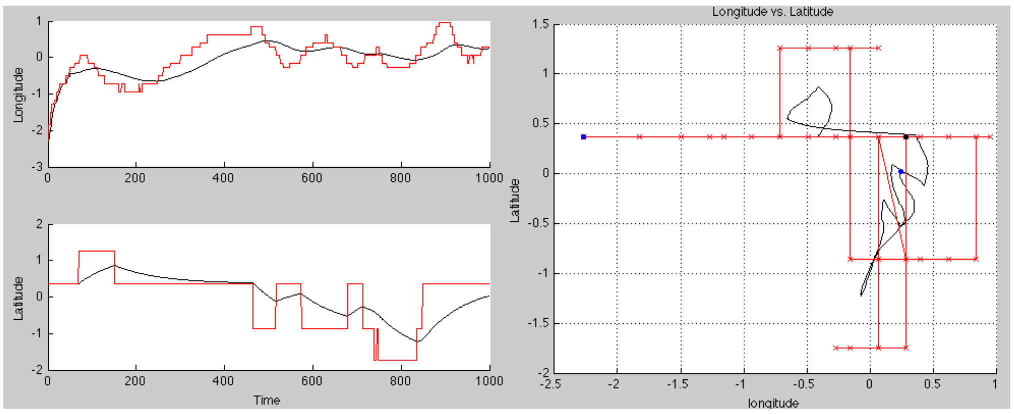


Figure 22: Filtered data superimposed on figure 20

Here one can see that the filter is not as stable as in the simulation this is because the signal is less evenly distributed, but with a Filtered (unfiltered) error of ± 1.3 meters (1.49 m) in the latitudinal direction and ± 1.45 meters (1.59 m) in the longitudinal direction. This is an improvement of ± 0.19 m and 0.14 m correspondingly over a thousand seconds (16.5 min)

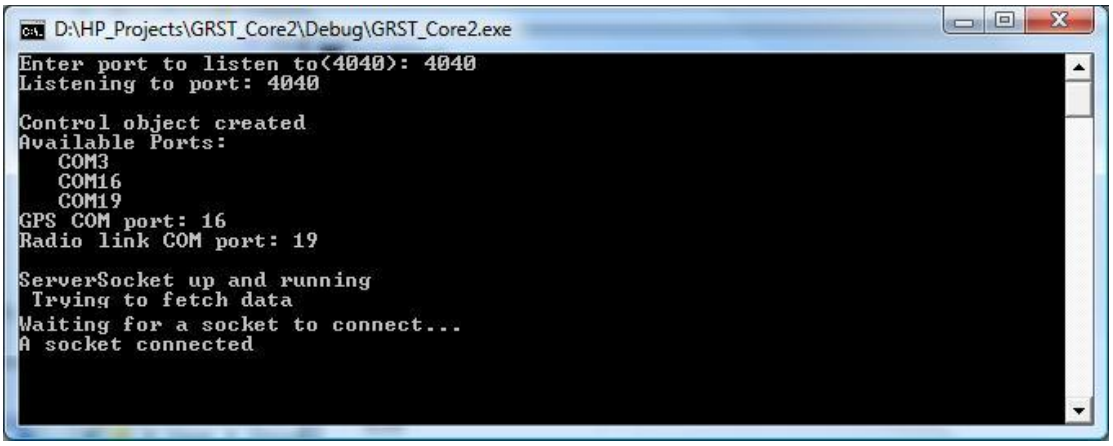
2.4 Core application

The Core application is the link between the UAV and the GUI. Its task is to get data from the GPS and radio link and send it to the GUI. The Core is an event based application, after initial startup it starts listening for events from the GUI, and responds accordingly.

2.4.1 Core operation

On startup the Core requests user input about what port the Server Socket (see chapter 2.4.2) should listen to, and what COM ports the GPS and radio link is connected to. It then initiates the Socket and requests data from the GPS and radio link, after receiving the data from the serial ports (see chapter 2.4.3) the data is run through an error check (see chapter 2.4.4). This was needed because without it there were some problems with that the GUI crashed because of array overflow.

When a socket connects the server socket starts sending and logging data. This will continue to do so as long as it receives a request for more data. After the error check the data was sent through the socket to the listening sockets connected and then logged to a text file (see chapter 2.4.5).



```
Enter port to listen to(4040): 4040
Listening to port: 4040

Control object created
Available Ports:
   COM3
   COM16
   COM19
GPS COM port: 16
Radio link COM port: 19

ServerSocket up and running
 Trying to fetch data
Waiting for a socket to connect...
A socket connected
```

Figure 23: Core, with all needed input and a connected socket

2.4.2 Socket

The first socket made was an asynchronous socket, but this was difficult to both implement and use compared to an event driven architecture, so today's solution is a Win32 Socket, built using network event architecture. This means that it reacts with preset responses to the chosen network events. Following is a response code example to a write event:

```
//Network write event:
if (NetworkEvents.lNetworkEvents & FD_WRITE){
            if (NetworkEvents.iErrorCode[FD_WRITE_BIT]){
            cout<<"Could not write, error:\n"
                <<NetworkEvents.iErrorCode[FD_WRITE_BIT];
      }
tempdata = mySerialData->Complete_data_send();
buff = tempdata.c_str();
send(SocketArray[Index - WSA_WAIT_EVENT_0], buff, 200, 0);
}
```

The socket can connect locally on one pc or over any LAN, or by the use of virtual LAN (for example Hamachi) it can easily connect over the internet. The current server socket may have as many as 64 client sockets connected at any time.

Client socket

The client socket is the GUI side of the interface between the core and GUI. It contains a different read event than the server socket. It does not have the capability to connect to more than one socket.

Server socket

The server socket is the core side of the interface between the Core and GUI.

2.4.3 Serial read

As the GPS and Radio link both communicate over some form of COM port the Core need to be able to get to the data made available there. First thing tried was to use a community developed serial port library. Because this was a non-native library, it was heavy and difficult to use, and had some stability issues, consequently this solution was discarded.

While researching for another solution, System namespace class library called "SerialPort()" [4] was found, that juxtaposed against the community developed library was easy to use and well documented. This class supported both read and write over serial port.

```
String ^Read(){
        try{
                _serialPort->Open();
                String^ buffer = _serialPort->ReadLine();
                _serialPort->Close();
                return buffer;
        }
        catch (TimeoutException ^) { }
        catch (...) { }
}
```

This class was used to create methods that initiates two serial ports, read both and write to the Radio link serial port. The only downside being this was a .NET framework library that made the code managed as opposed to the rests of the code, which is unmanaged.

> ***Managed C++***
> Managed code requires and will only execute under the "management" of a Common Language Runtime virtual machine, a core component in Microsoft's .NET imitative. The CLR provides services such as memory management and exception handling

To work around this, the resulting managed string was converted to unmanaged, which is done in the error check class.

2.4.4 Error Check

One of the overall requirements for this system was that it all should be failsafe; this is supported in the GPS related code throughout the system. If one was to disconnect the GPS somewhere along line, the system will stop updating its Ground Station position and thus keeping its last known position. Except if the entire GPS peripheral is disconnected from the Ground Station computer, then the Core application will prompt the user for a new GPS COM port number.

Error check

An error check was implemented to prevent data overflow, or other unexpected errors linked to symbol issues. The error check was implemented in the core application where the raw data is analyzed. This error check verifies that the GPS and RF data string is purely numerical and that they have the right length before the data strings are merged, sent over the socket to the GUI and logged.

This was found to occur especially when the aircraft-side radio link was low on power, which caused corrupted information to be sent to the core application, which was relayed to the GUI and made the arrays there overflow and crash the system. Some of the corruption caused the string to almost double in length and contained strange symbols.

The error check solves the problem and returns a single digit if something is wrong. This digit represents the type of error found. This will be logged instead of the corrupted information and the data in the GUI will not refresh. The error check also request new COM port information if serial connection is lost after startup.

Since both the GPS and RL data strings arrives from the Serial _Read class as managed text strings (System::String^) they are both converted to unmanaged (std::string) in this class. This is achieved by marshalling the string using the marshal_as method in the marshal_context class library [5]. The opposite conversion is done for the data string that is to be sent over RL from the Ground Station.

2.4.5 Logging

The data is stored to a text file named “LocalHawkLog.txt” that can be found in C:\ after the first time the Core application is started.

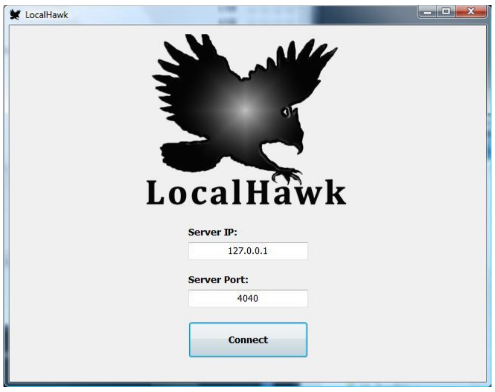
```
//Write to log file:
void Log::storeData(string sData) {
      out.open("c:/LocalHawkLog.txt",ios::app);
      out<<sData<<"\n";
      out.close();
}
```

2.5 Graphical User Interface

The Graphical User Interface is an application that is connected to the Core and receives data that the Core has relayed. The data received is displayed in a set of tabs.

2.5.1 GUI design

The GUI consists of two windows, a connection window (figure 24) and the display window (figure 25).



Server IP: 127.0.0.1

Server Port: 4040

Figure 24: GUI connect window

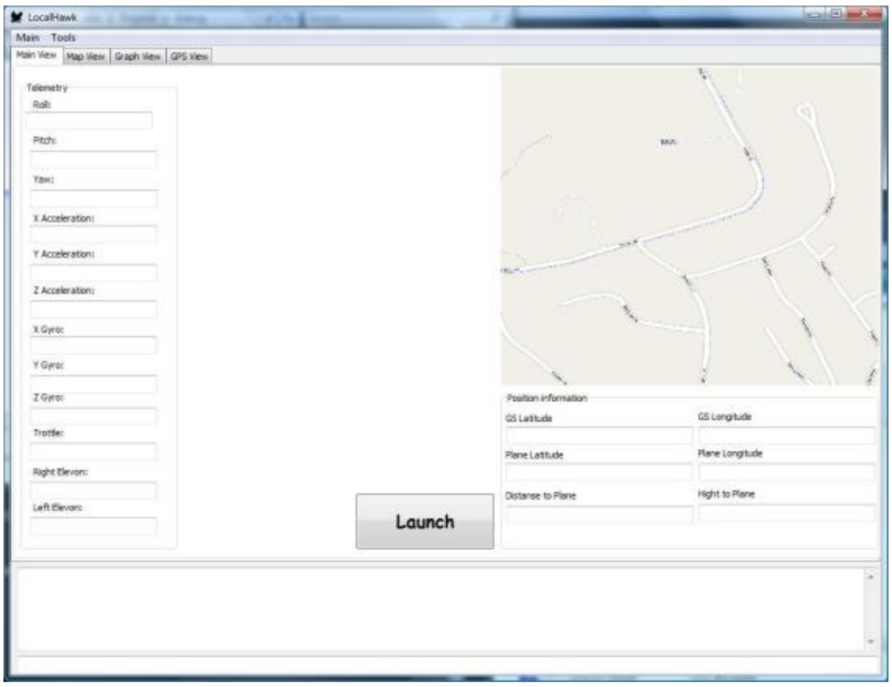


Figure 25: GUI main window

Underneath the GUI there is a Client socket (see chapter 2.4.2). As the GUI receives data form the core the data is parsed and distributed the data.

2.5.2 Functionality

The connection window is displayed when the application is started. It contains two input boxes that are used to set the IP address and port number of the Server socket. Default values of these are:

IP: 127.0.0.1
Port: 4040
These are the settings you want if you are running the Core and GUI on the same computer. After pressing the connect button, the main GUI window appears. It contains a console at the bottom of the window, which displays any errors messages for any errors that might occur. You can also switch between 4 tabs that each displays different information:

Main tab:

The main tab is the default tab, this tab contains a small map which can easily be changed (using a map import function) pending on where you are going to fly the aircraft. The map displays the position of the GPS and aircraft. It has all the functionality to set waypoint for the plane but this is not in use in current version.

It was first planned to use the Google map API for displaying the map, but the term of use, demanded that the map needed to be published on a website. Research was then done on other map API`s: Eniro, OpenStreetMap and Mapnic. But found that they were ether protected but user agreements or did not support C++. It was also considered creating the map renderer for OpenStreetMaps, but this would be too time-consuming, therefore a simple image of the wanted area was used.

To display the position of the plane and ground station on the map, a method was created to convert coordinates to pixels and pixels to coordinates.

```
POINT TForm1::coordToPix(double lati, double longi,
double dXPx, double dYPx){
            POINT p;

      p.x = fabs(longi-xMapEdge)/dXPx;
      p.y = fabs(lati-yMapEdge)/dYPx;

return p;
      }
```

Import map displays an import group box that allows the selection of the wanted map image, and input of the 4 GPS coordinates of the map edges. It is important that these coordinates is correct or the position of both ground station and aircraft will be wrong on the map.

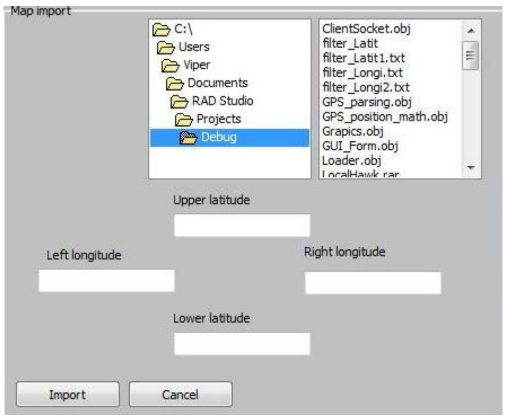


Figure 26: Import map box

OpenStreetMaps [6] have a great tool for downloading both maps and related coordinates.

The main view also has two group boxes displaying information.
This box contains different information about the aircraft:

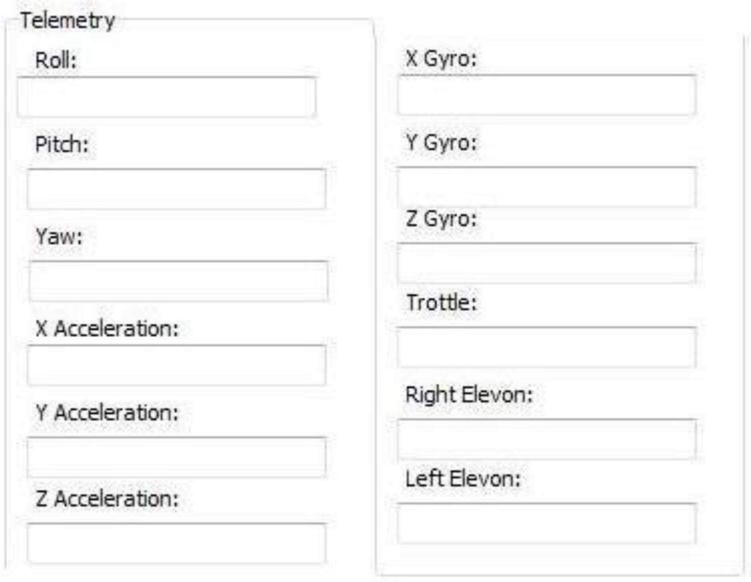


Figure 27: Telemetry box

Position information:

This box contains position information to the ground station and aircraft:

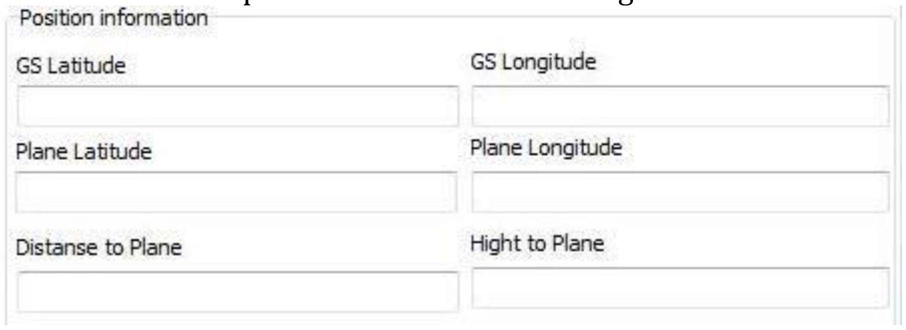


Figure 28: Position information box

Map tab:

The map tab contains a larger map that has the same functionality as the map in the main tab, as well as the Position information box.

Graph tab:

Graph tab contains a grid that can display received data, and that can graph displaying the wanted data from the grid

GPS tab:

GPS tab contains 5 boxes. 3 of them show different information about position, a position filtering box and satellite status:

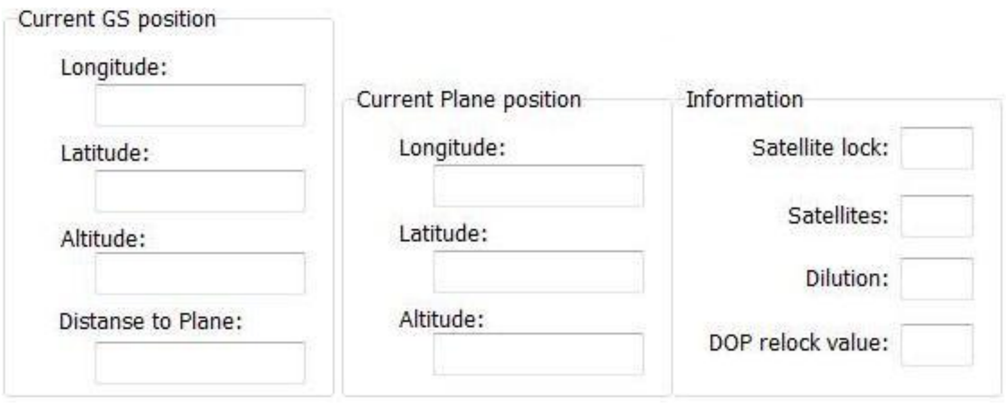


Figure 29: GPS view, information boxes

One enables filtering of GPS position (see chapter 2.3.4):

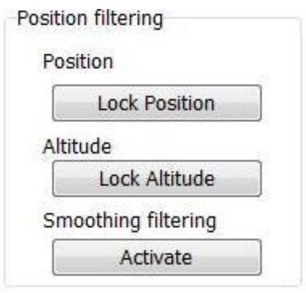


Figure 30: Position filtering box

One which enable manual insert the position of the ground station, the coordinates and altitude inserted here is used to calculate the differential GPS:

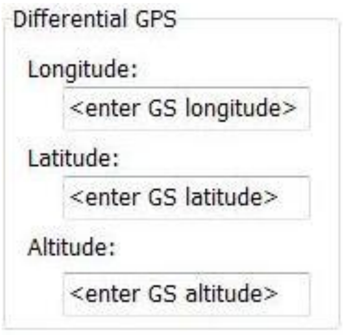


Figure 31: Differential GPS input box

2.5.3 Data possessing

After the socket is connected and the first data string is received, the string is separated into individual data variables and made into numerical values for further mathematical manipulation (see chapter 2.5.4).

To achieve this the vector class library [7] was used. A vector is a dynamic array and allows for the use of the push_back() method, which adds a character to the end of an array.

```
while ( getline(iss, token, ',') )
      {
            comma_count++;
            array.push_back(token);
            switch(comma_count)
            {
                  case 1(...)

                  ...
```

For assigning variable names to the string values a simple comma counting loop and push_back() was used.

2.5.4 Mathematical calculations

Some of the values needed in the GUI needs be calculated from other values.

Ground Station to UAV distance:

The distance is derived from the haversine[11] formula [8].

$$d = 2R\sin\left(\sqrt{\sin\big((\phi_1 - \phi_2) + 0{,}5\big) + \cos\phi_1 \cos\phi_2 \sin\big((\lambda_1 - \lambda_2) + 0{,}5\big)}\right)$$

Where R is the earth radius in meters, λ_1 and λ_2 is longitude in radians, ϕ_1 and ϕ_2 ϕ_1 is the latitude radians.

This formula takes the curvature of the earth into account and finds the distance between two points.

[11] Havesine – formula for special case of spherical trigonometry

2.6 File Converter

To easily study and analyze the data collected, an application to convert the log files was created.

It can be used to filter out the log entries that contain an error code. When using this function it also counts the number of each error occurrence.

This application can convert the standard log files to *.fdr files and MatLab (*.m) files. This enables the logged data to be opened in different types of graphs and analyzed.

2.6.1 Usage

Operating the program is very intuitive; you first select the wanted log file and where to store the output file and type in the wanted name on the file and select the wanted conversion options.

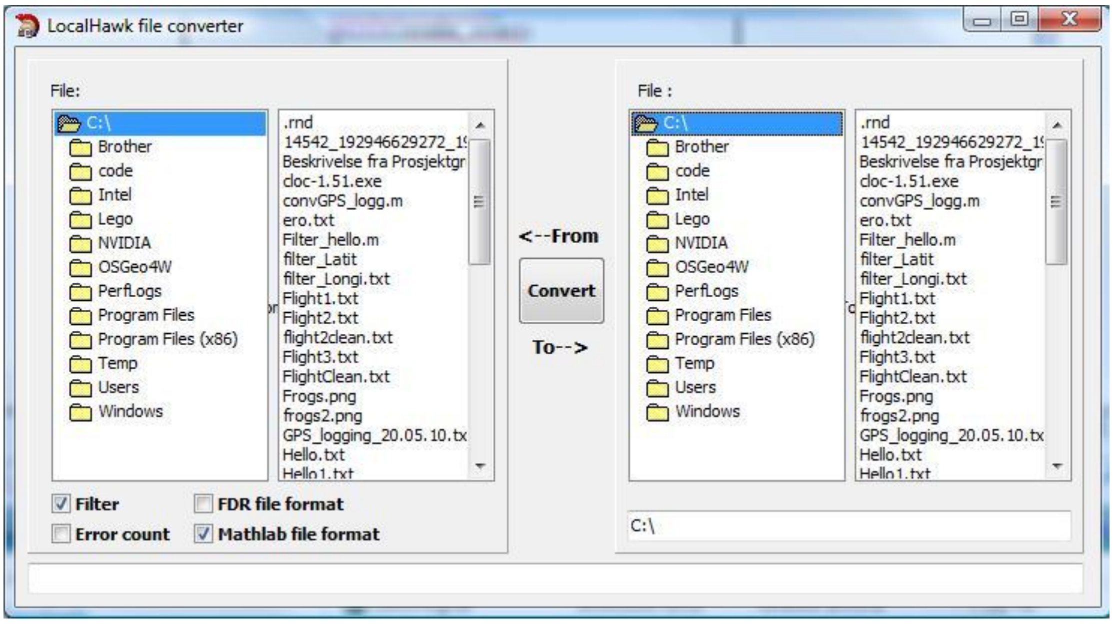


Figure 32: Log file converter

2.6.2 .fdr (Final Draft Document)

The converter goes through the selected log file and replaces all "," with an empty space, and removes the log entries with an error code.

2.6.3 .m (MatLab)

The converter goes through the selected log file, removes the log entries with an error code, and stores all the different variables in a temporary array and uses the array to write it all to a file out in the correct order and under each variable name. In the directory of the conversion program there's a MatLab script that plots the resulting *.m file data into graphs.

2.7 Serial port monitor

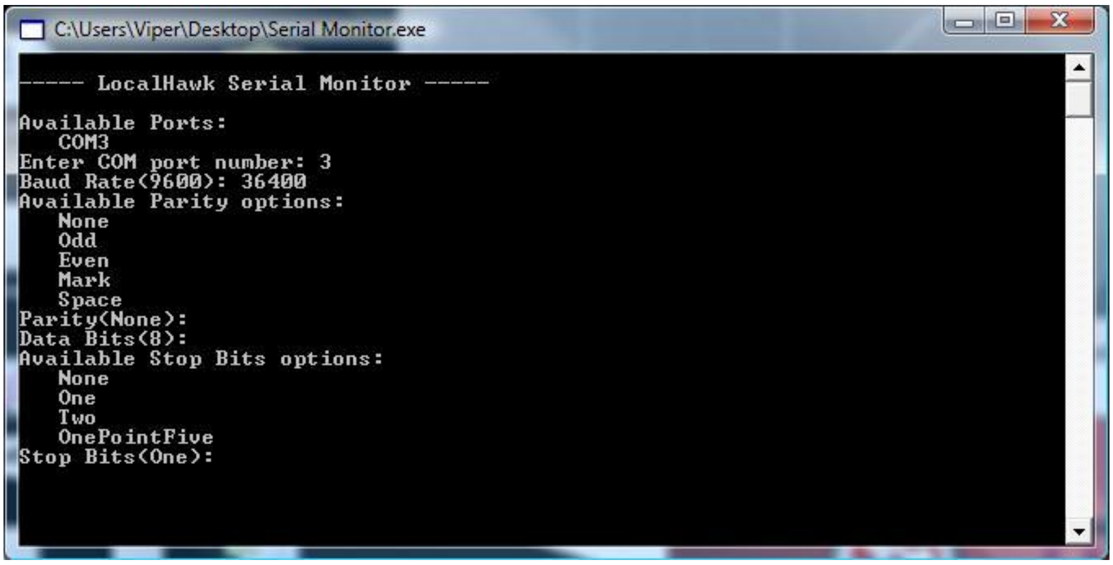
To easily debug and verify raw serial outputs of the Radio and GPS unit a serial port monitor was created. This is a program that listens to one of the computers serial port and displays the information received.

This is a console application with no graphical user interface. It relies in five input parameters, from the user to configure the COM port listening correctly.

2.7.1 Usage

To set up the program to receive the data correctly it asks for the following input

- COM port: The program lists all the available COM ports then prompts the user to choose one. Type the COM port number you want to read off of.
- Baud rate: If nothing is entered the program assumes default Baud rate, 9600
- Parity: Default parity is none
- Data bits: Default data bits 8
- Stop bit: Default stop bit 1



C:\Users\Viper\Desktop\Serial Monitor.exe

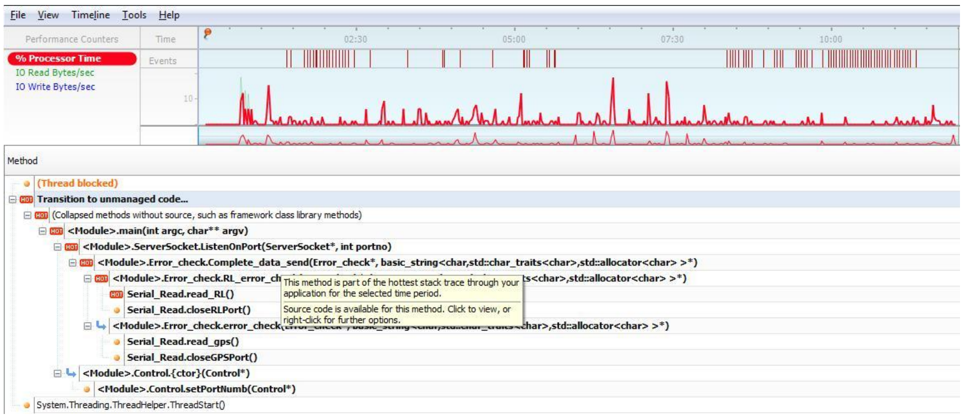
```
----- LocalHawk Serial Monitor -----

Available Ports:
   COM3
Enter COM port number: 3
Baud Rate(9600): 36400
Available Parity options:
   None
   Odd
   Even
   Mark
   Space
Parity(None):
Data Bits(8):
Available Stop Bits options:
   None
   One
   Two
   OnePointFive
Stop Bits(One):
```

Figure 33: Serial port testing tool

After the program is configured it starts to display the serial read feed.

2.8 Performance

To monitor the performance of the core used ANTS performance profiler was used.



```
(Thread blocked)
Transition to unmanaged code...
  (Collapsed methods without source, such as framework class library methods)
    <Module>.main(int argc, char** argv)
      <Module>.ServerSocket.ListenOnPort(ServerSocket*, int portno)
        <Module>.Error_check.Complete_data_send(Error_check*, basic_string<char,std::char_traits<char>,std::allocator<char> >*)
          <Module>.Error_check.RL_error_ch                                          ts<char>,std::allocator<char> >*)
            Serial_Read.read_RL()
            Serial_Read.closeRLPort()
          <Module>.Error_check.error_check                                          char>,std::allocator<char> >*)
            Serial_Read.read_gps()
            Serial_Read.closeGPSPort()
      <Module>.Control.{ctor}(Control*)
        <Module>.Control.setPortNumb(Control*)
System.Threading.ThreadHelper.ThreadStart()
```

This method is part of the hottest stack trace through your application for the selected time period.

Source code is available for this method. Click to view, or right-click for further options.

Figure 34: Core application’s CPU usage when it receives data

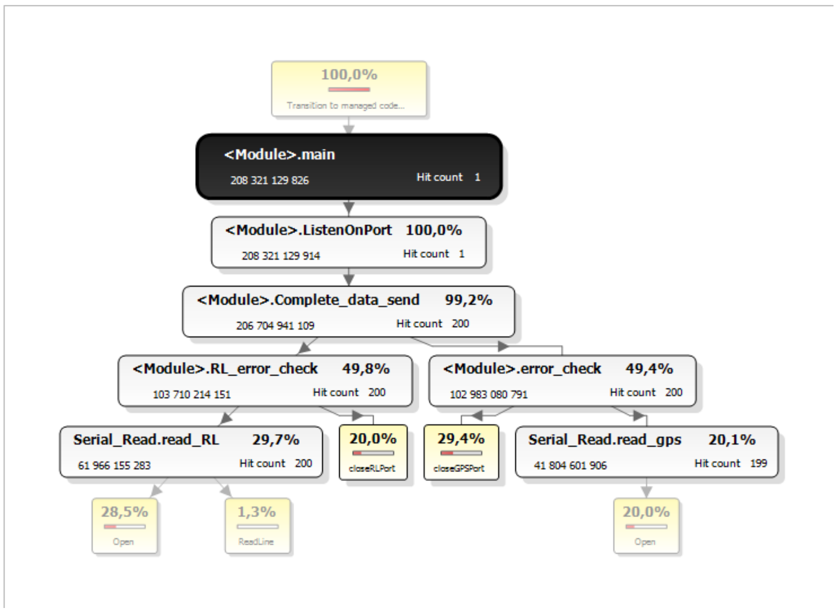


Figure 35: Core CPU time usage, over a selected 5 minutes

After running tests with ANTS profiler, it was clear that it is the Serial Read methods read_gps and read_rl (methods that get the data from the serial ports) was the most CPU[12] demanding part of the Core application.

[12] CPU – Central Processing Unit

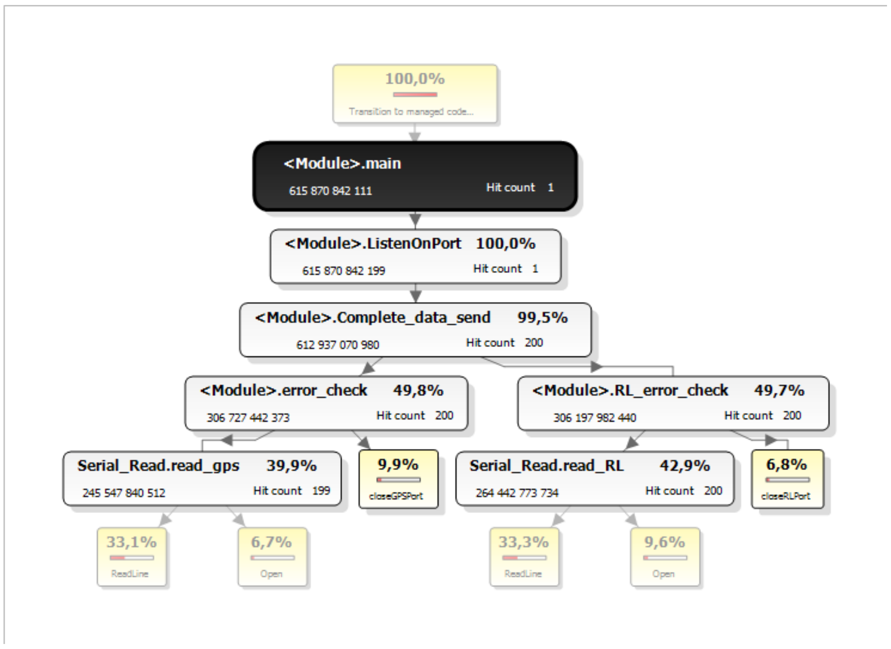


Figure 36: Core CPU Wall clock time[13], over a selected 5 minutes

As one can see from figure 38 and 39 these two methods stand for 82 % of time usage and about 50 % of the CPU usage of duration of 5 minutes. That equals to more than 1 and a half minutes are devoted to waiting for the port's to close, open and waiting for information.

If the connection is lost between the Ground Station radio link and aircraft radio link, the read_rl() method times out, this creates a drastic increase of the wall clock time for this method.

2.9 Future outlooks:

2.9.1 Core application:

- Most of the resources the Core application draws, goes towards listening to the serial port. This is not posible to get around, but by giving these two tasks there own threads and implementing circular buffers this will increase the data flow and overall speed of the system.
- The graph view was not completed and needs work.
- Develop a more permanent Ground Station GPS electronics

2.9.2 GUI:

Adding threads to do the parsing, and mathematical calculations, will increase the overall speed.

[13] Wall clock time – The sum of communication delay, CPU and I/O time, resulting in time to complete task.

3 Chapter 3

Launcher

3.1 System

The Launcher was developed during a student project given by KDS summer 2009. The main reason for developing a launcher was to make takeoff for the LocalHawk aircraft autonomous, and have more control of the aircraft during takeoff.

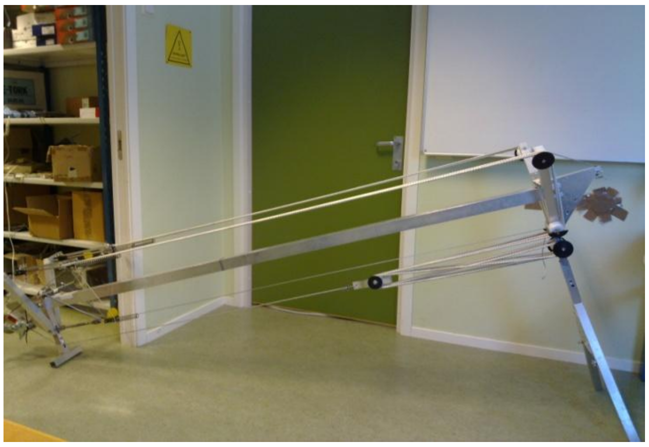


Figure 37: Launcher

The Launcher is a catapult launcher accelerating a wagon from rest to launch speed. The force is created by two double rubber bands, which are extended with a winch. A release mechanism holds the wagon in place, and releases the stored energy when launching. The wagon accelerates smoothly all the way, and some energy is still stored in the rubber bands after launch. When the wagon hits the springs it decelerate (negative acceleration) rapidly, and the aircraft gets airborne.

The launcher needed some modification to function properly. The areas given to work on were:

- The Wagon, making it capable of launching aircrafts with both front and rear mounted propeller.
- The release mechanism, creating a functional system.

3.2 Wagon

The wagon is a part attached to the launcher. Its purpose is to accelerate and get the aircraft in the air. The wagon created is limited to only launch aircrafts with rear propeller. The reason to create a new wagon was to increase its use of utilization. Giving it the possibilities to launch both front and rear mounted thrust.

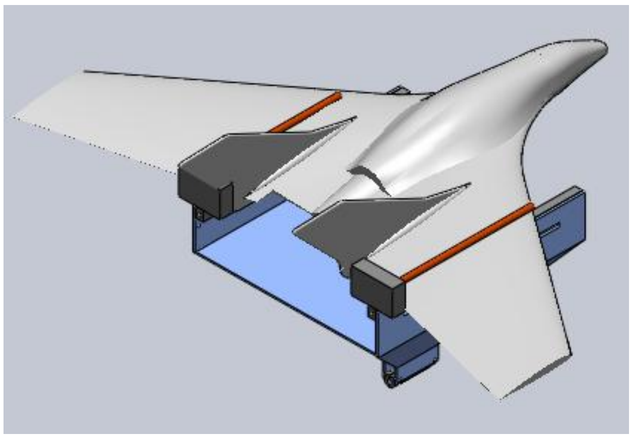


Figure 38: Shows a drawing of the wagon holding the LocalHawk airframe.

3.2.1 System Analysis

Before creating a new wagon design, research on the launcher had to be done. Firstly, it was important to find the current magnitude of the launch speed, and how this parameter varied with different weights. It was also crucial to know the forces affecting the wagon.

3.2.2 High speed analysis 1

A Casio EX-F1 given from KDS that also take high-speed video was used. The camera was capable of recording in 1200fps, but then the image was too small to analyze (336x96). When recording the launch sequence the camera was set to only 600 fps (432x192). The camera also needs a lot of light to get a clear image.

The first test was to find out the wagon speed in relation to weight.

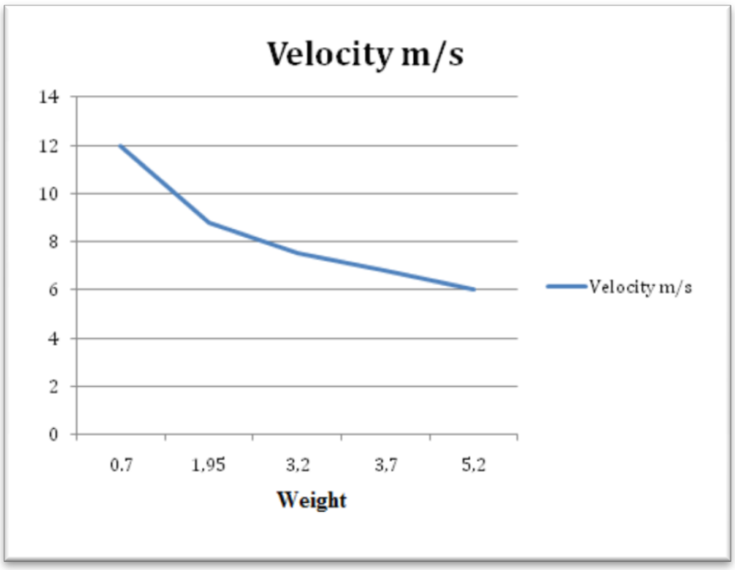


Figure 39 - Wagon velocity vs. weight

From figure 39, one can observe that the total weight of the wagon system affected the speed exponentially. The readings were a bit disappointing, since the aircraft needed at least 10 $^{m}/_{s}$ to get in the air. The figure shows that the wagon reach a total weight on 1, 60 kg (1.60kg-0.80kg (aircraft) =0.80kg) in getting the minimum velocity limit at 10 $^{m}/_{s}$.

3.2.3 Different kinds of energy storage (rubber band)

Because the first tests results failed the requirement, optimization on the entire system had to be done. For optimizing the system, the parts on the launcher needed to be analysed. Finding which parts affection the wagons velocity most.

Possible changes:

1. Different Rubber bands or another type of potential elastic energy.
2. Change the pulley system for reducing the friction.
3. Change or improve the guiding wires for reducing the friction.

The launcher had to be tested, finding different possibilities to gain more speed. The first thing tested was point 2, changing the pulley system. Since the pulleys drained so much force because of friction, it was optimal trying to eliminate them. The system was changed, placing the rubber bands under the launcher connected to a strong rope that was attached on the wagon. With this the rubber band pulled the rope with the wagon. When tested it was found that it was easier to gain more force, but the rubber band did not accelerate the wagon all the way, as before. Instead it gave the wagon a more rapid force, only when released.

The old system accelerated the wagon all the way. This velocity was only a bit higher, but the system was somewhat awkward.

A tension spring was also tried, replacing the rubber bands. The result was worse because of the small elongation in the spring, mainly since the force needed to stretch it was so high.

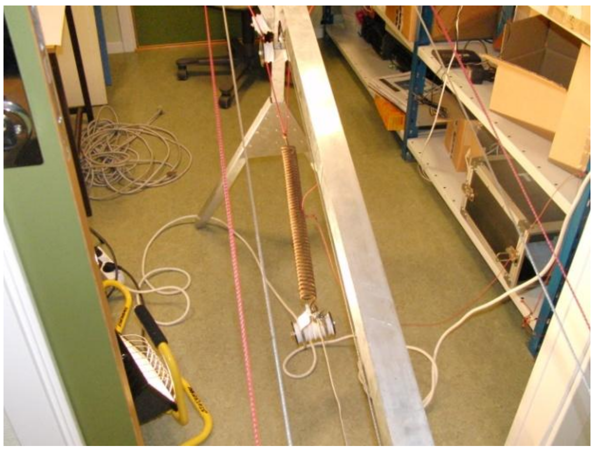


Figure 40 -The launcher with a tension spring

Then the guidance wires that support the wagon, needed to be analyzed. An idea was to change the whole system, using rails instead of wires. This was not only because of the reduction in friction, but also to improve the safety. The solution led to more changes on the system than wanted, and was rejected. Instead change was made on the wire clamps, and tightening all the wires.

When they were tightened, a new test was executed with the high-speed camera. This time the result was satisfied. More testing was performed with different kind of loads.

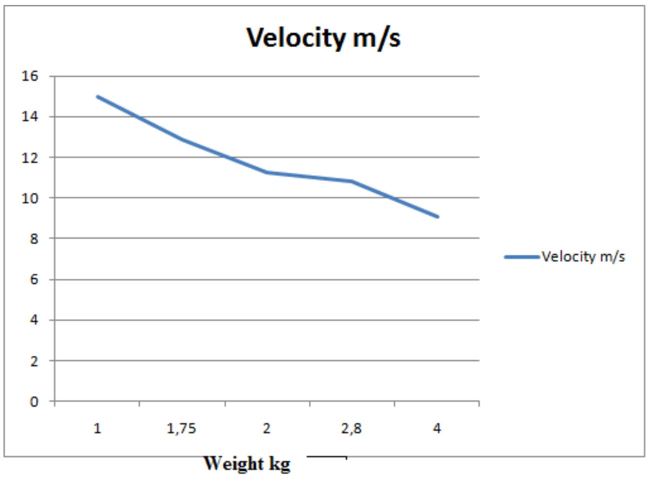


Figure 41 -Wagon velocity/weight

Calculation in change of velocity with regard to weight theoretically was also made.

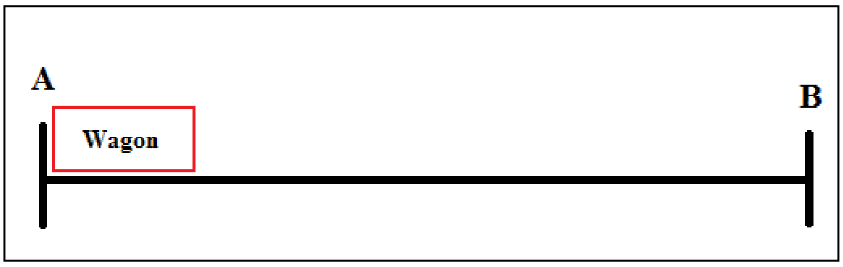


Figure 42 -Theoretical drawing

$$Energy\ start = Energy\ end$$

$$\frac{1}{2}kx_1^2 = \frac{1}{2}mv^2 + \frac{1}{2}kx_2^2 + R * S \qquad (3.1)$$

, where ($x_1 - x_2$) is change in distance, m is mass, v is velocity, R is friction, and S is length from A to B.

$$v = \sqrt{\frac{kx_1^2 - kx_2^2 - 2RS}{m}} \qquad (3.2)$$

By setting all independent variables the wagon's velocity is can be calculated

$$v = \sqrt{\frac{C}{m}} \qquad (3.3)$$

, where

$$kx_1^2 - kx_2^2 - 2RS = C = constant \qquad (3.4)$$

The equation (3.3) shows that quadrupling the mass will bisect the velocity. This matches the graph from figure 44.

3.2.4 Acceleration and friction

The acceleration of the wagon was also needed to be found.

$$\sum F = m \cdot a \qquad \text{1.3. Newton's 1. Law}$$

$$a_{max} = \frac{kx_1 - R}{m}$$

From equation 3.4

$$R = \frac{\frac{1}{2}kx_1^2 - \frac{1}{2}mv^2 - \frac{1}{2}kx_2^2}{s}$$

, where X_1 = 3,01m, X_2 =1,40m, s=1,63m and k=2*63

If weights of 1, 75kg (approximately wagon + aircraft) are added to the wagon, a speed of 13,1 $^{m}/_{s}$ is archived. Then R=182,3N and the max acceleration a=111, 8 $^{m}/_{s^2}$, but that is only for a very short period.

$$A_{avg} = \frac{\Delta V}{\Delta t} = \frac{13{,}1}{0{,}27} = 48{,}5\ {}^{m}/_{s^2} = 4{,}95g$$

Because testing was performed using weighs instead of the aircraft, the result may vary a bit from the actual result. The air resistance on the aircraft is less. The velocity and time from the high speed camera is also a bit inaccurate, Not only because of the "low" frame rate, but also the camera/software quality. When analyzing you could see that the frame length might vary, giving inaccurate data.

3.2.5 New Wagon design

The wagon had to be modified so it could launch aircrafts with front or rear mounted thrust.

The system requirements for the new wagon system:

- Be able to launch both types of aircrafts (front/rear mounted propeller).
- Give the aircraft the desired take-off speed on at least 10 $^{m}/_{s}$
- Be able to launch aircrafts with up to 11 inch propellers.

3.2.6 Concepts

Four possible concepts that fulfil our requirements were found. These concepts have different levels of complexity and workload.

Concept 1

The systems function is the same, but consists of two different types of wagons. One for front mounted propeller and one for back mounted. The one for the back mounted propeller already exist. The other wagon needs to be created from scratch. The idea with this new wagon is to have the airframe in a type of mould. The airframes propeller is also large 11 inch, so the platform where the airframe lies needs to be lifted up more than the existing wagon.

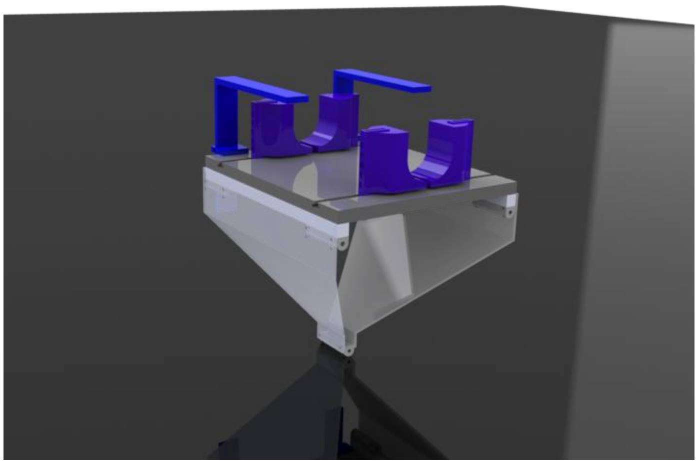


Figure 43 - This drawing show an idea of the new wagon

Concept 2

To design a completely new wagon that has the capability of launching airframes independent of the propellers position and size. This means that the wagon needs a lot of opportunities for adjustments.

Concept 3

The concept contains one main wagon frame instead of two different wagons, to reduce the modification needed to be done when launching both aircrafts. Then one "top" for each airframe type that can be assembled on the main wagon frame. The existing frame can be modified so it becomes the main frame, with possibilities to assemble each of the two platforms developed for each airframe.

Concept 4

To create a system that does not use the wagon for transporting the airframe. Instead of having a wagon, rubber band can be fastened directly on the airframe body. The idea is to have a mounting bracket to a wire and just drag this wire, and release. This concept is a more complex solution. It will maybe need one bracket for each airframe, and a different rail system.

3.2.7 Solution Choice

After evaluating the concepts, concept 3 was chosen. This was the best solution; with less change to the total launcher system than they others. All it needed was a new part for launching front mounted thrust aircrafts (like the Siren aircraft).

It was also decided to keep the old wagon, and just make a new top part, which could be mounted on top of it. This way the launcher kept its system for launching the existing aircraft, Stryker F27c.

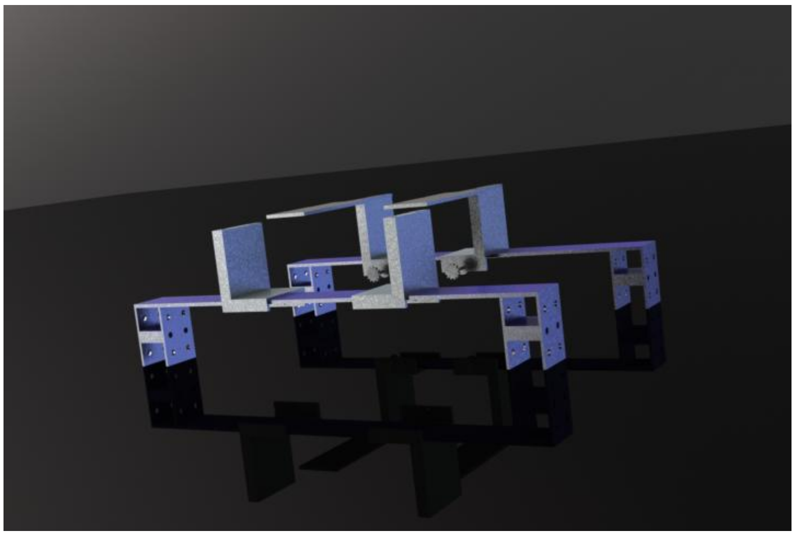


Figure 44 Shows a drawing of the first new wagon part

The part will not only fit the airframe, but also raise the aircraft body. This is important for preventing the big propeller on 11inch to hit the launcher frame.

Figure 7 show a drawing of the first idea for the new wagon part. The part weighs 700g with Aluminium 2025 T6[14].

3.2.8 Choice of material

The aircrafts working with are Stryker and Siren .These airframes need an takeoff velocity of minimum 10 m/s . Today the launch system gives the aircraft a velocity on 11 -13 m/s calculated with a total weight of 2,3kg. Therefore the weight is the most important factor when designing the wagon, but strength cannot be forgotten. This is because the wagon is first accelerated up to a velocity and then quickly decelerated down to zero velocity, with help from some springs. The most harmful force is when the wagon hits the frame.

If the Launcher didn`t have springs at the end, slowing the wagon speed before it hits the frame. The force would be:
(Without the aircraft weight, since it has left the wagon)

$$F \cdot t = mv_0 - mv_1$$

$$F = \frac{-mv_1}{t}$$

, since the start velocity is zero.

[14] See choise of materials appendix 1 #

$$F = \frac{-1kg \cdot 13m/s}{0.02s} = -650N$$
(3.5)

, (the springs will reduce a lot of the force even when they not are strong enough)

But the springs slow down the wagon and giving it at calculated stoppage time on approximately 0, 05 seconds, also reducing the wagon velocity.

$$F = \frac{-1kg * 8m/s}{0.05s} = 160N$$

Material

The first thing considered was Steel and Aluminium. Steel was rejected because of its weight, while Aluminium was preferred because of its weight and strength, and possibilities with different alloys. Another factor was that the launcher frame and the existing wagon were built in Aluminium so this would go well with the rest. Plastic was also considered, but this was also rejected because of lacking experience using it.

Aluminium was the chosen material, but which alloy to use was not selected. Available from KDS was 2024, 6061, 6082 and 7075 Aluminum alloys. After putting them up against each other, it was found that the one with most yield strength/density was using 6061 alloy.

Different heat treatment method for Aluminum alloys was also looked at. [9]. The one chosen was T6 because of it was the one that suited best, and because of its availability.

The materials properties, Aluminum 6061-T6:

Property	Value
Elastic Modulus =69000	69000 N/mm^2
Poisson's Ration	0.33
Shear Modulus	26000 N/mm^2
Mass Density	2700 kg/m^3
Tensile Strength	310 N/mm^2
Compressive Strength	N/A
Yield Strength	275 N/mm^s
Thermal Expansion Coefficient	2.4e-005 K
Thermal Conductivity	166,9 W/(m·K)
Specific Heat	896 J/(kg*K)

3.2.9 FEM Analysis

The Finite element method (FEM) is used for calculating critical parts or areas on an assembly. This method can find out exactly where there are most stresses affecting, and how much the force is. The method helps in finding critical areas.

Program used is Solidworks Simulation.

Procedure in SolidWorks Simulation.

1. Open your 3d-model over to the simulation area.
2. Chose type of simulation
3. Define how parts interact with each other(how they are connected)
4. Define the materials
5. Mesh the model (dividing the part into small elements)
6. Define physical constrains (where the part is locked in place)
7. Define which forces the part is exposed to, and the value.
8. Run the analysis

After this different types of data can be seen.

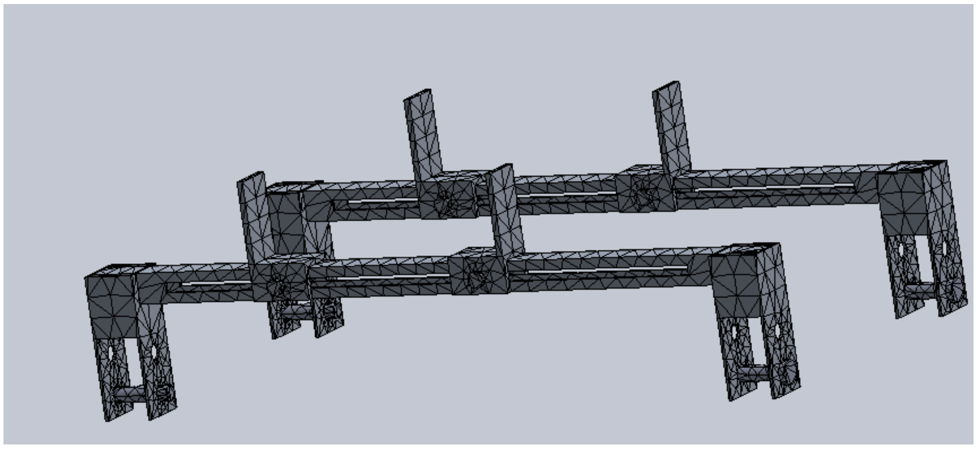


Figure 47: A mesh of the model

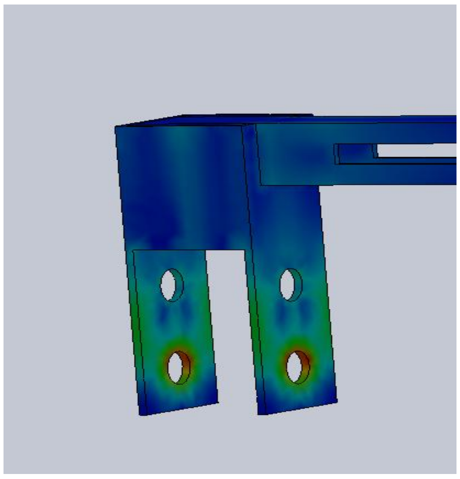


Figure 46: The most critical areas

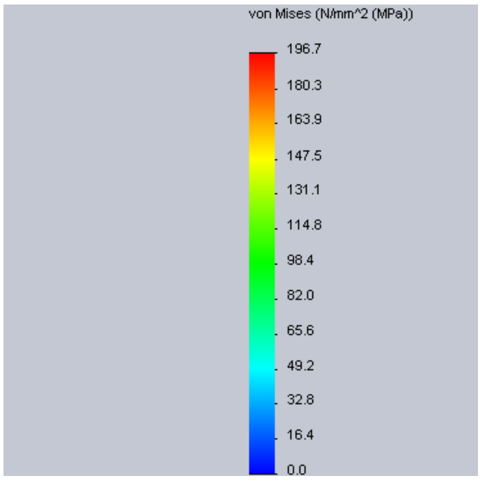


Figure 45 : Von Mises stress (MPa)

The model was calculated using a direct force on 650N (3.5) which was worst case scenario without the springs reducing its speed, and increasing the stoppage time. It was also applied g-forces in the acceleration phase on 10g. All the forces where applied in the direction of the movement to the wagon.

Von Misses

Calculating how much stress the material can withstand was done using Von Misses method. This is a measure of stress intensity for ductile materials, such as aluminum. When a material is exposed for stresses equal to the stress limit, it will start to yield at the location. If the stress intensity is less than the materials yield strength then stresses are within yield and the structure will not break, as this construction.

Factor of safety: $\frac{yield\ \ strengt\ h}{maximum\ \ \ stress} = \frac{275MPa}{196{,}7Mpa} = 1{,}4$

From Figure 46, one can see that the point where the part gets connected to the wagon needs to withstand most stresses, but the material is capable of handling it. After time the material will be weary, so a fittings can help reducing a lot of the stresses at the area.

3.2.10 Final product

To get more use of the materials properties, it was decided to go with a rectangular hollow section tubes.

The new wagon consists of these parts:

Amount	Type	Length	Dimensions
2	square hollow tubes	300mm	15x15x2mm
4	square hollow tubes	48mm	20x20x2mm
2	square hollow tubes	70mm	15x15x2mm
2	rectangular hollow tubes	80mm	30x20x2mm
2	rectangular hollow tubes	60mm	30x20x2mm
4	Plate	95mm	15x3mm
8	Hexagon bolts	30mm	M5
14	Hexagon bolts	30mm	M4
8	Lock nuts		M5
14	Lock nuts		M4
44	Washers		5,3x10mm

The wagon system for the Siren aircraft consists of two main parts. On each of those parts there are added two tubes that hold the aircraft. The tubes are adjustable, to suite other aircrafts. There are also two vertical airframe supports that can be regulated after size of the aircraft. From the FEM analyze, it was found out that most of the force reacting on the wagon parts, where on the

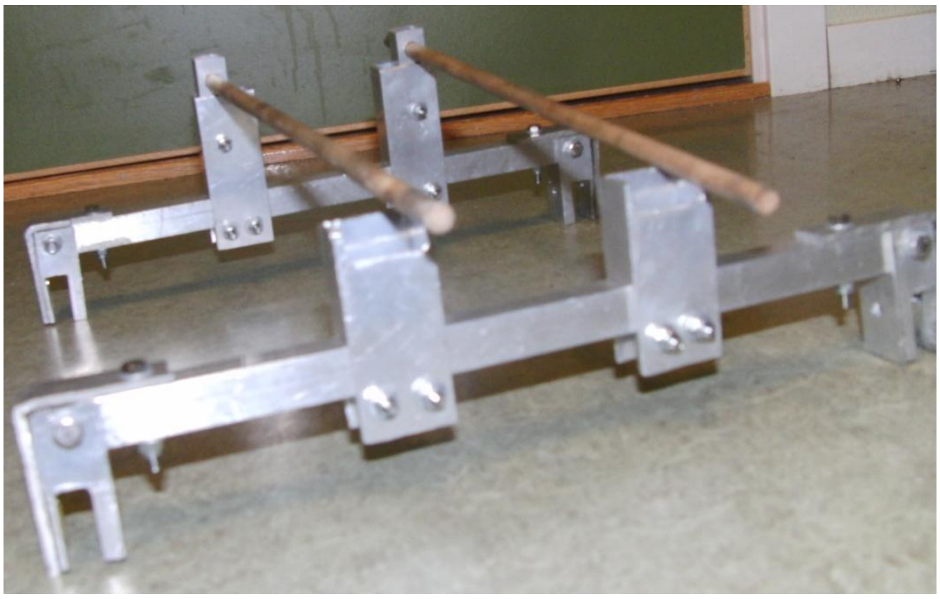


Figure 48: A of the two wagon parts for the Siren airframe

sections, where they got connected to the main wagon frame. To reduce the wear on the parts, fittings to all 4 sides for increasing the strength in the area was added.

3.3 Release mechanism

The release mechanism is the device that holds the wagon in rear position while the bungee cords are being extended. It also releases the wagon when being launched.

The old device was a servo based solution, removing a hook to release the wagon. It locked a bar attached on the wagon and with help from the servo it was designed to remove/open the lock.

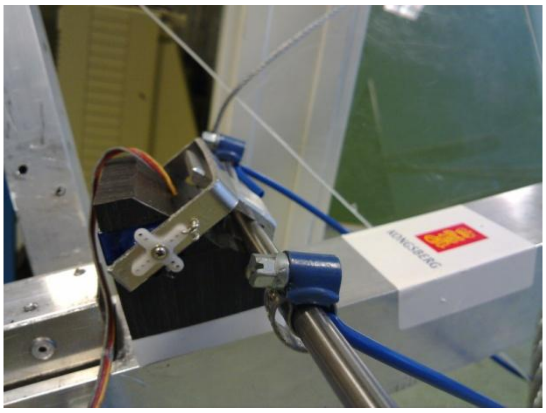


Figure 49: Old release mechanism

When testing the old release mechanism, it did not work. The servo was not strong enough to move the lock pin. It had to be released manually using a rope connected to the lock pin.

3.3.1 New release mechanism design

The release mechanism was also a very unsafe device, because of the possibilities of releasing during stretching the rubber bands. It was therefore decided to make a new release system.

The system requirements for the new release mechanism:

- The mechanism must be strong enough to hold and release the wagon.
- The mechanism must have a safety device for preventing premature launch.

3.3.2 System concepts

Four possible concepts that fulfil our requirements were made-up, with different level of complexity.

Concept 1

A simplified solution based on the existing release mechanism. Use a rope to manually open a hook to release the wagon.

Concept 2

The system is similar to the one that exist, just with a stronger servo and an additional safety device (adding a safety pin)

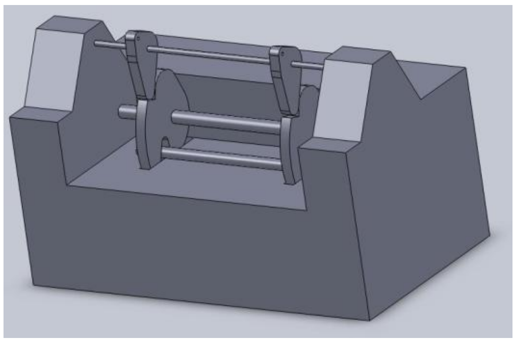


Figure 50: The picture shows a design solution to the release system

Concept 3

This solution is based on the same principle as the one that exist, but an improvement (locking a bar and releasing using a servo). Use a car door latch to hold the wagon and a servo for releasing. In addition add a safety pin.

Concept 4

This is a more complex system. It consists of on permanent magnet and in additional one electro magnet. The system locks the wagon, and is only released when a power source is applied, counteract the existing magnetic field.

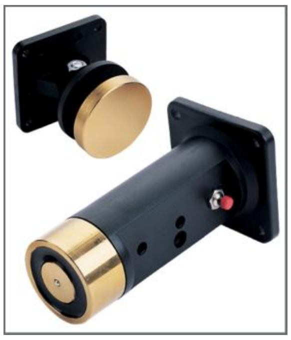


Figure 51 Shows electro/permanent magnets

Solution Choice

The solution chosen was concept 3 containing the car door latch. The door latch has been developed by the car industry over several years, and has become a good solution. The force needed to open the lock is less than the force needing to hold it. That's why it is a good system. In addition to adjust the car door a safety device will be added. This will consist of a pin that locks the wagon when it is placed in the trigger. This needs to be removed before releasing the trigger lock.

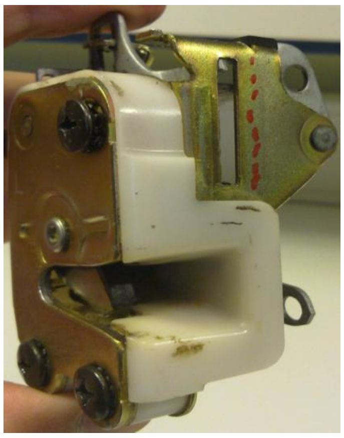


Figure 52: This shows a car door latch

3.3.3 Prototype testing

First the door latch had to be modified so it could fit our system. The door latch had some rods attached to it, and also a lot of unnecessarily plastics. Then it had to be tested to see if it was strong enough to hold the wagon in place, when the bungee cord was extended maximum. It was important that the locking bolt (holding the wagon) did not open before it was allowed to.

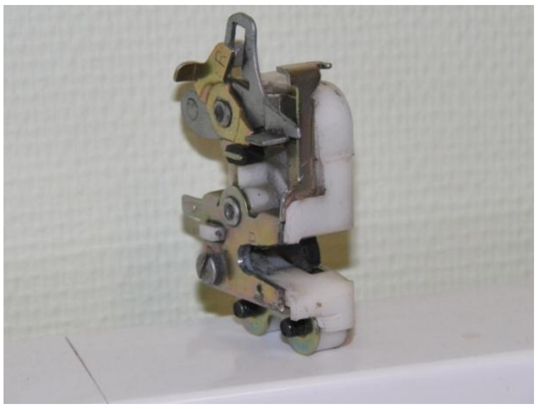


Figure 53: Shows the modified door latch

After it was modified it was installed on the launcher and tested it. The test was satisfying and indicated that it was more than sufficient.

3.3.4 Designing the release mechanism

The new mechanism consists of the car door latch and a servo. The old servo was worn out and too weak. A new stronger servo was added. This was a HPI SF-5 servo with metal gears. It had a torque force of 8,9 kg/cm at 6 volts, opposed to the old which was only 2,1 kg/cm at 5 volts.

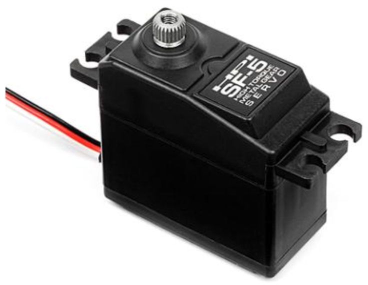


Figure 54: HPI SF-5 servo

The trigger base design

It was designed a frame for assembling the door latch and servo together.

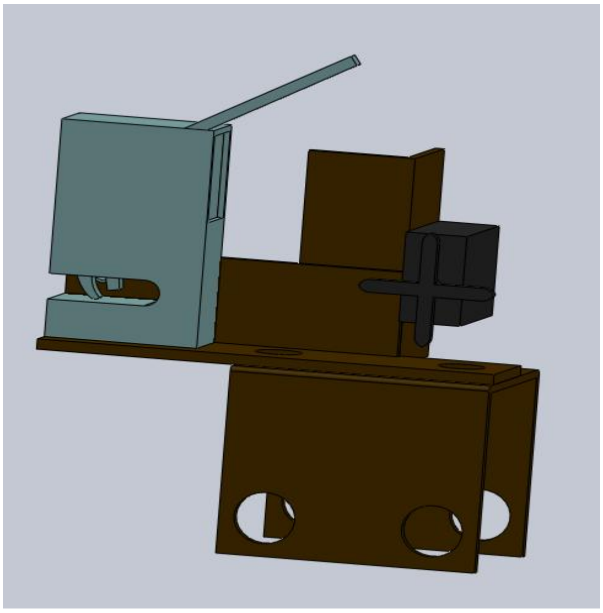


Figure 56: Shows a drawing of the trigger base (brown), with the servo (dark grey) and the door latch (light blue).

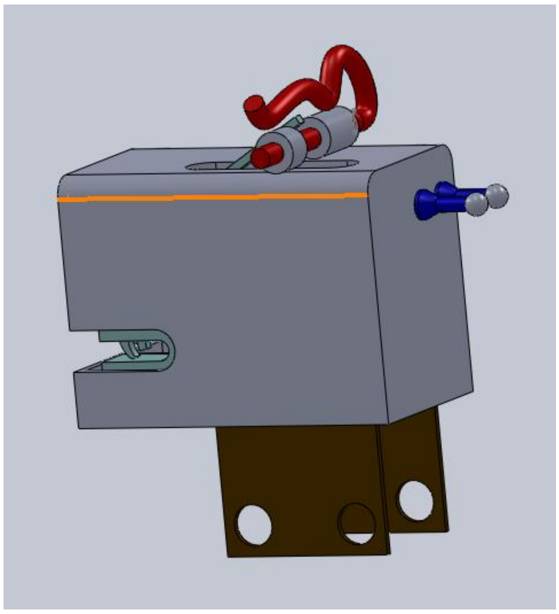


Figure 55 Shows a drawing of the cover around the trigger base system, with the red safety pin

It was also designed so that it could be assembled to the launcher frame. It could also easily be disassembled, so that changes and any errors that may occur could easy be fixed. The release handle on the car door latch had to be extended, for reducing the force needed to trigger using the moment of force.

The release mechanism cover was designed mainly to hide the parts in the release mechanism, and also give the internal components protection. The safety pin is located on top of the cover. The idea is that this will prevent the release mechanism from triggering before it is removed. With this solution the pin denies the release handle from triggering.

3.3.5 Electronics

While operating the release mechanism, an important feature should be that it is reliant. When the operator wishes to release the wagon, he should be certain that the mechanism actually would work. That means; if one trigger system failed, another would be easily available to use. Before describing the different solution designed, it should be explained how the Release Mechanism is made to work.

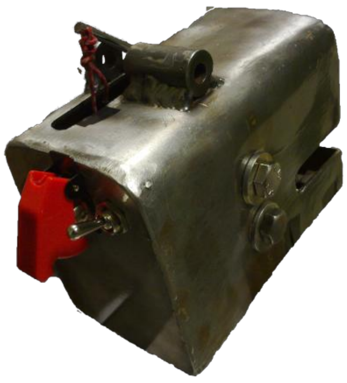
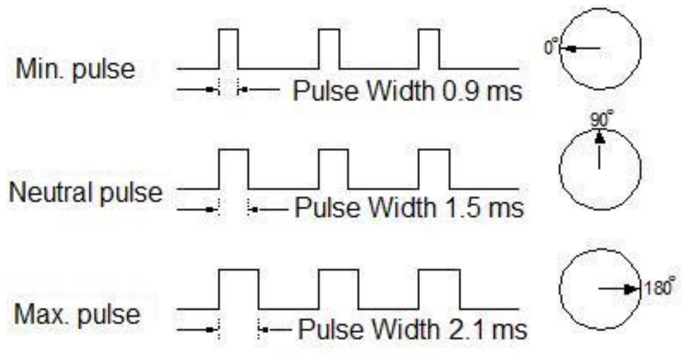


Figure 58: The Release Mechanism with power- and launch signal switches

Figure 57: The servo connector

The servo

As mentioned above, the SF-5 servo was used to release the wagon. This is a servo and uses pulse coded modulated (PCM)-signals to control the position of the motor. The servo has three cords attached to the servo board. Figure 57 shows a servo connector where the red and black carries power and the white carry the pulse signal. The angle of the servo wheel is determined by the duration of the pulse that is applied to the pulse wire. The servo expects to see a pulse about every 20th millisecond. As shown below a pulse with the length of 1.5 milliseconds will turn the motor to neutral state, 0.9 milliseconds will turn the motor to 0 degrees and 2.1 milliseconds will turn to motor to 180 degrees. Since the servo just opens or closes the trigger, the servo receives respectably either a short pulse or a long pulse.



The servo uses a negative feedback system to regulate itself. This will make the servo push on as long as it is not at the right angle. There are therefore two states that one should be aware of: if the servo is stalled or if the servo receives a shorter or longer pulse than required to reach the maximum and minimum it might break.

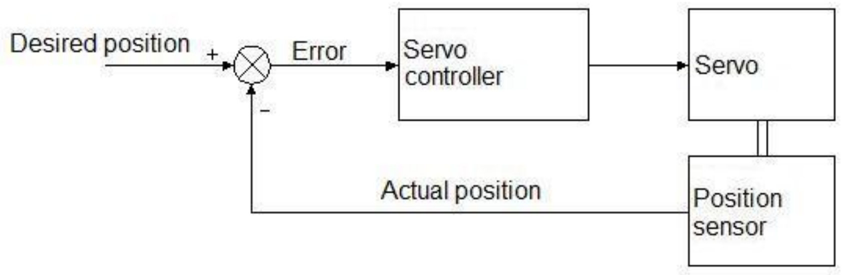


Figure 59: Servo negative back loop

Servo board

To control the servo, the servo circuit board shown in figure 60 is used. An Atmega168 is used to receive data and execute commands. To give the board power, a 5 volt battery regulator was created, see figure. This was attached to the one end and the radio circuit board in the other.

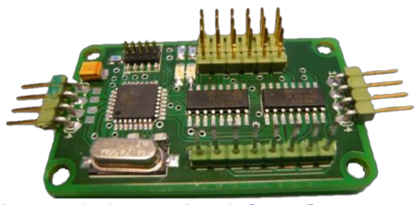


Figure 60: Servo circuit board

The microcontroller is constantly checking if the state of the switch has changed. When this happens, it either closes or opens the servo, depending on the desired outcome. The same goes for signals from the radio frequency board. The servo is connected to the servo connection pins on the far right. Current is drawn directly from power supply, which never puts the microcontroller in danger, even when servo is under heavy load. To be able to use the SERVO_0 to drive the servo, a modification has been made on the back of the board connecting MUX_SELECT to V_{CC}.

Servo circuit maximum ratings calculations

Our calculations on how much current that can pass through the servo power was important to know if the SF-5 servo could be used. This was not specified in any of the reports we have used as reference. Therefore we calculated by measuring by hand and finding the smallest standard for thickness. The maximum rated current that can pass through the power wire depends on the current, thickness and the trace width. The trace width was measured to be 20mm. Since the thickness was unknown, we used the thinnest standard for printed circuit boards, 17µm. Using the web calculator [10] the highest current flow possible with these values is 1170 mA.

Radio board

To receive signals from the Ground Station, the radio board shown in figure 61 was used. The board is connected to the servo board from where it is powered. This board's RC1240 module uses a planar antenna. The module sends received data to the Atmega16 where the data is processed. The filtered signals are sent through the I^2C to the servo board.

The radio circuit board is placed inside the release mechanism cover. This can be done because steel is not nearly as conductive as for example copper. The cover will work to some degree as a faraday cage, but with a working range of at least 130 meters[15] from the Ground Station, which is highly acceptable. In the future, the radio board can read from several other sensors like an anemometer or any other sensor that might be desired. Important procedures when programming the Atmega16 is described in 3.3.7.

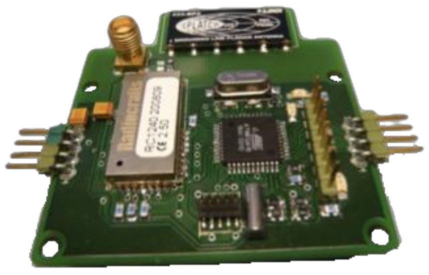


Figure 61: Radio circuit board

Power supply

The release mechanism requires 5 volts to function. As a result, it was created a 7805 linear regulator circuit to bring down the voltage, shown in figure 62. This regulator consumes the excess voltage above 5 volts. Any battery between 5 volts and 35 volts can therefore be used, but be aware; the higher the voltage, the more current is wasted and battery life shortened. The formula below shows the current drawn from the battery using a 7805.

$$I_{battery} \approx \frac{P_{circuit}}{V_{in} - V_{out}}$$

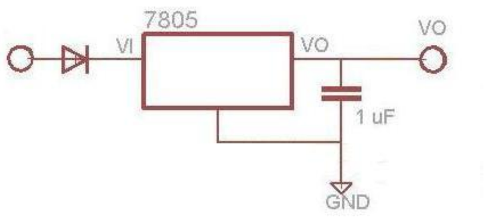


Figure 62: Release Mechanism voltage supply

A battery with a voltage just above 5 volts is therefore preferred. During a test a 9 volt battery[16] was tried. It was very noticeable how weak the servo became. While the wagon was under high tension from the bungee cords, the trigger

[15] Tested the range of the release mechanism. Connection up to at least 130 meters.
[16] 9 volt Energizer Industrial battery

could not release the wagon. When connected to a 6 volt NiMH RC battery pack the servo acted much stronger. It then released the wagon without difficulty, this even under the highest possible strain from the launcher's bungee cords. Under this high load on the servo the release mechanism has a peak current of 1100 mA. Looking at the calculations done [10], one can see that the servo can be used. The release mechanism draws 80 mA in standby mode.

3.3.6 The different ways of releasing the wagon

Before building a system, different concepts that each helped the system become more failsafe had to be designed. Each of these concepts is using the same servo board and driver to run the servo. The code is described in chapter 3.3.5.

Concept of using the aircraft radio board to trigger the Wagon release

When planning on using one of the radio boards on the Ground Station, there was a shortage of radio boards. A solution of carrying the trigger signal from the radio module placed in the aircraft using a cord was made. This would use three conductive patches underneath the aircraft. Two of the patches would carry the I^{2}C protocol and one patch to set a common ground. The three patches could be placed underneath the wings. The radio circuit board could then send data down the wing, through the patches, onto another cord and into the servo board releasing the Wagon.

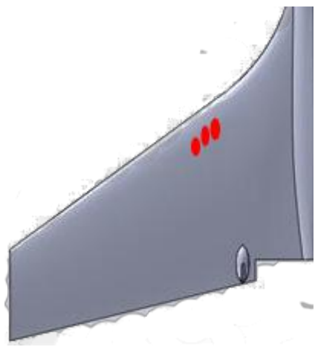


Figure 63: This shows an example on how the conduction patches may be placed underneath the Stryker

Using switches to trigger the Wagon release

Using a switch to trigger the launch on the Release Mechanism would permit the Launcher to be used without setting up a connection with the Ground Station. Using a second switch to turn the power on also allows a battery to be connected at all times without the need of being removed to save the battery after every use.

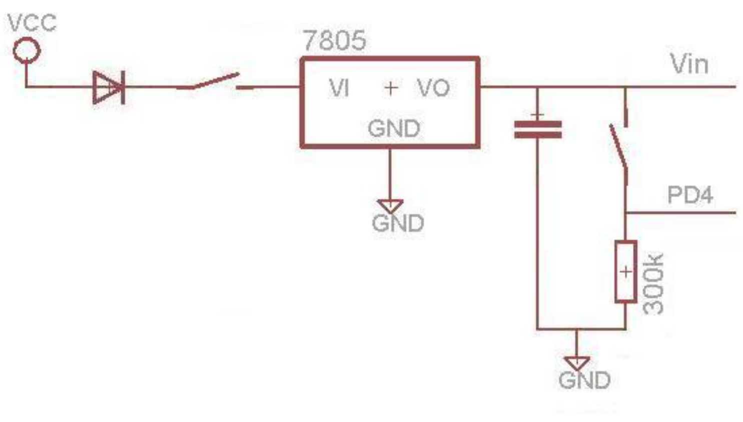


Figure 64: Release mechanism with switches

Figure 64 shows the power circuit in the Release Mechanism. V_{in} is the power to the circuit boards. PD4 is the connection to PIND4 on the Atmega168 placed on the servo board. When the trigger switch is used the microcontroller receives a change at the specified pin, and then drives the servo.

```
x=PIND&0b00000010;
if(oldx!=x && x==0b00000010){
        oldx=x;
        _delay_ms(2000);
        ServoDriver(OPEN);}
else if(gammelx!=x && x==0b00000000){
        oldx=x;
        ServoDriver(CLOSED);}
```

Using the aircraft's radio board to trigger the Wagon release

The Ground Station receives constantly data from one address on the radio system. At the same time, it is capable of sending data out using another address. Using the address of the radio board in the Release Mechanism enables the Ground Station to send a trigger signal. The radio board picks this up and sends the corresponding go signal to the servo board.

```
while(1){...
switch(inputdata[0]){
        case 0:ServoDriver(CLOSED);break;
        case 1:ServoDriver(OPEN);break;
        case 2:ServoDriver(OPEN);
                _delay_ms(1000);
                ServoDriver(CLOSED);
                break;
        default : for(int i=0;i<40;i++){PORTC^=(1<<PC0);_delay_ms(25);}
        break;}
}...
```

The Release Mechanism can be triggered using either the radio link with the Ground Station or by using the switches on the cover. This makes the Release Mechanism easy to use. If anything would go wrong with the electronics, one still has the option of manually pressing the handle on the top of the cover down to release the wagon.

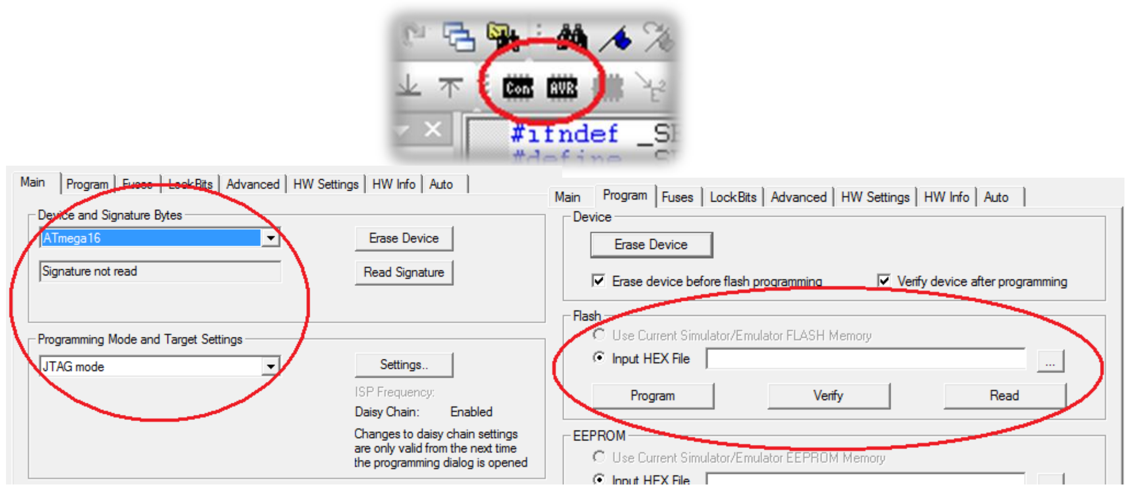
3.3.7 How to

In this section will describe some basic features that might come in handy if you plan on digging deeper into the project groundwork.

Program microcontroller

All the microcontrollers used have the same tiny headers to connect the programming tools. The Atmega16 requires JTAG connection to the board, while the Atmega168 requires ISP. A JTAGICE from Atmel for the Atmega16 was used. Connect this onto the programming adaptor and place this onto the 10-pin header. For the Atmega168 an AVRISPmkII was used. Connect this to the adaptor and place it on the 10-pin header.

Open AVRStudio (version 4.15 was used for this project). Be sure the correct device corresponds to the microcontroller about to be programmed. Press either button as shown below. Select the programming tool and microcontroller. A new widow will appear. Under the "main" tab select the mode (JTAG or ISP). Under the "program" tab select the hex file to be loaded, and then press the "Program" button.



Device and Signature Bytes
ATmega16
Signature not read

Programming Mode and Target Settings
JTAG mode

ISP Frequency:
Daisy Chain: Enabled
Changes to daisy chain settings are only valid from the next time the programming dialog is opened

Device
☑ Erase device before flash programming ☑ Verify device after programming

Flash
○ Use Current Simulator/Emulator FLASH Memory
● Input HEX File

EEPROM
○ Use Current Simulator/Emulator EEPROM Memory
● Input HEX File

Configure the RC1240 module

Configuring the RC1240 incorrect may damage the EEPROM. Follow these instructions to avoid this.

To configure the development-kit, use Radiocrafts' Configuration and Communication Tool. If not using hardware handshaking, the jumpers should be set as shown in figure 65.

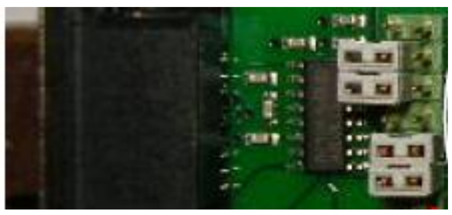


Figure 65: RC1240 DK jumper placement

Making the radio modules enter configuration mode on every startup may be harmful, and should be prevented. To enter EEPROM, send the character M to the module and wait for reply prompt. Important: always end the EEPROM configuration with '0xff'. For configuring commands, see datasheet. The code for configuring the module is included in rc1240.c, but out commented for safety.

Use UART

To use UART, Atmel's own c- and h- files for programming microcontroller in c was used. These files can be found on the CD. This approach works well when sending one char at a time of an char array, but when handling more information the printf() function was preferred. This still requires the "stdout = uart" line. Be aware that the Atmega16 and the Atmega168 are different when it comes to Interrupt Routine Service.

3.3.8 Troubleshooting

This section is mainly to help future work on the Local Hawk system. The list of problems we encountered and spent a lot of time doing while working on the project. The list of obstacles we had to overcome grew to a respectable size as the work progressed. Here are some of the problems we stumbled upon.

The RC1240 module responds but does not send data

The EEPROM (non-volatile memory) may be corrupt. The memory must be configured using a list of values. Use the readme on the CD. To fix the development-kit, use Radiocrafts' CCT. To fix modules on radio board, use

The RC1240 module does not respond to request

If working on the development-kit, make sure the jumpers are placed correctly. If working on the radio board, make sure the power to the module is enabled.

The UART/radio module sends corrupt data

If one receives faulty symbols like question marks or music notes, some library files need to be included. To do so; do the following:

1. Access the configuration options by pressing the Project tab.
2. Under Libraries you must include "libm.a" and "libprintf_flt.a".
3. Under Custom Options, select [Linker Option] and type: "-Wl,-u,vfprintf"

This should fix the problem.

The microcontroller does not respond after having been programmed

There is probably something wrong in the configuration options. Often this comes from one of two things:

1. AVR Studio is configured to program a different Atmega model. Check that the model described in the very bottom is matching the microcontroller being programmed.
2. The CPU frequency, which is the same as the crystal. All the circuit board we are operating uses a 14.745600 MHz crystal. To define this, if not already defined, is done in

For both cases, access the configuration options by pressing the Project tab.

3.3.9 The Final Product

With the accurate drawings from SolidWorks (Figure 66 and Figure 67) the building was a small process. First a base for the trigger, with clamping plates for suiting the launcher frame had to be made. Plates and beams were welded together. We also had to add new support plates on the launcher. The door latch and servo where placed so that the centre of the release bar and the wheel on the servo was directly above each other.

For reducing the force needed to release, a bar was added to extent the release bar which was only 20mm long to 110mm. Reducing the force by 6 times.

With this the force needed to release is 3,53N (0.36kg), where the servo can handle force up to 43,65N (divided the real force by 2 because of the pulley). The bar became connected to the pulley on the servo with a strong rope, which rolling in the rope.

A release mechanism cover was then built, by bending a plate into a "U" form and welded two plates on each side. Two pipes where adjusted and welded on top for the safety pin, and two gaps where made. One for the extended release bar and one for the wagon stick.

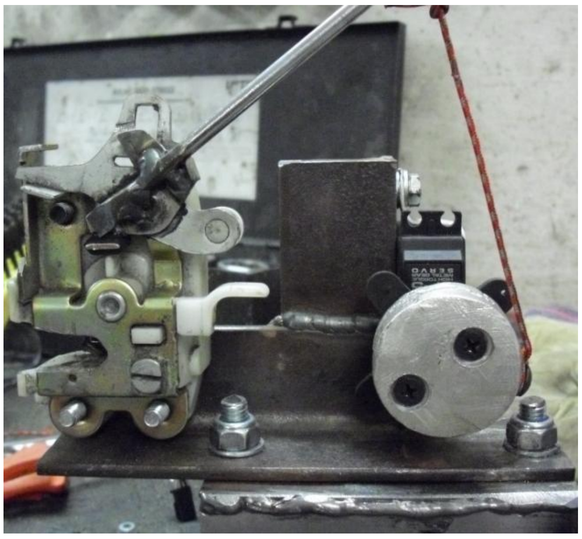


Figure 67: Shows the whole release mechanism on the launcher

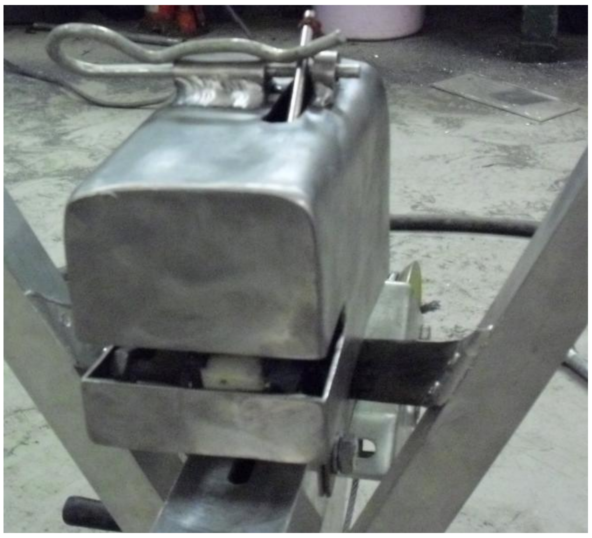


Figure 66: Shows the release mechanism

Triggering the release can now be done in several ways. Since the release mechanism is wirelessly connected to the Ground Station there the aircraft can be sent on its way from pressing a button in the Ground station. Because of the great range of the RC1240 module the Ground Station can be placed far from the launcher, still able to trigger. Additionally, the release mechanism can be triggered locally by using the switches on the back of the launcher. This makes the system both more modular and failsafe. It is in fact not vital to set up a connection with the Ground Station if one experience problems setting up a connection to the release mechanism. If desired to reduce power consumption, the RF board can then simply be removed. If the electronics fails completely because of flat battery etc., the wagon can still be released by pressing down the bar on the top of the cover, which will mechanically trigger the wagon.

3.4 Further improvements:

This section contains suggestions of improvements to the completed launcher system:

Launcher

- When operation the launcher you see think of possible thing that can happen regarding safety. The wires can snap because of the wire clamps. Wire clamps are not a god solution, and should be changed to improve the safety.
- Change the wires to reduce friction, and also increase the safety (maybe rails).
- It is very heavy to extend the rubber bands using the existing winch. Implementing a stronger and better winch would do the job easier. For making the system more autonomous an electric winch would work.
- Reduce the weight on the wagon to get higher results (velocity). This can be done by changing the metals or finding another solution.
- Improve the wagon for a more stable Siren launch.
- The springs reduce the impact force on the wagon, but they are not strong enough. They could be stronger.
- Change the rubber bands to get a system that performs better.
- Change the pulley system for reducing the friction(implement bearings)

Release mechanism

- The release mechanism could have an indicator that shows when it releases. It has a delay, but nothing that shows: how much time left before launch.
- The release mechanism could benefit from implementing a force measurement tool.
- The electronics are vulnerable when mounting the cover. An idea would be to implement rails that would lead the cover straight to the correct position.
- The battery is connected to the circuit through two identical pins. To make the battery failsafe it is necessary to implement a solution. Using a futaba jack or three pins with ground, positive, ground would make it impossible to mistakenly connect the wrong poles.

4 Chapter 4 Test document

4.1 TST-COR – Core Application

GRS-COR

Requirements

REQ-COR-1		The Core application design
Requirement	**Pri**	**Description**
REQ-COR-1.1	1	The Core shall be written in C++
REQ-COR-1.2	1	The core application shall not be dependent on any other part of the ground station software. *This means that the core shall continue to run even if the GPS and/or the Wireless link is disconnected and any software crashes, other software includes GPS, wireless link drivers and/or GUI.*
REQ-COR-1.3	1	There shall be defined a interface between the peripheral circuit and the core
REQ-COR-1.4	2	There shall be a socket interface between GUI and the core

Test

TST-COR-1.1

Disconnect the GUI from its socket end, if the core continues to run the test is considered a success.

TST-COR-1.2

Run the following tests for the GPS: TST-GPS-2 and Wireless: TST-WRL-3, if these tests are passed this test is also considered a success.

TST-COR-1.3

Send incorrect/destructive information in to the core, if the core handles the information and continues to run the test is considered a success.

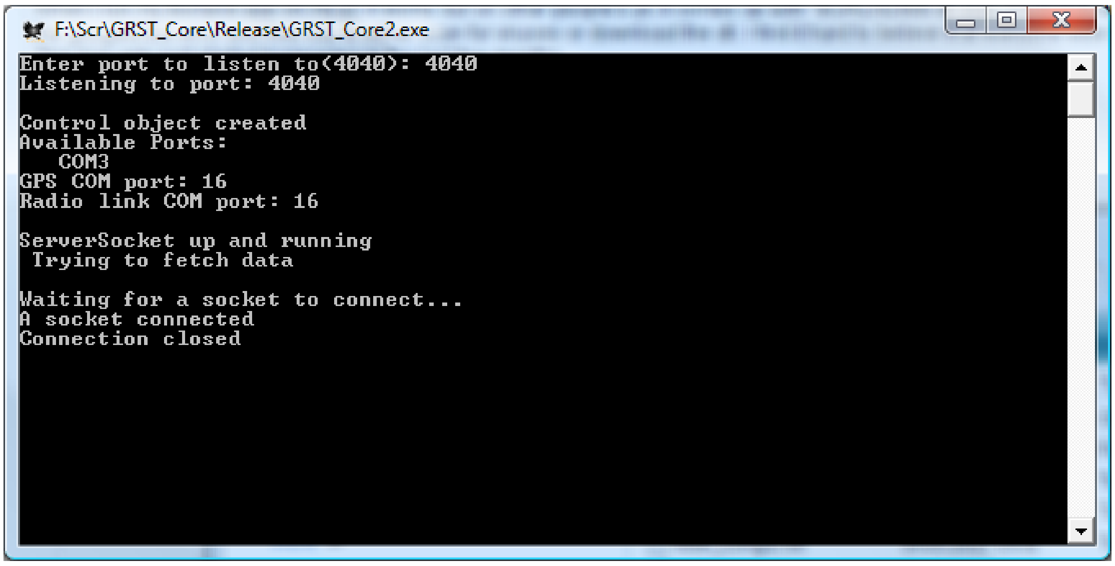
TST-COR-2.1

Run simulated information trough the interface and if the information is correctly formatted after passing through the interface the test is considered a success.

Result

REQ-COR-1.2	1	The core application shall not be dependent on any other part of the ground station software.

If the GUI is closed or crashes the core displays a message stating that a socket connection is closed:



F:\Scr\GRST_Core\Release\GRST_Core2.exe

```
Enter port to listen to(4040): 4040
Listening to port: 4040

Control object created
Available Ports:
   COM3
GPS COM port: 16
Radio link COM port: 16

ServerSocket up and running
 Trying to fetch data

Waiting for a socket to connect...
A socket connected
Connection closed
```

REQ-COR-1.3	1	There shall be defined a interface between the peripheral circuit and the core
REQ-COR-1.4	2	There shall be a socket interface between GUI and the core

The interfaces have been defined in Interface design document.

4.2 TST-GUI – Graphical User Interface

GRS-GUI

Requirements

REQ-GRS-1	The GUI design	
Requirement	**Pri**	**Description**
REQ-GUI-1.1	1	It contains a map; the map shall enable the user to set waypoint for the aircraft.
REQ-GUI-1.3	2	It contains a console; the console shall give messages and error messages as well as take commands from the user.
REQ-GUI-1.4	2	It contains a graph that can show a statistics of different logged information chosen by the user.
REQ-GUI-1.5	2	It is divided into tabs for each task; *For example map, video, console will have its own tab.*
REQ-GUI-1.6	3	It contains possibilities for user defined settings; *Like the user shall be able to choose whether the map should follow the aircraft or not.*

4.3 Test

4.3.1 TST-GUI-1.1

Run simulated messages and error trough the system, disconnect the GPS and the wireless. If the correct messages are displayed, the test is considered a success.

4.3.2 TST-GUI-1.2

Type inn commands and read the output in the other end of the system. If the commands match, the test is considered a success.

4.3.3 TST-GUI-2.1

Send simulated information, and if the information displayed is correct, the test is considered a success.

Result

REQ-GUI-1.1	1	It contains a map; the map shall enable the user to set waypoint for the aircraft.
REQ-GUI-1.3	2	It contains a console; the console shall give messages and error messages as well as take commands from the user.
REQ-GUI-1.5	2	It is divided into tabs for each task. *For example map, video, console will have its own tab.*

All these requirements were maintained:

REQ-GUI-1.4	2	It contains a graph that can show a statistics of different logged information chosen by the user.

This requirement was down prioritized, because of the creation of the log converter program was created, this enables all the log files to be viewed in ether eagle three or matlab which enables many more complex graphs views and comparisons.

REQ-GUI-1.6	3	It contains possibilities for user defined settings; *Like the user shall be able to choose whether the map should follow the aircraft or not.*

This is requirement was not fulfilled because there was no need, sins the map implemented is static.

4.4 TST-GPS – Global Positioning system

REQ-GPS-1 Ground stations GPS position		
Requirement	**Pri**	**Description**
REQ-GPS-1.1	1	The ground stations GPS position (Coordinates) shall be available to the ground station software.
REQ-GPS-1.2	1	The position data available shall be at least the NMEA 0183$GPRMA/B data *Longitude, latitude, height,*

REQ-GPS-2. GPS failsafe		
Requirement	**Pri**	**Description**
REQ-GPS-2.1	1	The ground station shall handle a GPS disconnect without freezing, crashing or otherwise inhibit the continued operation of the core application.
REQ-GPS-2.2	1	The loss of satellites shall not bring the entire system down
REQ-GPS-2.3	2	Provide the code app with a warning/indication if all satellite connections are lost

Tests

TST-GPS-1.1
Connect to the GPS the ground station and see if the ground station displays the GPS position. The test is considered a success if the position is showed correct.

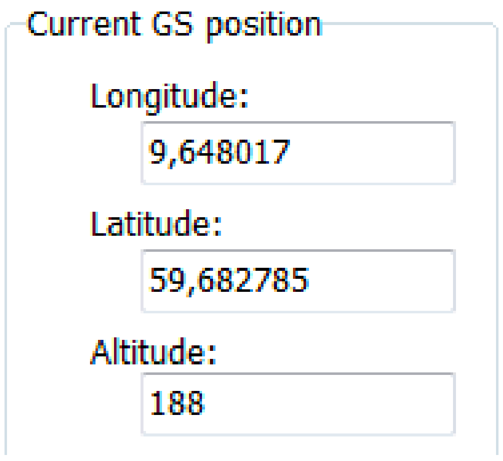
TST-GPS-2.1
Connect up the GPS and wait for satellite lock, then disconnect the cable between the GPS antenna module and the circuit board. If the core application still operates, the test is considered a success.

TST-GPS-2.2
Connect up everything as in 2.1, and then provoke a satellite loss. If the core application still operates, the test is considered a success. If the ground station also shows an indication of satellite loss the REQ-GPS-2.3 is also a success.

Result

REQ-GPS-1.1	1	The Ground Station's GPS position (Coordinates) shall be available to the ground station software.

The coordinates are shown several places in the GUI


Current GS position

Longitude: 9,648017

Latitude: 59,682785

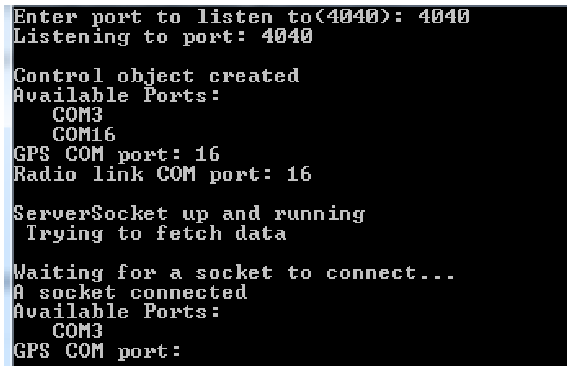
Altitude: 188

REQ-GPS-1.2	1	The position data available shall be at least the NMEA 0183 $GPRMA/B data *Longitude, latitude, height,*

The GPS module did not support RMA (Recommended Minimum A) or RMB but it did support RMC appendix 1, which was used

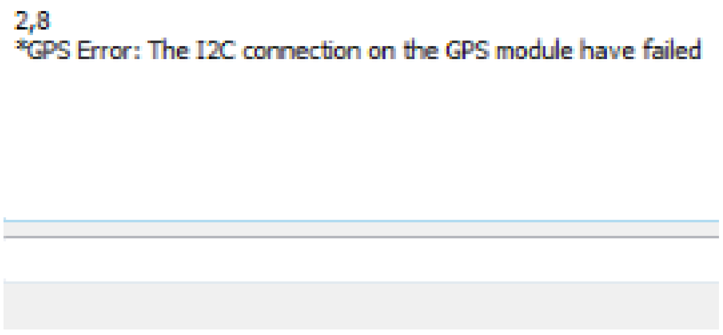
REQ-GPS-2.1	1	The Ground Station shall handle a GPS disconnect without freezing, crashing or otherwise inhibit the continued operation of the core application.

This requirement has not been fully reached, A GSP disconnect does not freeze or crash the system. It does prompt the user that it cannot find a GPS module connected to the specified COM port and ask the user to input new port number. Doing so pauses the Core application and prevents it from reading the radio link COM port or sending data through the socket since the Core is a single thread application. This can solved by making it a multithread process, which is described in future outlook.



```
Enter port to listen to(4040): 4040
Listening to port: 4040

Control object created
Available Ports:
   COM3
   COM16
GPS COM port: 16
Radio link COM port: 16

ServerSocket up and running
 Trying to fetch data

Waiting for a socket to connect...
A socket connected
Available Ports:
   COM3
GPS COM port:
```

Since a solution that has a I^{2}C connection was chosen, this is a possible weak spot and can fail. An error message was created to tell the user if it is the I^{2}C that is causing the problems



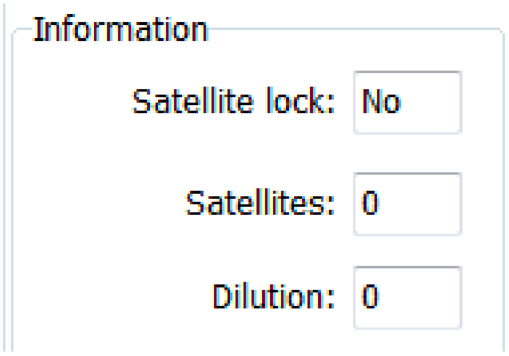
```
2,8
*GPS Error: The I2C connection on the GPS module have failed
```

REQ-GPS-2.2	1	The loss of satellites shall not bring the entire system down

In the event that the GPS module loses all the satellites the system recognizes that this has happened, It then keeps the last known position as the valid position off the Ground Station until the GPS reacquires satellites.

REQ-GPS-2.3	2	Provide the core application with a warning/indication if all satellite connections are lost

This information is relayed to the GUI and displayed there it is altos used in the accomplishment of the requirement above.



Satellite lock: No

Satellites: 0

Dilution: 0

4.5 TST-WRL – Wireless link

GRS-WRL

Requirements.

REQ-WRL-1. Communication		
Requirement	**Pri**	**Description**
REQ-WRL-1.1	1	The wireless link needs to communicate with the new ground station core application now written in C++
REQ-WRL-1.2	1	Contain CRC-16 error detection.

REQ-WRL-2. Package flow stability		
Requirement	**Pri**	**Description**
REQ-WRL-2.1	1	The microcontroller shall not send a package while receiving and vice versa.

REQ-WRL-3. Failsafe		
Requirement	**Pri**	**Description**
REQ-WRL-3.1	2	The system shall become aware if connection to aircraft is lost.
REQ-WRL-3.2	3	The signal strength shall be available.
REQ-WRL-3.3	2	An error in the wireless shall not freeze, crash or otherwise inhibit the continued operation of the core application.

Test

TST-WRL-1.1

Load the code onto the microcontroller and connect the antenna to the board together with the primary microcontroller. Try to send information through the wireless link. If the input data is equal to the received data, then the test is considered a success.

TST-WRL-2.1

Try and transmit data from two radio modules simultaneously[17]. These two shall have established a connection in advance. If this is prevented so that only one transmits at a time, then the test is considered a success.

TST-WRL-3.1

Hopefully, the GUI is able to show information about connection status. If not, a test to test this function will be too time consuming and will not be required.

1. Establish a connection with two radio modules (ground station and the aircraft).

[17] One should be the ground station module, while the other one is LocalHawk, or similar.

2. Find a way to weaken the signal (either by distance, obstacle, Faraday cage, remove the antenna, etc.).

If the GUI shows a notification of lost contact, then the test is considered a success.

Results

TST-WRL-1:

Using the printf()-function on the microcontroller the signal is sent through the RC1240 modules on both sides and into the core. When the data sent corresponds to the data expected (GPS data struct) the core receives this and visualizes it in the GUI.

The RC1240 modules contain CRC-16. Additional error correction is done in the Core->Error Check.

Requirement passed.

TST-WRL-2:

Since the aircraft only transmits and the Receive Mechanism only receives, they do not have a problem as described. The Core receives once per second. When data is sent to the Release Mechanism, it simply does not read in the meanwhile. Any lost packages will be noticed and the Core moves on.

Requirement passed.

TST-WRL-3.1:

This does not happen. The Core receives nothing and skips this process.

Requirement passed.

TST-WRL-3.2:

There is no active function to show when connection is lost, but the GUI updates the aircraft's GPS information when there is a connection. Therefore, the operator can see when the connection is lost.

Requirement partly passed.

TST-WRL-3.3:

A function to read the signal strength can be implemented sending a single char to the RC1240 and waiting for a value indicating this. Because of time issues, this requirement was not completed.

4.6 TST-WGN – Wagon system

LSH-WGN

Requirements

REQ-WGN-1. The Wagon		
Requirement	**Pri**	**Description**
REQ-WGN-1.1	1	The Wagon system shall manage to launch the two different airframes (Stryker and Siren) into stable flight.
REQ-WGN-1.2	1	The Wagon system shall manage to launch the two airframes with or without the propeller running.

REQ-WGN-2. Design requirements		
Requirement	**Pri**	**Description**
REQ-WGN-2.4	2	For obtaining the required takeoff speed (approximately 10 $^{m}/_{s}$) *The length of the wagon shall be limited to max 400mm.*

Test

For launching the Siren aircraft a new made part was made, to attach on the old wagon. Before we could try to launch the aircraft, test had to med made so it was certain that the wagons initial velocity was high enough. Both requirements were tested simultaneously.

TST-WGN-1.1-1.2 Pre-test

The wagon test is executed with weights as a replacement for the aircraft. It is also use a high-speed camera for collecting the data. If the wagon has a velocity above 10 m/s when it hit the springs, the test is considered a success.

TST-WGN-1.1-1.2

When it where certain that the launcher would give the aircraft enough take-off speed the final test could be performed. Set up the launcher on a model aircraft strip, for having a secure area. If the Wagon launches the aircraft into stable flight the test is considered a success.

Results

REQ-WGN-1.1	1	The Wagon system must manage to launch the two different airframes (Stryker and Siren) into stable flight.
REQ-WGN-1.2	1	The Wagon system must manage to launch the two airframes with or without the propeller running.

The pre test of the wagon for the Siren indicated that the aircraft would get the required initial velocity to get into stable flight. The speed on the wagon (with weights) was 11, 4 m/s, while the minimum require was 10 m/s.

The first launch for the Siren aircraft was very unstable. Right after the plane had left the wagon it started to fall down, before it got manoeuvred up again by the operator. When analysing the launch sequence it was found that the problem was because of the old wagons behaviour. When the wagon hits the front frame on the launcher, it starts to tip forward. The outcome of this was that the back frame on the aircraft got pushed upwards. This results in a wrong angle of attack on the aircraft.

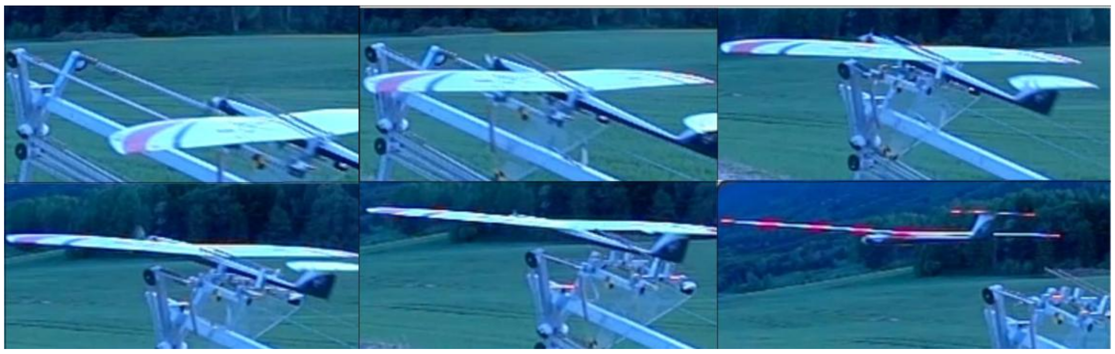


Figure 68: Foto series of the Siren launch.

The aircraft did get into the air, but not into stable flight. The reason this happens with the Siren and not the Stryker is the long tail. The Stryker gets airborne before the wagons tips forward, while the Siren`s tail still is on the wagon. The test was not considered a success because the wagon push the aircrafts tail vertically, changing the angel of attack.

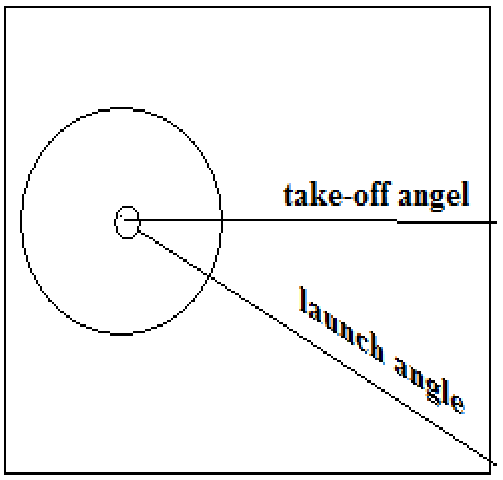


Figure 69: Show the aircraft change in angel

4.7 TST-TRG – Release mechanism

Requirements

REQ-TRG-1. Release mechanism force		
Requirement	**Pri**	**Description**
REQ-TRG-1.1	1	The release mechanism must be strong enough to hold the wagon in lock position, when max force (from the bungee cords)
REQ-TRG-1.2	1	The release mechanism must be strong enough to release the wagon immediately after triggered. *A very important thing regarding safety is to prevent premature launch.*

REQ-TRG-2 Safety		
Requirement	**Pri**	**Description**
REQ-TRG-2.1	1	The release mechanism shall have a safety device for preventing premature launch.

Test

TST-TRG-1.1-1.2

The wagon will be pulled back and locked into the release mechanism. An operator uses the winch to extend the bungee cords to the maximum[18]. This will create max force from the bungee cords on the trigger. When area is safe, an operator triggers the release. This procedure must be done two times, to see if the system behaves the same. If the release mechanism is capable of holding maximum force, and then release, the test is considered successful.

TST-WGN-TRG-2-1

Extend the bungee cords to the maximum with the wagon locked in the release mechanism. When max force from the bungee cords is gained, release of the trigger without releasing the safety device can be done. If the safety device is strong enough to hold the wagon, the test is considered a success.

TST-WGN-TRG-2-2

Set up the system done in 1.1-1.2. Use the electric device to trigger the wagon however this is done. If the wagon releases immediately, the test is considered a success.

[18] Maximum is in this case when the wire is winded back as far as possible

Results

REQ-TRG-1:

The Release mechanism was strong enough to hold the force when the bungee cords were extended maximum. It was also capable of release when ready.

REQ-TRG-2.1:

The release mechanism contains a safety pin that makes sure that the wagon is kept in place, until it is ready to launch. The safety pin is strong enough to hold the release pin.

REQ-TRG-2.2:

The Release Mechanism is equipped with switches to trigger the wagon. We connected the battery and turned the electronics on. After triggering the Release Mechanism it took about 2 seconds before the wagon was released. This is because of the safety delay in the servo software.

Chapter 5

References

Reference

[1] Great plane's Siren aircraft
http://www.electrifly.com/largeelectrics/gpma1065.html

[2] Parkzone's Stryker F27c
http://www.parkzone.com/Products/Default.aspx?ProdID=PKZ4275

[3] Eric S. Raymond. NMEA Revealed v2.3, Mar, 2010

http://gpsd.berlios.de/NMEA.txt

[4] SerialPort class library

http://msdn.microsoft.com/en-us/library/system.io.ports.serialport.aspx

[5] Marshalling managed code

http://msdn.microsoft.com/en-us/library/bb384865.aspx

[6] OpenStreetMap

http://www.openstreetmap.org/

[7] Vector class library

http://www.cplusplus.com/reference/stl/vector/

[8] Ed Williams. Aviation Formulary V1.44 Nov 11, 2008
http://williams.best.vwh.net/avform.htm#Dist

[9] Roy Woodward. Aluminium and Aluminium Alloys - Apr 24, 2001

http://www.azom.com/details.asp?articleid=310

[10] Printed Circuit Board maximum current calculator

http://www.circuitcalculator.com/wordpress/2006/01/31/pcb-trace-width-calculator/

5 Chapter 5

Appendices

5.1 Appendix 1

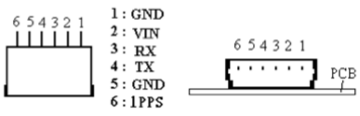
The GPS that is being used is the GlobeSat EM 406 (See picture)

Dynamic conditions
Altitude: 18000 m
Velocity: 515 m/s
Acceleration: 4g
Jerk: 20m/s^3

Protocol

Refresh rate:	1 Hz
Baud rate:	4,800 bps
Electrical level:	TTL level,
Output voltage level:	0V ~ 2.85V
Output Message:	NMEA 0183 GGA, GSA, GSV, RMC, VTG, GLL see [3] for more information.

Power:
4.5V ~6.5 DC input power
Power consumption 44 mA



5.2 Appendix 2

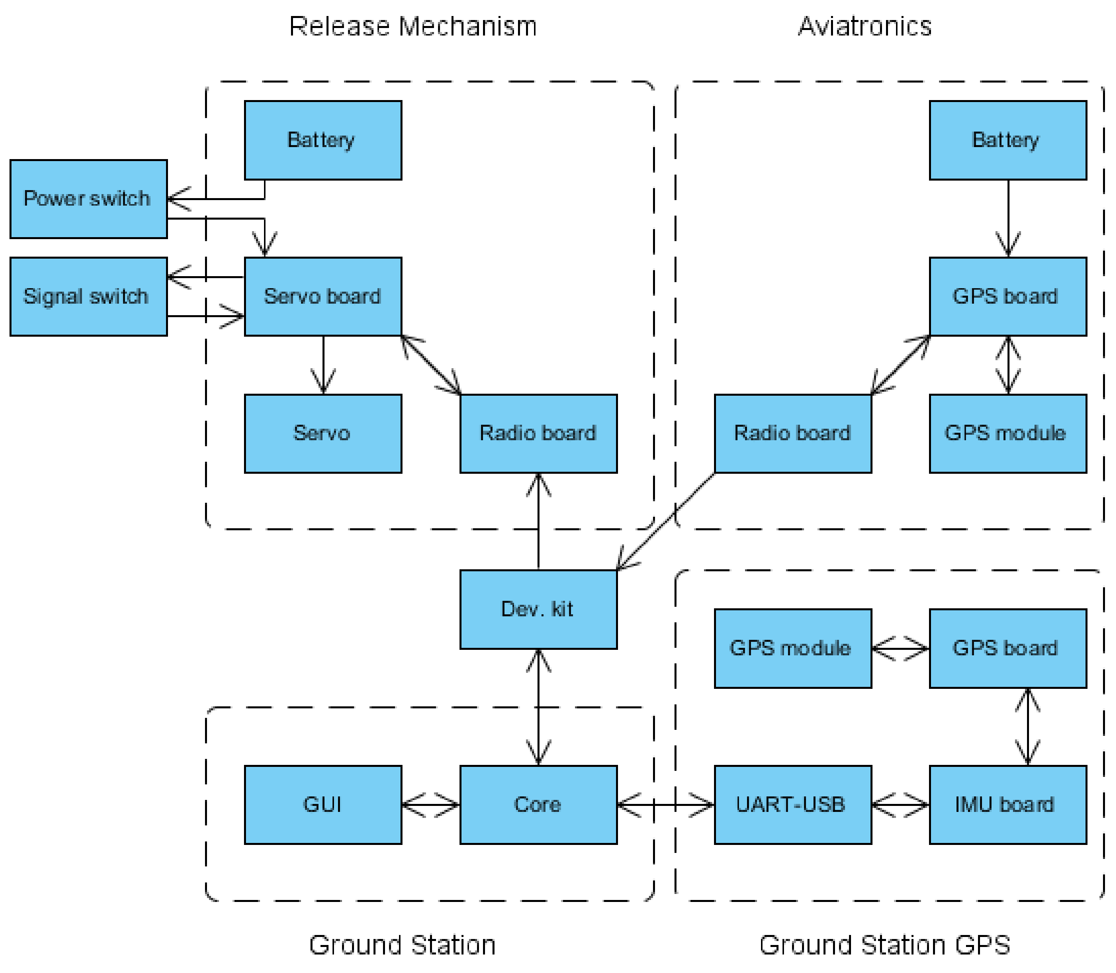
http://www.azom.com/details.asp?articleid=310

Table 1. Typical properties for aluminium

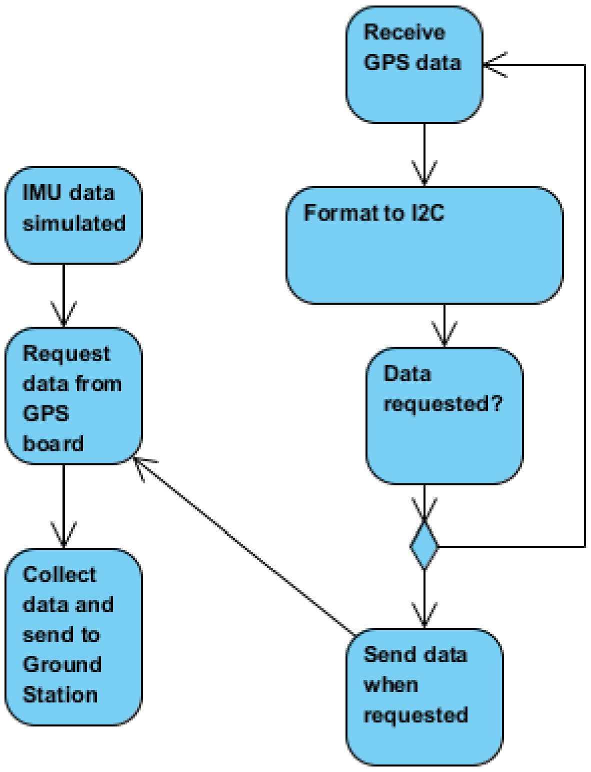
Property	Value
Atomic Number	13
Atomic Weight (g/mol)	26.98
Valency	3
Crystal Structure	Face centred cubic
Melting Point (°C)	660.2
Boiling Point (°C)	2480
Mean Specific Heat (0-100°C) (cal/g.°C)	0.219
Thermal Conductivity (0-100°C) (cal/cms. °C)	0.57
Co-Efficient of Linear Expansion (0-100°C) ($\times 10^{-6}$/°C)	23.5
Electrical Resistivity at 20°C (μΩcm)	2.69

Density (g/cm^3)	2.6898
Modulus of Elasticity (GPa)	68.3
Poissons Ratio	0.34

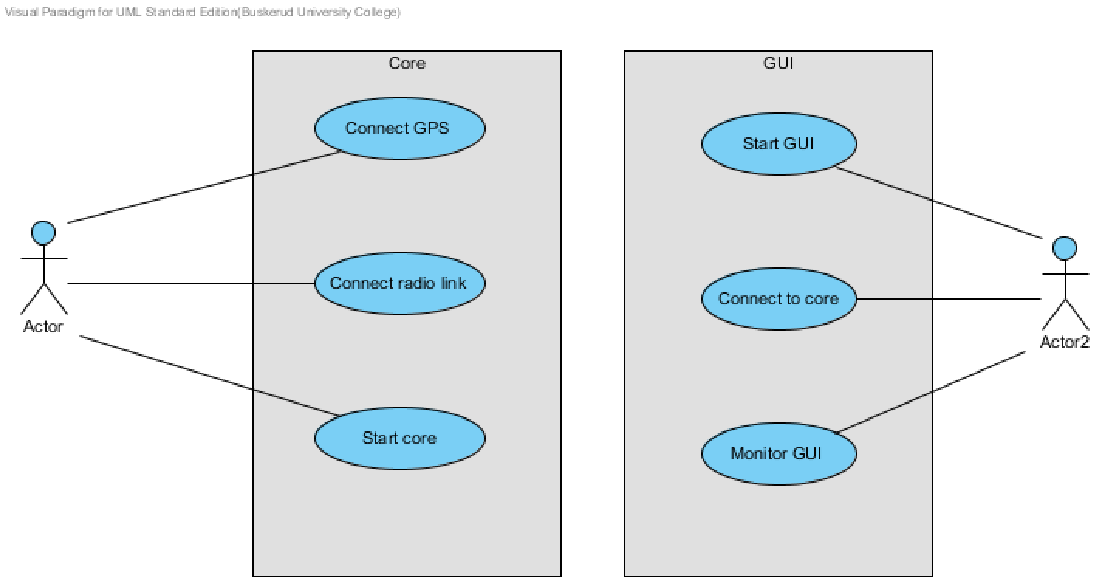
UML schematic



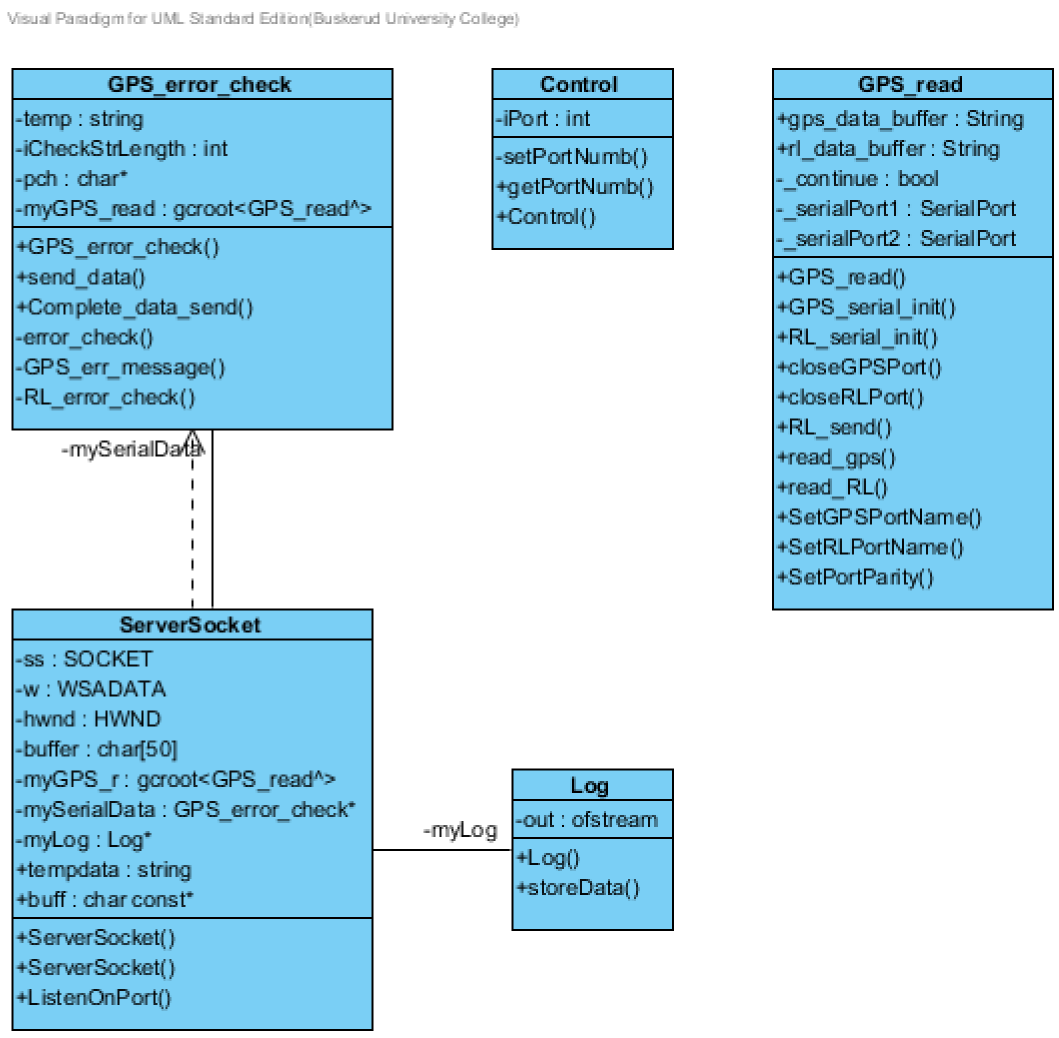
Logic Diagrams



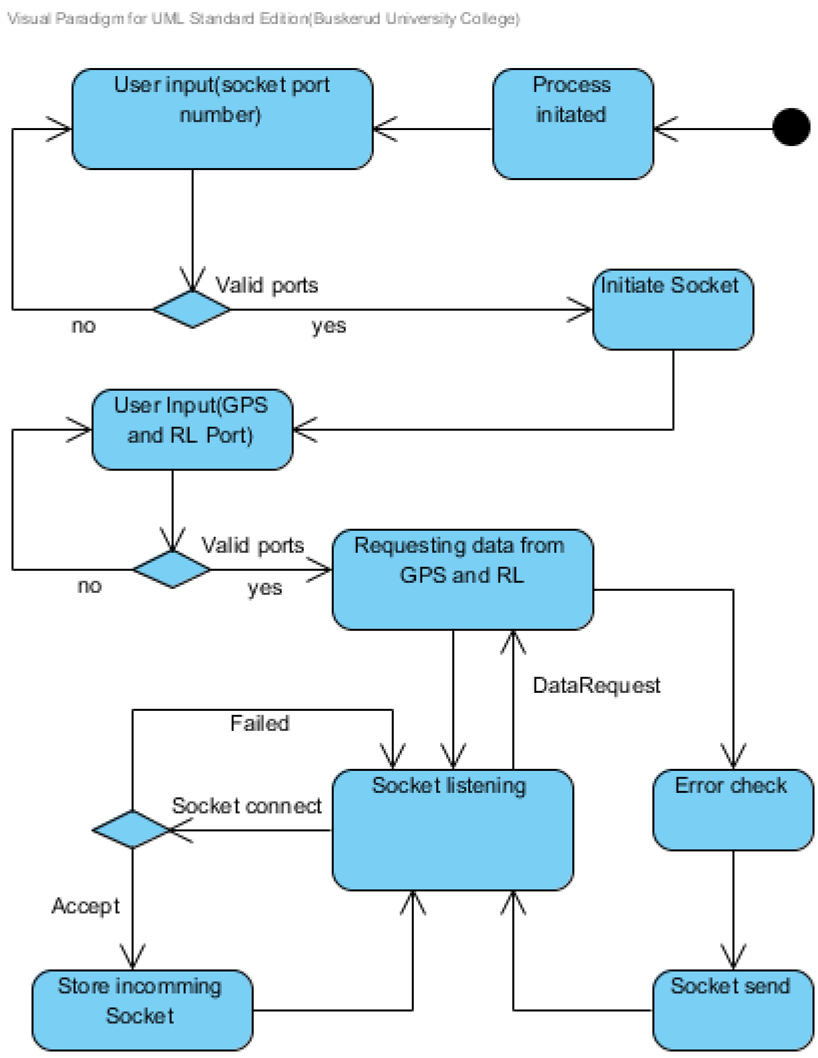
UseCase UML


Visual Paradigm for UML Standard Edition(Buskerud University College)
Core
Connect GPS
Connect radio link
Start core
Actor
GUI
Start GUI
Connect to core
Monitor GUI
Actor2

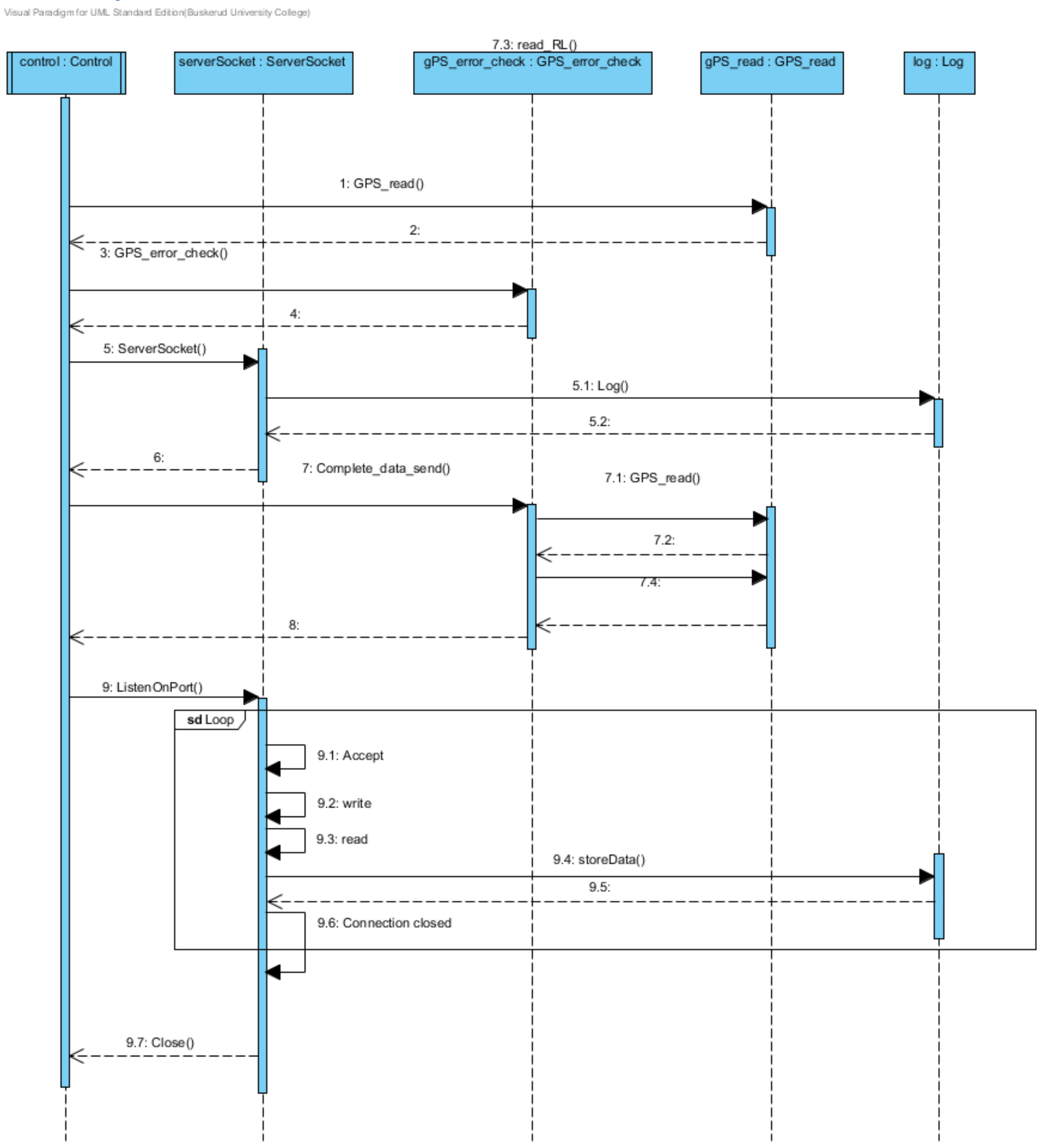
Core class UML



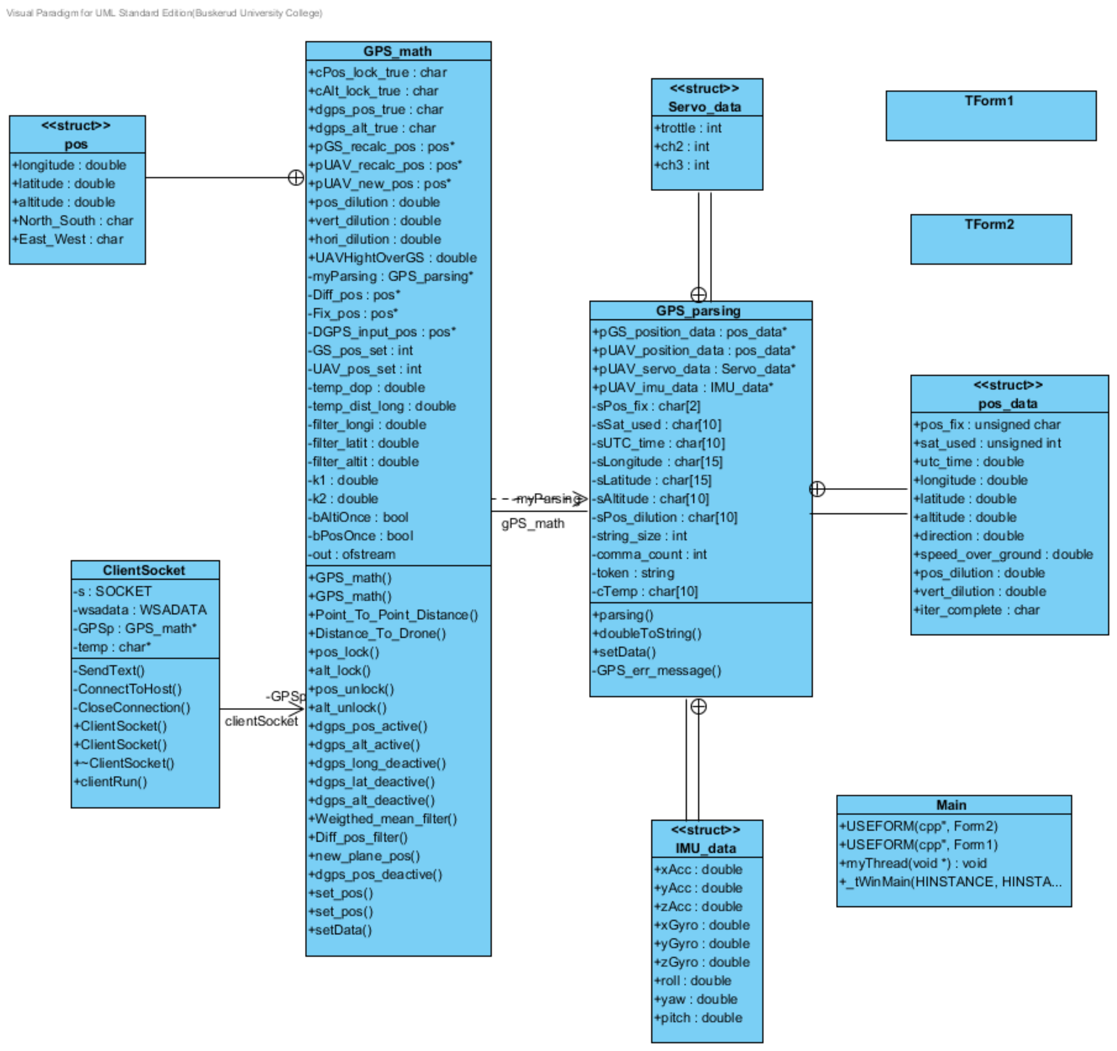
Core state UML



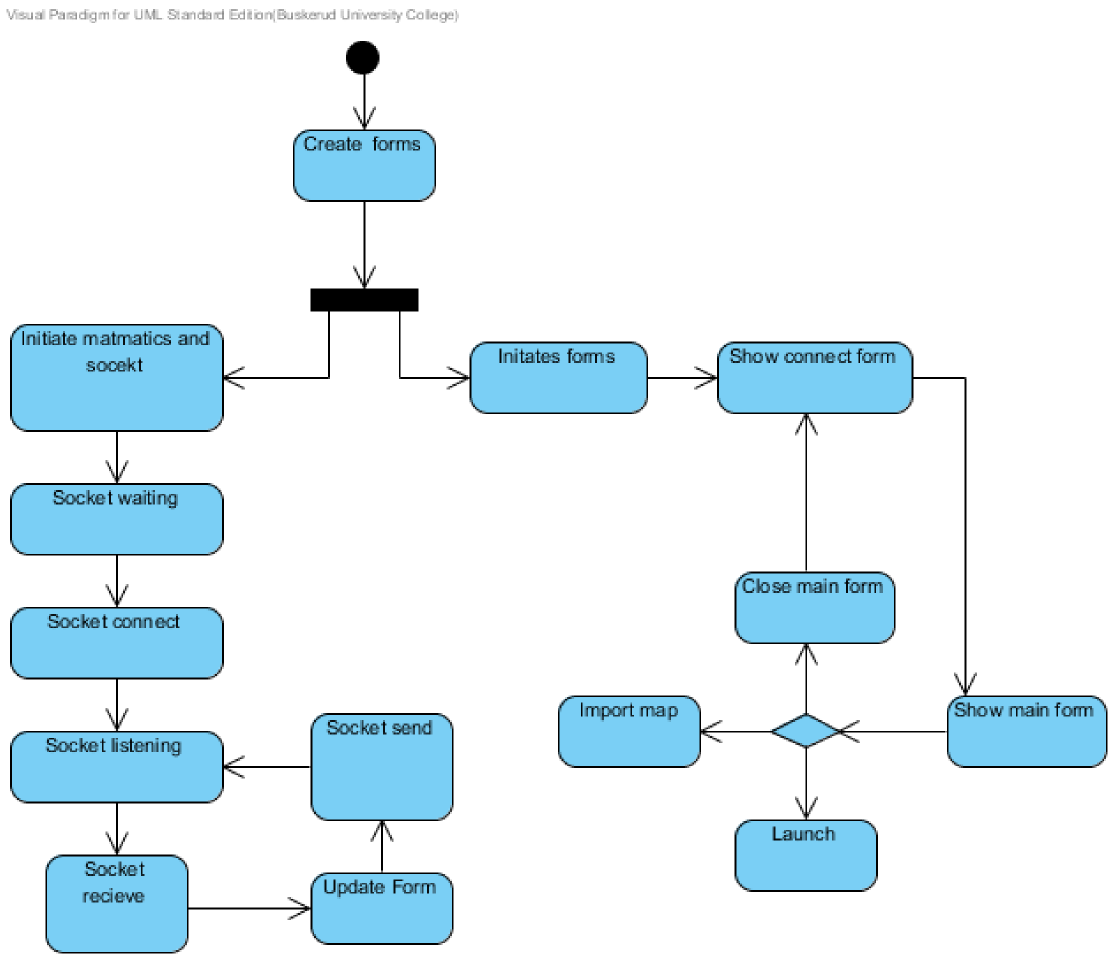
Sequence UML



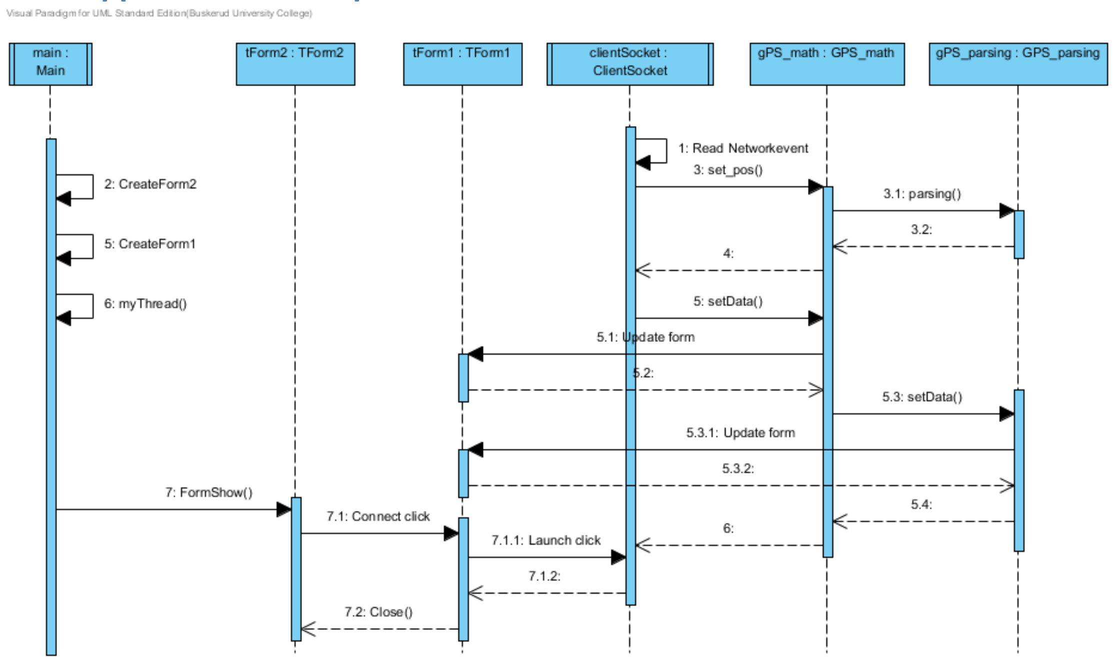
GUI class UML



GUI state UML



5.3 Appendix 3 GUI sequence UML



5.4 Appendix 4 Circuit board schematics

The following circuit boards have been given to us by KDA.

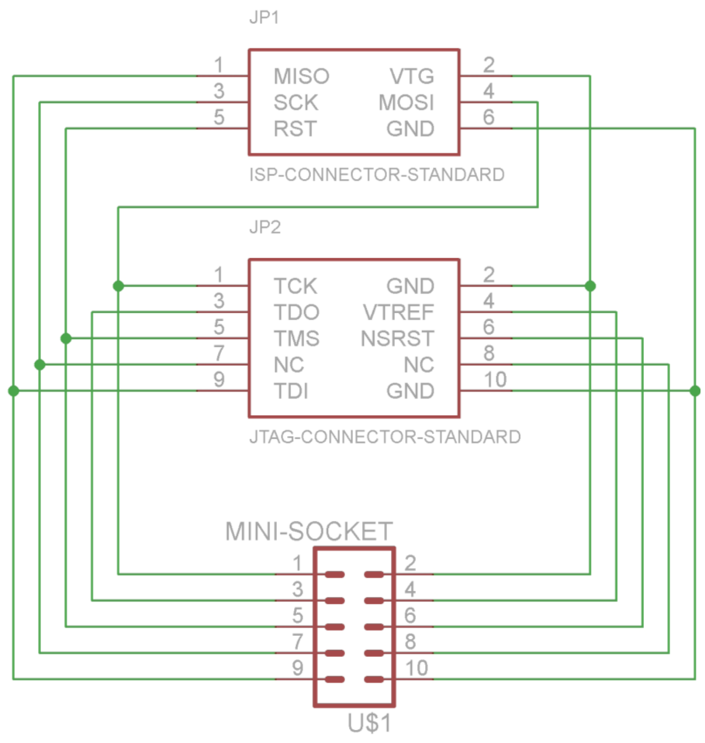


Figure 70: Programming adaptor

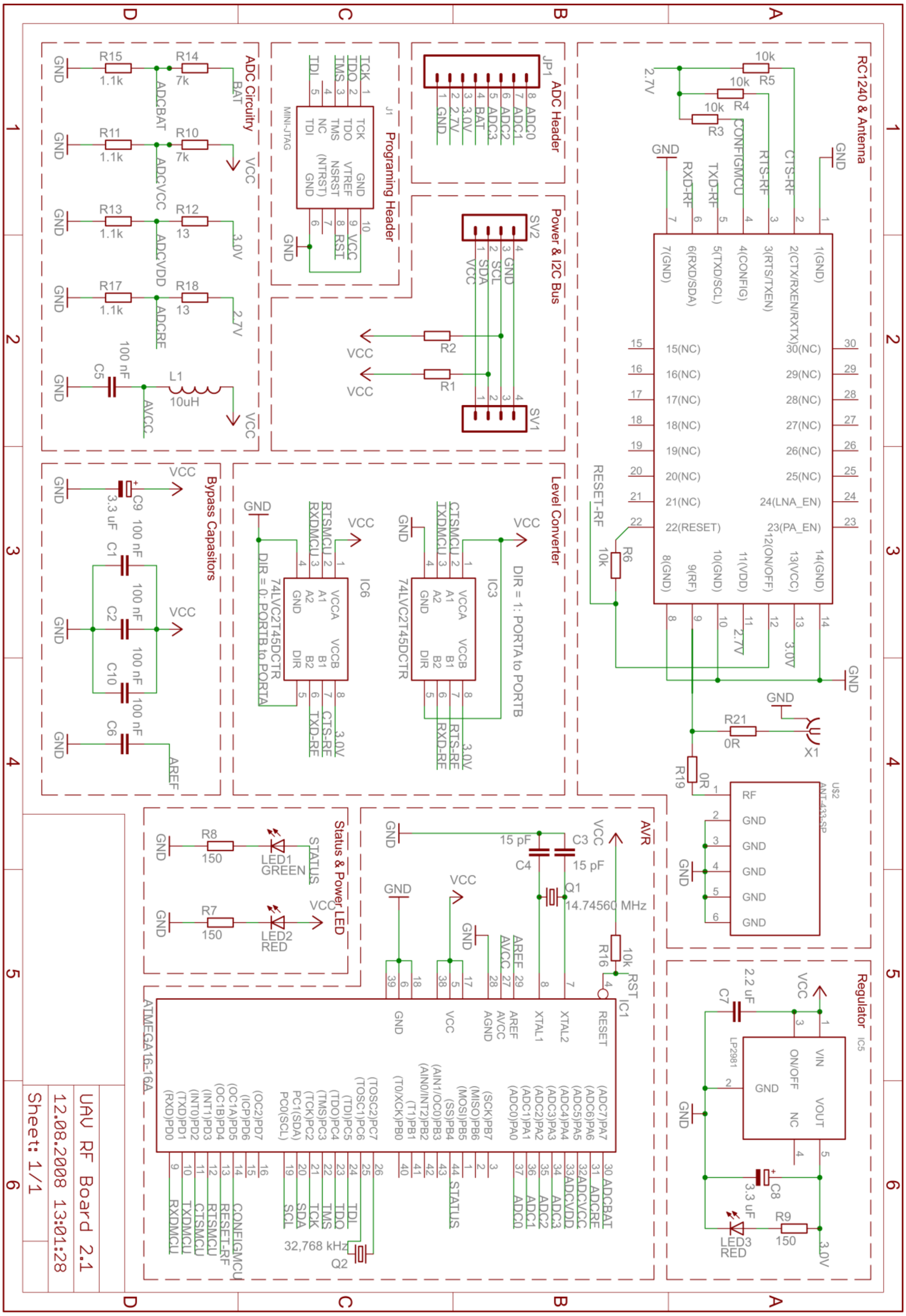


Figure 71: Radio circuit schematic

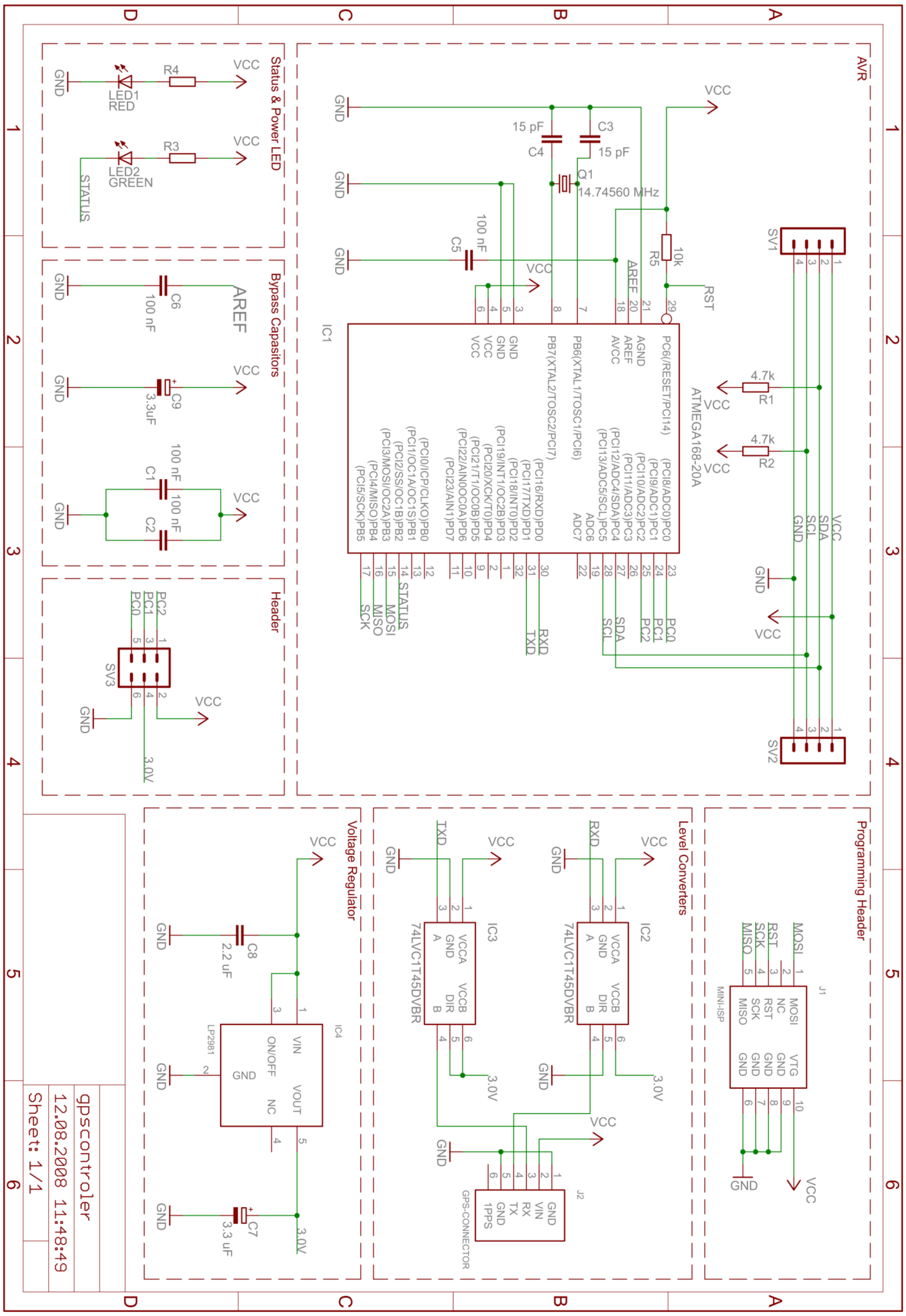


Figure 72: GPS circuit schematic

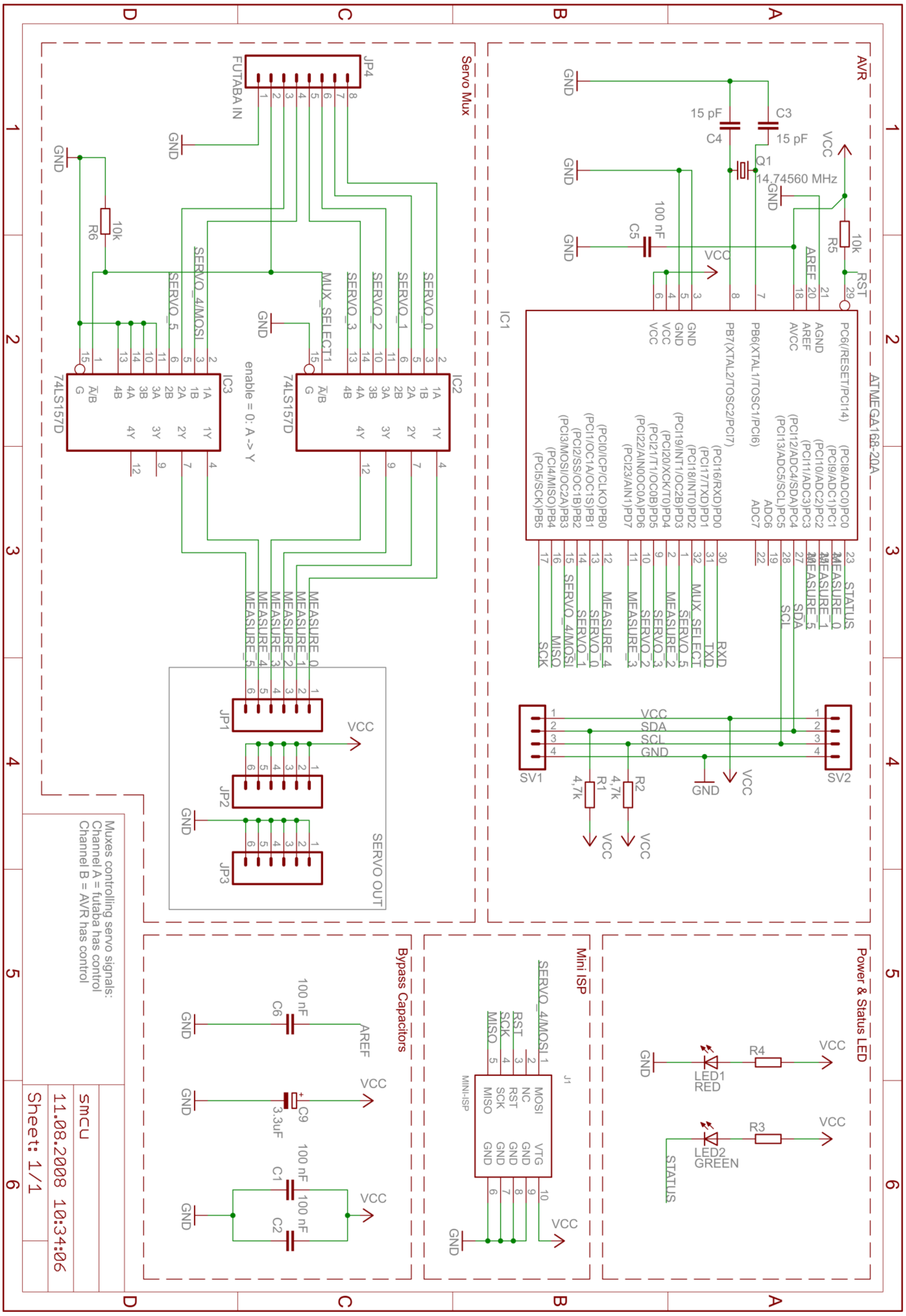


Figure 73: Servo circuit schematic

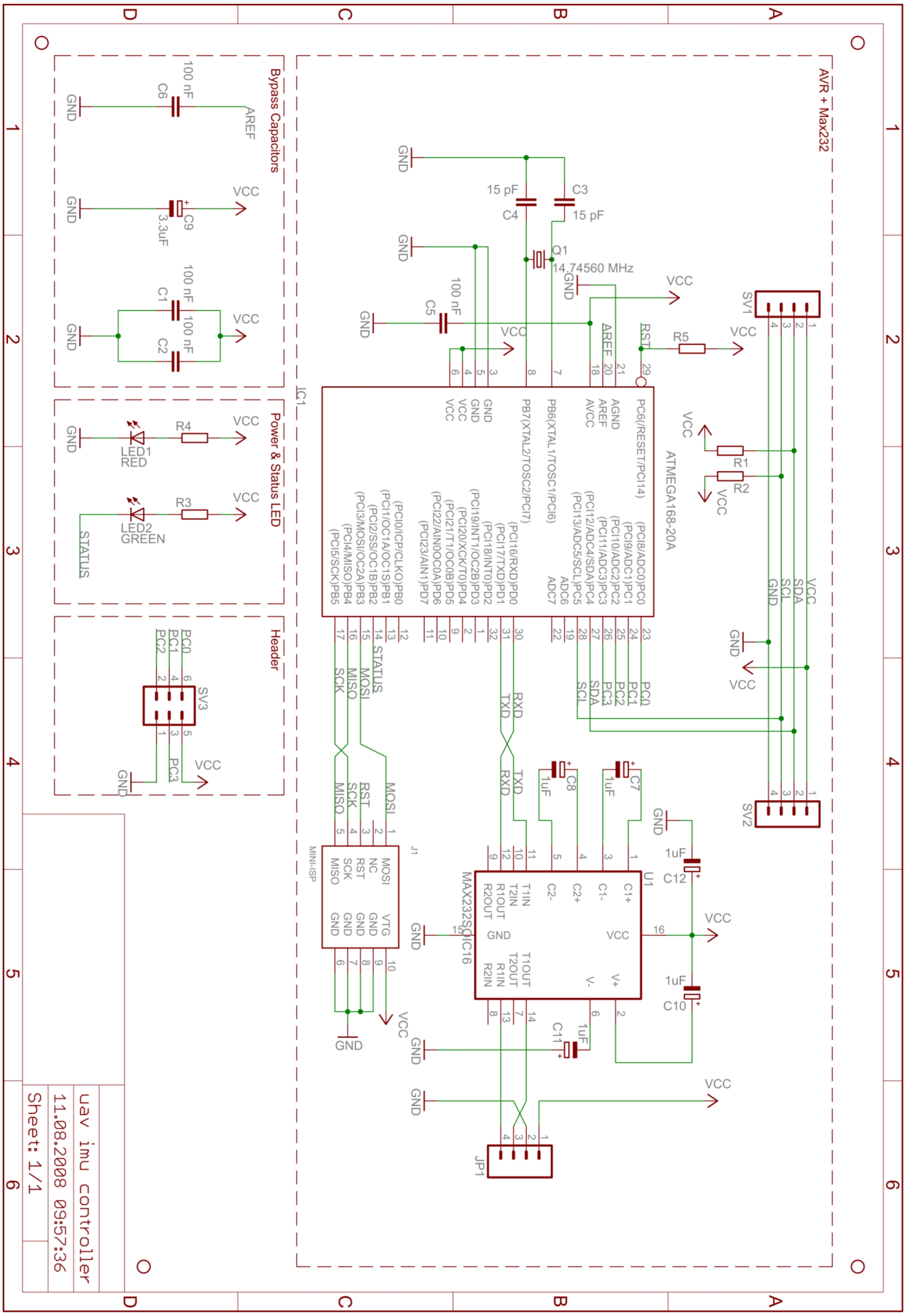


Figure 74: IMU circuit schematic

HØGSKOLEN
i Buskerud

ACCOUNTANCY

LocalHawk

Jon Bjørnland

Erik Myklebust

Lars Erik Røise

Lars Hallvard Sannes

May 25th 2010

Table of Contents

1. Introduction

In this document you will find out about our restriction in the economical part of the project, and how we have taken care of the financial. The budget was made in the start phase of the project, based on what resources we thought we needed. The actual investments completed are also part of the document.

2. Budget

The LocalHawk system already encompasses a lot of parts; therefore it was not necessary to buy everything that was going to be used. KDS also had some materials for the launcher available.

Total monetary assets:

Post	Total	
Budgeted KDS	3000,00	NOK
Total	3000,00	NOK

Following is the distribution of the monetary resources that was available to us.

Post	Total w/o materials	Total w /materials
Launcher		
Wagon materials	-	800,00
Rubber band	300,00	300,00
Bracket	50,00	50,00
Fasteners	30,00	30,00
Basic 16x2 Character LCD - 5V	175,00	175,00
Battery	20,00	20,00
Trigger		
Strain measurer	250,00	250,00
Cable between aircraft and Launcher	100,00	100,00
Metal Parts, trigger	300,00	300,00
Microcontroller (minimum 14 digital and 1 analog I/O)	50,00	50,00
LM1117 5V regulator	10,00	10,00
Ground Station		
GPS interface circuit	250,00	250,00
Team building	365,00	365,00
Total procurement	1900,00	2700,00
Tax base	1425,00	2025,00
Tax	475,00	675,00
Shipping	300,00	300,00
TOTAL, LOCALHAWK- PROJECT	**800,00**	**0,00**

There are two columns one with materials and one without, this is because KDS, at the time the budget was written, did not know what they had in storage.

3. Accountancy

This is the actual cost on our project. Material was delivered by KDS, and some was found in lying around in the workshop used, thus reducing the expenses.

Post	Amount :	unit dollar price	total price	Store
Launcher/Wagon				
superlim gel	1		19,90	Biltema
planskive 100stk	1		39,90	Biltema
sekskantskrue M4x30	1		34,90	Biltema
sekskantskrue M5x30	1		34,90	Biltema
låsemuttersett	1		49.90	Biltema
bor spiral SB 5,5	1		19,00	Byggmakker Ski
bor spiral SB 4,2	1		12,50	Byggmakker Ski
baufilblad 2 pakk	1		52,50	Byggmakker Ski
wireklemme 5mm	1		19,00	Jernia
Ståltråd	1		59,90	Jernia
Trigger				
bor spiral SB 6,8	1		19,12	Byggmakker Ski
bor spiral SB 4,5	1		10,62	Byggmakker Ski
0,27 kg bolter skruer mutter	1		59,92	Byggmakker Ski
Kutteskive	1		84,00	Jernia
Ground Station				
FTDI Basic Breakout - 5V	1	$13,95	90,49	Sparkfun
Interface Cable for EM401 and EM406	2	$1,95	25,30	Sparkfun
Breakout Board for Serial DB9	2	$0,95	12,33	Sparkfun
Break Away Headers - Straight	1	$2,5	16,22	Sparkfun
shipping and handling		$9,23	59,88	
TOTAL, LOCALHAWK- PROJECT			670,37	

Remaining in the budget: 3000NOK – 670,37 NOK = 2329,63 NOK

Used a dollar price of 6.40 NOK

4. Conclusion

The budget was made from what was thought needed and what was essential to have. Many of the components that have been included in the LocalHawk system were already available from KDS. Some components was already bought for earlier LocalHawk projects, and some was at KDS own workshop. This made a lower actual cost than expected. Some parts in the budget were also eliminated, because the decision was made to reuse some of the old electronics instead of creating new. On the other hand several tooling equipment had to be bought, because Buskerud University College did not have everything that was needed.

HØGSKOLEN
i Buskerud

CONCEPT DOCUMENT

LocalHawk

Jon Bjørnland
Erik Myklebust
Lars Erik Røise
Lars Hallvard Sannes

April 9th 2010

Introduction

This document contains the concept ideas developed during the planning stage this project

The Ground station is developed for the LocalHawk project, run by Kongsberg defence system (KDS). It is comprised of two parts, a hardware part and a software part that works together and makes it possible for you to control and communicate with the LocalHawk unmanned air vehicle (UAV)

The Launcher is developed for the LocalHawk project run by Kongsberg defence system. It is build for making the takeoff of LocalHawk airplanes autonomous. This system consists of the launcher frame and a corresponding wagon (WGN) that holds the aircraft. The wagon is held in place by a trigger device (TRG) as the rubber bands gets tighten.

Table of content

Software

The software part of the ground station is a graphical user interface (GUI) that fronts a core application. The core application communicates with a hardware that relays information from the plane.

GUI

The GUI will graphically show you the information that's received from the aircraft, it also allow you to manipulate the aircraft's parameters and control heading, speed, etc.

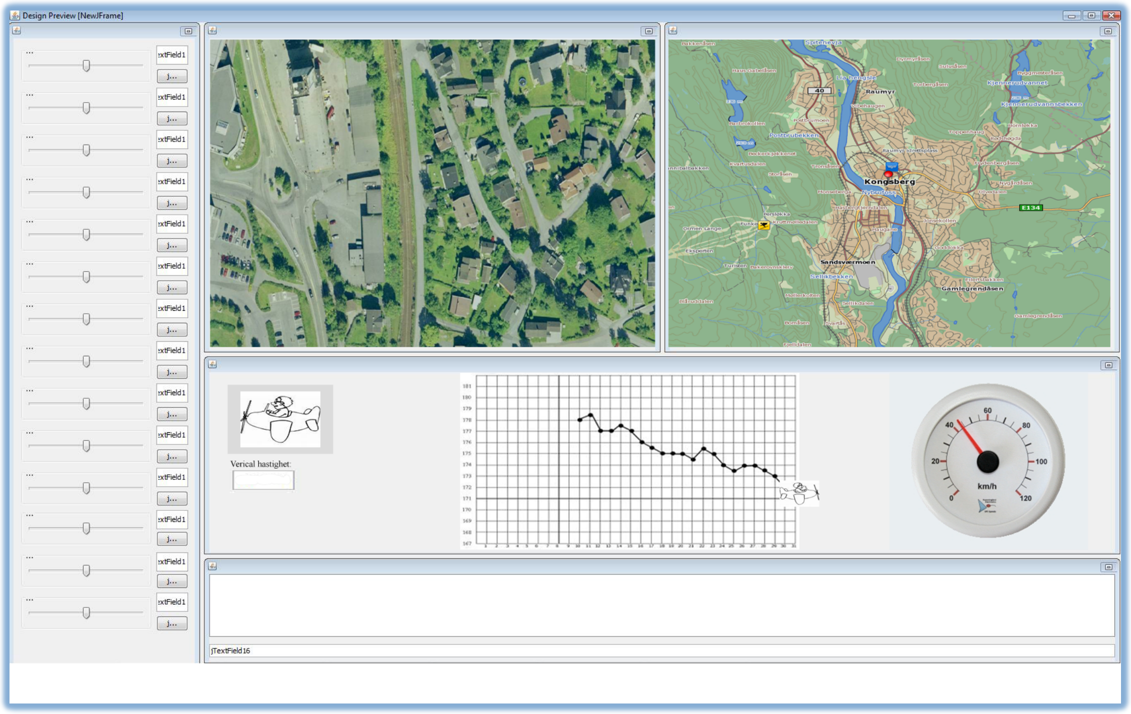


Figure 1: The GUI, this shows a quick concept sketch of how the GUI might look

The will contain:

- A map, that shows the position of the aircraft.
- Telemetry information (Speed, height, heading etc.)
- A consol, for direct input.
- Parameter slider input, for error correction.

Concept

Concept 1

The fist concept was to simply display all information in one window. Have all the GUI objects at a locked position.

Concept 2

The second was to group the GUI objects in separate windows in one frame; the windows should be sizable. This allows the user to focus on specific information.

Concept 3
The third concept was to use tabbed view, with one tab for all the info in a smaller format, and tabs with more detailed view of chosen grouped information.

Choice
The solution we went with is the tabbed view (concept 3), we found that the development of the GUI application would be less complex to develop, and would be easier to use and understand. It would also allow the ground station to ask more often for certain information, for example telemetry if that is the tab the user is viewing.

Core application
The core application is as previously mentioned the GUI's back bone, it acts like a driver that formats the data it receives from the hardware part and makes it "GUI friendly" and vice versa. It is required by KDS that this application is cross-platform.

Concepts
The different designs we discussed were to use a socket connection between the GUI and core vs. using a direct connection between the two. The socket would make it possible for remote viewing of the information relayed from the plane, and free up more possibilities for the feature development of the application. This socket solution would be more complicated to implement then the integrated GUI, because this would add another communications layer between the GUI and core.

The other aspect of design we viewed was the logging and/or storage of the information; we considered using a database (DB) for storing the information. A DB based storage would give easy access to wanted information and good organization of the flight data if one use a online data base it would very easily be accessed from anywhere if there is a internet access available, but you would be dependent on a constant internet connection to log the flight info, and it would also mean you need to have DB programs installed on your computer. The other option was to simply store the data to a file, this would be easy to implement and would not need any other programs to work, and the file would be easy to use with other programs if correctly formatted.

Choice
We chose to go with a design implementing a socket interface; this will open for more possibilities for both usage and further development of the GUI for example in the way of multiple users. For logging we chose to go with the file logging instead of the DB storage, the reason for this is that it is easier to implement and doesn't rely on any other applications as the DB does.

Hardware

The hardware part is a circuit that interfaces the GPS and wireless link with the ground station computer. The circuit encompasses two AVR ATxmega microcontrollers, connected together using the I^{2}C protocol, the RC1240mc wireless link, the EM406 GPS antenna and a RS232 level converter for communication with the ground station computer. It includes most of the GPS parsing and formatting code as well as the entire wireless communications protocol.

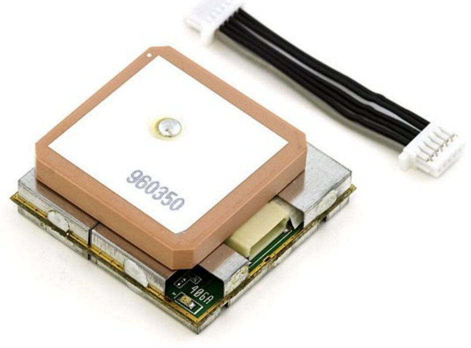


Figure 2: The EM406 GPS antenn3a

Figure 3: The RC1240mc RF module

Concepts

We originally had two concepts for interfacing

Concept 1

Connect the two components separately as peripherals and program all the data formatting, parsing and wireless code in the ground station software. Here we would have to create a simple circuit that made it possible to connect the GPS to the computer. KDS has an RC1240mc demo board that can be connected to the computer. This is what is used for today's ground station solution.

Concept 2

Connect the GPS and wireless link to one circuit board and program all the data formatting, parsing and wireless code in microcontrollers on the board. Here we would have to create a circuit. This circuit would need to have the RC1240 mc RF module, connections for the GPS module, minimum one microcontroller and some form of computer interface (RS232 or USB).

We later in the project stage found that an other bachelor student, Per Magnus Veierland, who is working on improving the onboard avionics have created just this kind of circuit, The Phoenix II. We contacted Per Magnus and it was possible for us to get a copy of this circuit board.

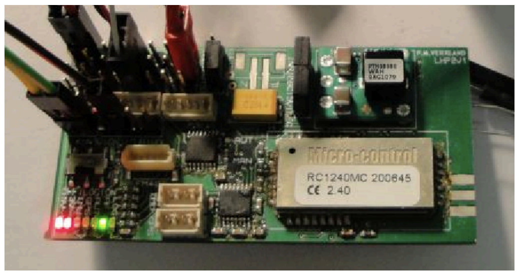


Figure 4: The Phoenix II, Designed by P.M. Veierland

Concept 3

Use the old existing avionics circuit boards from the LocalHawk aircraft and program a form of computer interface. There are originally four avionic circuit boards, a GPS board, an IMU[1] board, a servo measurement board and a radio link board connected together via 4 pins, a two-wire interface (TWI or I2C[2]), 5V and ground

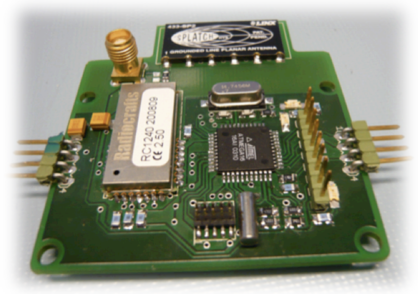


Figure 6: The RF circuit board

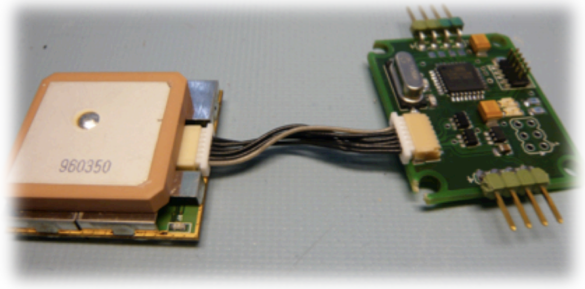


Figure 5: The GPS circuit board with connected GPS

The main gist of this concept is that we will use the RF– and the GPS circuit connected together with either a UART programmed on one of the boards or using the IMU board as a dedicated UART circuit

As these circuits will later this year be replaced by the Phoenix II, as described in concept 2, we will be able to use the old circuits mainly for this purpose.

Choice

We opted for the concept 2 when we got the info about the possibility of receiving a circuit from Veierland. As opposed to connecting the GPS and wireless link as peripheral. We found that by moving the processes out of the core application and on to an embedded platform we can further ease the development of the cross-platform aspect of the core application and using preexisting components lowers the cost and time spent by eliminating the need to create and produce a new circuitry.

Comment

As it became clear to us that we would not get our hands on the Phoenix II until after Easter, this meant that we could not start working on this platform until the middle of April. This gave us a time problem, we got nothing to do until April and too much to do after.

Due to this we started to work on developing concept 3 in January. This concept is almost identical to concept 2, except that we had what we needed to start working right away. The work would be almost interchangeable in the sense that we could export what we had done on the concept 3 platform over on the concept 2, with minimum effort.

[1] Internal Measurement Unit; is an electronic device that measures and reports on a craft's orientation in three-dimensional space
[2] IIC or Inter-Integrated Circuit is a multi-master serial single-ended computer bus invented by Philips that is used to attach low-speed peripherals, such as microcontrollers

At the end of March we, in dialog with our external supervisor, decided to drop concept 2 entirely as we saw that we would not have enough time to successfully export the programming to the Phoenix II platform.

Wagon system

The wagons function is to transport the airplane, while it builds up enough speed for takeoff (approximately 10 $^{m}/s$). The existing wagon is limited to only launch airframes with back mounted trust (as the existing LocalHawk plane). Therefore we need to come up with a new solution that can also handle airframes with back mounted propeller.

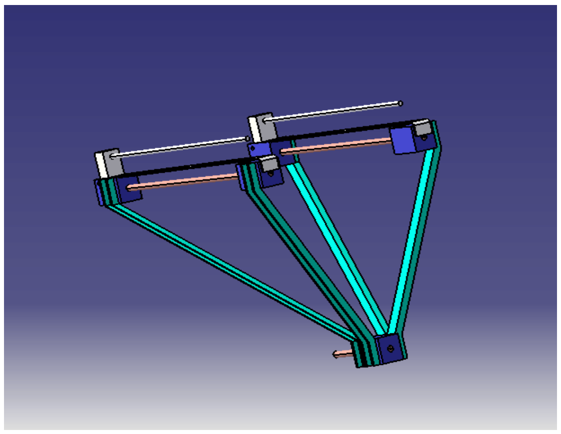


Figure 7:The Wagon, this shows an approach of the old wagon.

When launching the Siren airframe with the old wagon you will find some difficulties. One of them is that the propeller will hit the launcher frames front. The size of this propeller is 11”, and will therefore not go through the gap on the launcher, which is a bit smaller.

Concepts

We came up with four possible concepts that fulfil our requirements. These concepts have different levels of complexity and workload.

Concept 1

The systems function is the same, but consists of two different types of wagons. One for front mounted propeller and one for back mounted. The one for the back mounted propeller already exist, but can be improved. The other wagon needs to be created from scratch. The idea with this new wagon is to have the airframe in a type of mould. The airframes propeller is also large (11 Inc), so the platform where the airframe lie needs to be lift up more than the existing wagon.

Concept 2

To design a completely new wagon that has the capability of launching airframes independent of the propellers position. This means that the wagon needs a lot of opportunities for adjustments.

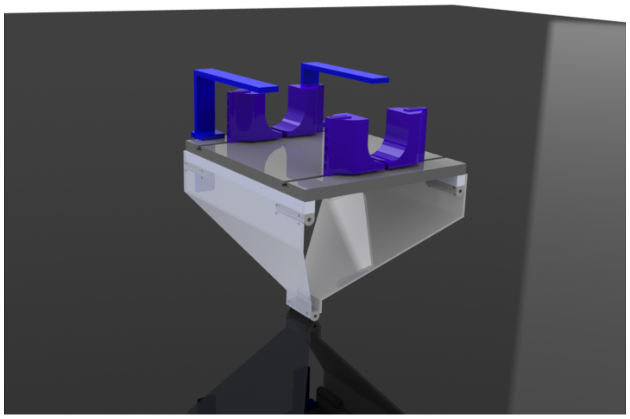


Figure 8:This shows an early concept CAD drawing of the wagon created for the Siren airframe

Concept 3

Instead of having two different wagons we have one main wagon frame. Then we can have onn “top” for each airframe type that we can assemble on the main wagon frame. The existing frame can be modified so it becomes the main frame, with possibilities to assemble each of the two platforms developed for each airframe.

Concept 4
To create a system that doesn't use the wagon for transporting the airframe. Instead of having a wagon, we can fasten the rubber band directly on the airframe body. The idea is to have a mounting bracket to a wire. And just drag this wire, and release. This concept is a more complex solution. It will maybe need one bracket for each airframe, and a different rail system.

Choice
The concept 3 where you have one base wagon and two different wagon set is chosen. This is for making the launcher easier to modify when you want to launch a different type of airframe. It`s also an advantage that the one airframe is easy to launch (Stryker f27c) with the existing system. The platform for the Siren airframe needs to be raised up higher. This is important for the propeller to the Siren. If not, we risk that the propeller hit the launcher frame.

Figure 3 shows an idea for the wagon to launch the Siren airframe. The top on this wagon can be redesigned to be an assembly part for the main frame.

Release mechanism
The function of the release mechanism is to hold the wagon in position when loading the launcher and to launch the wagon when desired. The system is created to be automatic using a servomotor for releasing.

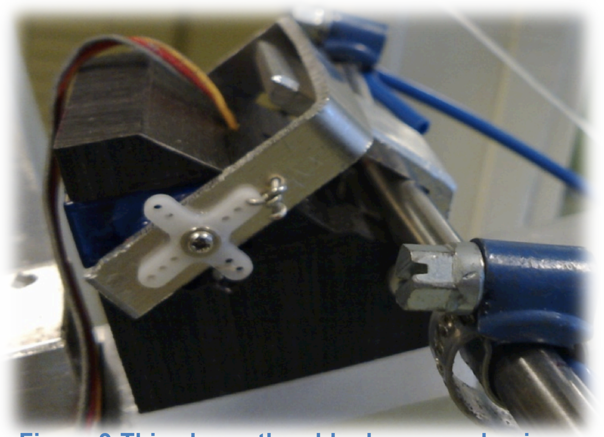


Figure 9:This shows the old release mechanism when holding the wagon

The old system did not function properly. The trigger was not automatic because of a too week servo. To get the release mechanism to work, they attached a rope around the holding pole. With this they could use the trigger, but it was not automatic, either safe. When operating this system, you had to be careful when the launcher was loaded, because of the risk for premature launch.

Concepts
The release mechanism must be redesigned, so it can function properly. It also has to be a much safer device, regarding the danger of premature launch.

Concept 1
The system is almost the same as the one that exist, just with a stronger servo and an additional safety device, some form of device that needs to be removed before releasing the trigger.

Concept 2

The system is simplified a lot from the existing release mechanism. Using a manual triggered mechanism. The idea is to lock the wagon and release with a rope.

Concept 3

This is a completely new system consisting of one permanent magnet and one electro magnet. This system locks the wagon, and is only released when a powers source is applied, removing the magnetic field.

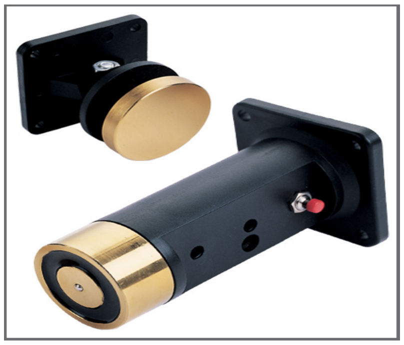


Figure 10: This shows an electro and permanent magnet

Concept 4

This is a system similar to the one that exist just an improvement. It is based on copying the solution from the car industry on door lock. Here we can just take a door lock and redesign it and add a servo for triggering. In addition add a safety device.

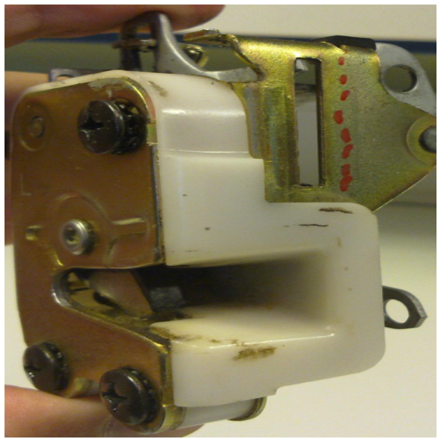


Figure 11: This shows a car door latch

Choice

The solution we have chose is concept 4 containing the car door latch. The solution is developed by the car industry over several years, and has become a good solution. The force needed to open the lock is less than the force needing to hold it. That's why it is a good system. Another reason is that it is relative easy to adjust to fit our launcher/wagon. In addition to adjust the car door lock we will add a safety device. The safety device will consist of a pin that locks the wagon when it is placed in the trigger. This needs to be removed before releasing the trigger lock.

HØGSKOLEN
i Buskerud

DOCUMENT OF DISCRIPTION

LocalHawk

Jon Bjørnland

Erik Myklebust

Lars Erik Røise

Lars Hallvard Sannes

March 10th 2010

Table of Contents

Document of description (visjonsdokument) Localhawk HiBu-09/10

Introduction

Localhawk is the name of a project that has been worked on by several project teams to generate one product. Localhawk has evolved throughout the last years from what is defined as the beginning in 2007 up to what was completed this summer of 2009. Kongsberg Defence Systems (KDS) is the company managing this project as a whole. Our contact at KDS is Jon Bernhard Høstmark that also will be (one of) our external guidance counselor.

The status of the project as a whole today is an Autonomous Unmanned Aerial Vehicle (AUAV), which is an aircraft equipped with a GPS, a communication antenna, an autopilot, and a camera that processes and analyses video in real time. It is launched from a "slingshot" into air where it is operated mainly from a ground station. The ground station communicates with the AUAV through a two-way radio-link. This gives orders to the UAV like magnitude and direction for the autopilot.

Tasks
The project will this year, as well as previous years, be worked on by students of different universities and colleges. We have been given the task of:

Software
- Rwrite the Ground Station application from Java to C++ to enhance real time computation.
- Create a user friendly GUI (Graphical User Interface) possibly in three dimensions with new applications

Electronics
- Make a GPS-unit for the ground station and making the GPS data accessible from the server application
- Prevent the wireless link from collapsing due to data overflow (today's solution sends too much information compared to what the wireless link can manage to process)
- Calibrate the load measurements
- Look into electronic load measurements
- Trigger-mechanism on the launcher, possibly remote controlled from the ground station

Mechanics
- Look at the structural weaknesses of today's launcher and make necessary improvements
- Perform lab tests (high speed camera, escape velocity, etc.)
- Upgrade the breaking system for the wagon on the launcher. Today's solution does not stop the wagon before crashing into the frame
- Improve the launcher wagon for better usage and enable different aircraft-design
- Use a different structure stabilizer, today the structure is stabilized by a wire, which is to unmanageable

Other
- Complete the component criteria, test specifications etc.
- Record a movie teaser, which publicizes our work throughout the year for students to come
- Create a user manual for the ground station and the launcher so that the next group working on this project will more easily understand previous works
- Make an A2 poster presenting the work achieved during this project

KDS has expressed interest in pursuing the software part even further by implementing transmission and inquiry protocols and a log system for security of the data in case of system malfunction. For this task to be completed we need a second software engineer. This part is still undecided.

Why we have chosen this project

Our team (at this point) consists of four members; one machine design-, two mechatronic-, and one computer simulation engineer. We are all excited to take part of this project for two reasons:

- KDS has run similar projects several times before and know what to expect from a student team. This also gives us, the students, confidence in that KDS will be following the project regularly.
- The project itself looks interesting in many ways. It focuses on incorporating several branches within engineering making a unified system. This gives the students insight to other parts of engineering.

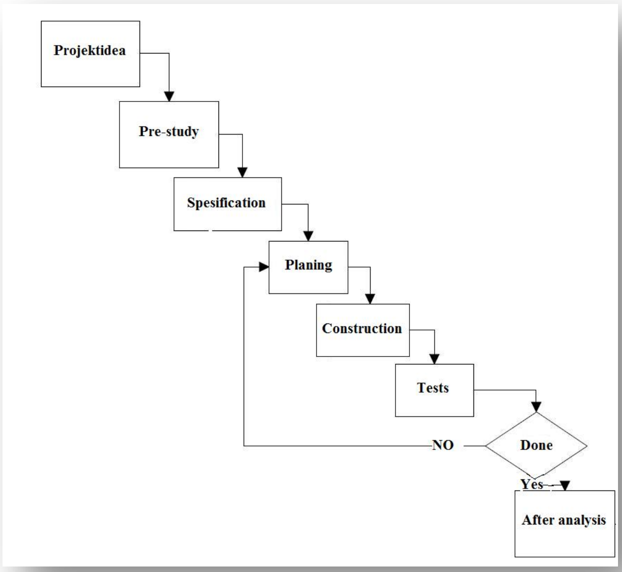
Project Model

Incremental development

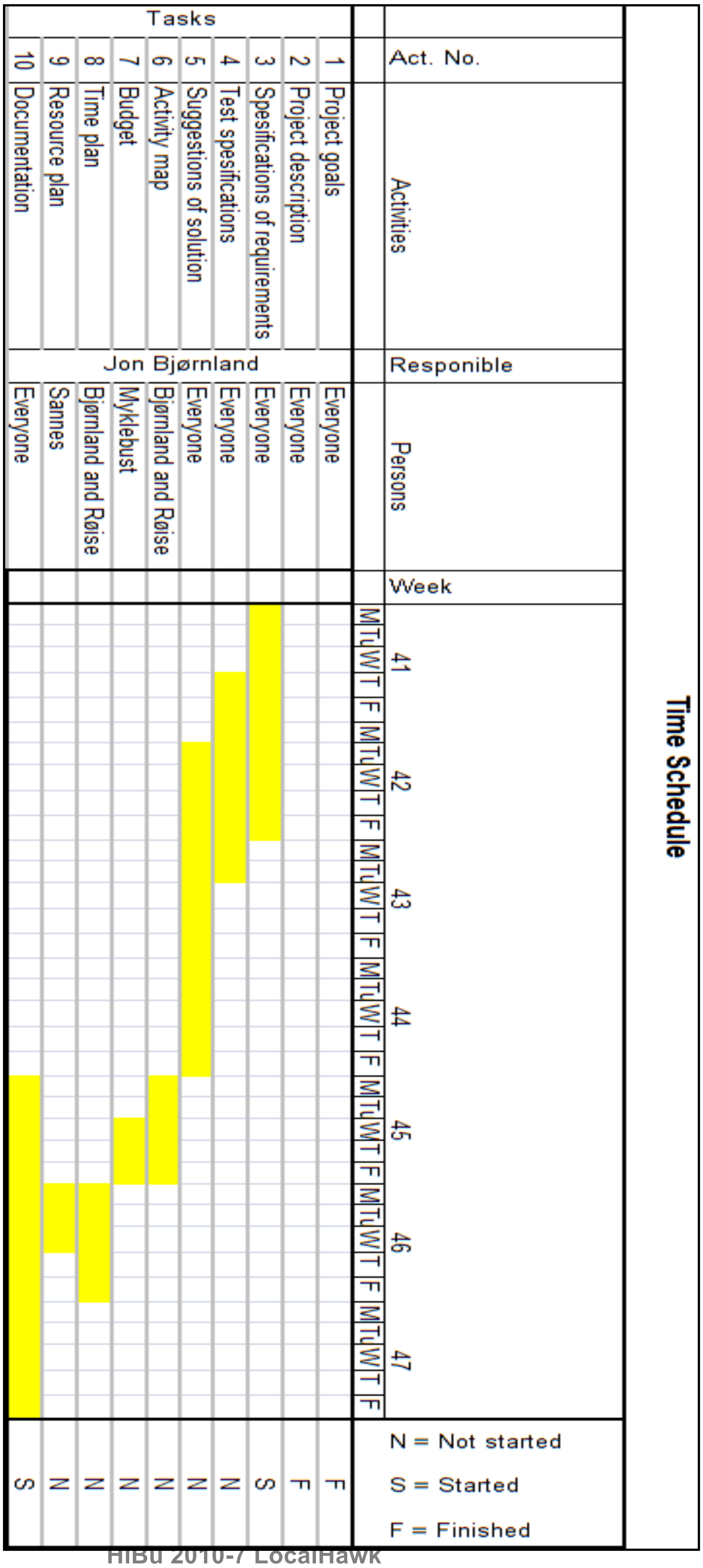
This project is built from many different components. These components are divided into smaller parts that will eventually come together as one. The components all have a different importance in the project; taking into account how much that is dependent on the component and also its significance. This is called incremental development.

Prototype development

For best time efficiency while working on this project we have chosen the evolutionary project development method. The reason this matches best our kind of work is that none of us have previously worked on a project of this magnitude. Earlier there has been no need for planning of this caliber. Either the complete objective has easily been foreseen, or the Ad hoc-model has worked well. There is a great chance the plans that are being made needs further improvements, or be thrown away, because of unforeseen obstacles. This way of progressing towards a goal is called prototyping. The reason this suits us best is the dynamics that helps us go back and improve plans because of inaccuracy.



Time Schedule



Time Schedule

Tasks — Responible: Jon Bjørnland

Act. No.	Activities	Persons	41					42					43					44					45					46					47					
			M	Tu	W	T	F	M	Tu	W	T	F	M	Tu	W	T	F	M	Tu	W	T	F	M	Tu	W	T	F	M	Tu	W	T	F	M	Tu	W	T	F	
1	Project goals	Everyone																																					F
2	Project description	Everyone																																					F
3	Spesifications of requirements	Everyone	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨																											S
4	Test spesifications	Everyone				🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨																									N
5	Suggestions of solution	Everyone							🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨																	N
6	Activity map	Bjørnland and Røise																					🟨	🟨	🟨	🟨	🟨												N
7	Budget	Myklebust																							🟨	🟨	🟨												N
8	Time plan	Bjørnland and Røise																										🟨	🟨	🟨	🟨	🟨							N
9	Resource plan	Sannes																										🟨	🟨	🟨									N
10	Documentation	Everyone																					🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨		S

N = Not started

S = Started

F = Finished

Contact Information

Kongsberg Defence Systems

Business activities

“Kongsberg Defence Systems is Norway's premier supplier of defence and aerospace-related systems. The portfolio comprises products and systems for command and control, weapons guidance and surveillance, communications solutions and missiles. Kongsberg Defence Systems also makes advanced composites and engineering products for the aircraft and helicopter market. The Norwegian Armed Forces is the Business Areas most important customer. Whether developed in collaboration with the Norwegian Armed Forces, international partners or alone, the BA's solutions have proven highly competitive internationally. One key element of the Business Areas market strategy is the formation of alliances with major international defence enterprises. All defence-related exports are contingent on the approval of the Norwegian authorities.” – About Us - Kongsberg Gruppen

Contact information
Kongsberg Defence Systems
Kirkegårdsveien 45
P.O. Box 1003
NO-3601 Kongsberg
Norway
Telephone: +47 32 28 82 00

The Project Group

Jon Bjørnland

Project leader
Field of study: Mechatronics
Phone: 93609860
E-mail: bjornland@gmail.com
Area of expertise: Reverse engineering

Erik Myklebust

Field of study: Mechatronics
Phone: 93838712
E-mail: erikmykl@gmail.com
Area of expertise: Theory

Lars Hallvard Sannes

Field of study: Product design
Phone: 90884799
E-mail: larshsann@hotmail.com
Area of expertise: Problem solving

Lars Erik Røise

Field of study: Simulation and game development
Phone: 40610744
E-mail: Roise88@gmail.com
Area of expertise: Troubleshooting

HØGSKOLEN i Buskerud

INTERFACE DESIGN

LocalHawk

Jon Bjørnland

Erik Myklebust

Lars Erik Røise

Lars Hallvard Sannes

March 10th 2010

1. Table of Contents

2. Introduction

This document describes how and on what form the information is passed between different parts of the system.

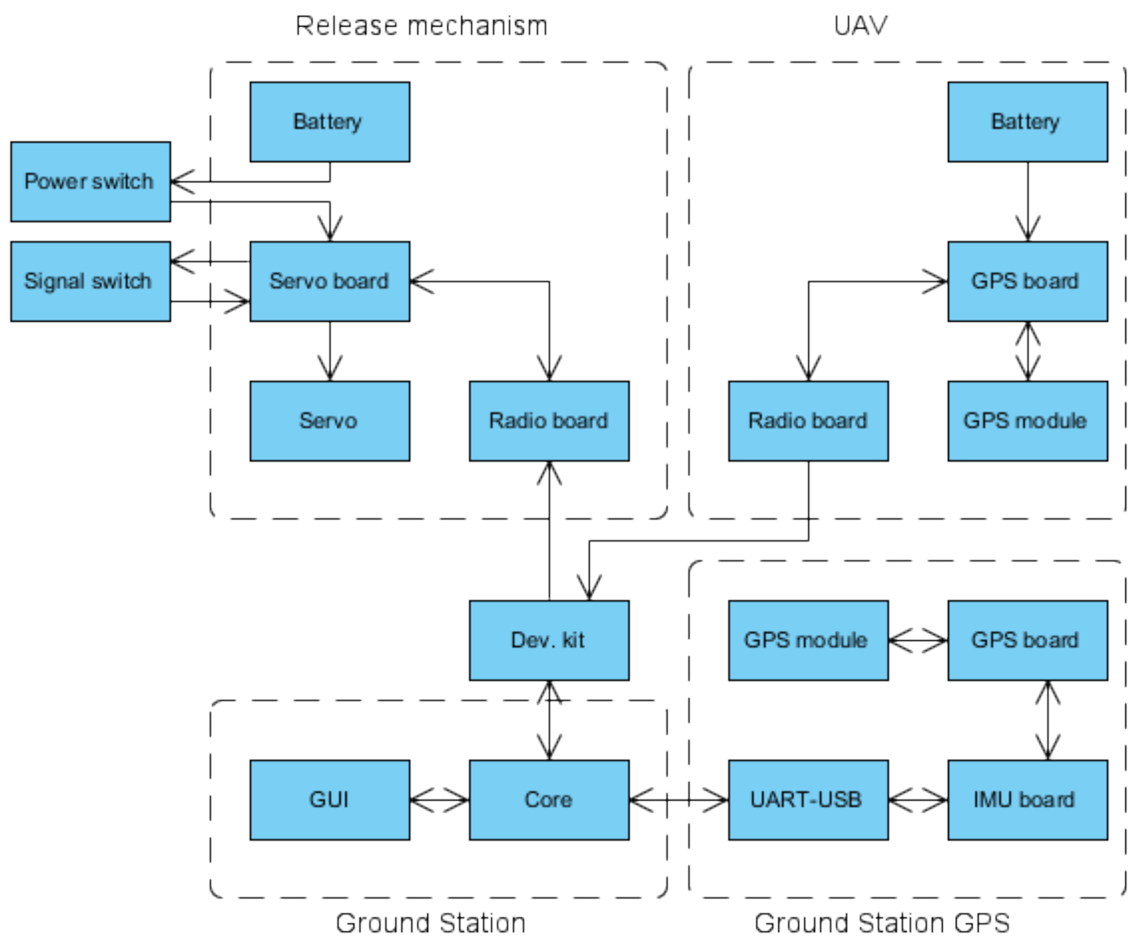


Figure 1: Shows the entire system

3. Ground station

From	*To*	*How*	*Type*	*Reference:*
Dev. kit	**Core**	Serial_Read::read_RL()	String^	See 1.1
GPS	**GPS card**	NMEA string	UART	See 1.2
GPS card	**IMU**	GPD	I2C	See 1.3
IMU	**Core**	Serial_Read::read_gps()	USB - Virtual COM port	See 1.4
Core	**GUI**	Socket	Char*	See 1.5
GUI	**Core**	Socket	Char*	See 1.6

3.1. Dev. kit-Core

Communication between the two is done over a COM to USB virtual COM port. The Core fetches data from the virtual COM port using Serial read method that returns a String^:

```
String^ Serial_Read::read_RL(){
      try{
            _serialPort2->Open();
            String^ rl_data_buffer =_serialPort2->ReadLine();
            return(rl_data_buffer);
      }
      catch (...){
            return "0";
      }
}
```

Data format:

“RL error code, RL number of satellites, RL time, RL Longitude, RL latitude, RL Altitude, RL direction , RL pos dilution ,xAcc, yAcc, zAcc, xGyro, yGyro, zGyro, roll, pitch yaw ”

E.g.

RL char*: “1, 940717, 9.64927, 59.68316, 192, 0.00, 0.00, 1.90, -1.76, -0.03, 10.69, 0.22, 0.19, -0.15, -1.72, 17.26, -20.88, 149, 178, 148\n” (1.1)

3.2. GPS – GPS card

The GPS module sends GPS information over UART. The GPS is set up to send only $GPGGA, $GPGSA and $GPRMC data using the standard GPS data format NMEA.

Data format:

NMEA 0183 (or NMEA for short) is a combined electrical and data specification for communication between GPS receivers and many other types of instruments. It has been defined by, the U.S.-based National Marine Electronics Association.

Bit rate	4800
Data bits	8
Parity	None
Stop bits	1
Handshake	None

E.g.

$GPGGA, 184343.000, 5940.10.58, N, 00939.1053, E, 1, 03, 3.0, 149.8, M, 41.0, M, , 0000*5A

$GPGSA, A, 2, , 12, 18, 15, , , , , , , , , , 3.2, 3.0, 1.0, *3E

3.3. GPS card – IMU

The GPS parses and creates a struct of the achieved GPS data. That data is then made available on the I2C bus. The IMU circuit then reads the I2C bus.

Data format:

```
typedef struct {
        float utc_time;                 // Time
        double longitude;               // Longitude in decimal degrees
        double latitude;                // Latitude in decimal degrees
        float altitude;                 // Altitude in meters
        float direction;                // Course over ground
        float speed_over_ground;        // Speed in knots
        float pos_dilution;             // Position Dilution
        float vdop;                     // Horizontal DOP
        float hdop;                     // Vertical DOP
        unsigned char pos_fix;          // Is position fixed?
        unsigned char sat_used;         // Number of satellites used
} gps_data_packet;
gps_data_packet gdp;

i2c_SSendBuffer( &gdp, sizeof(gps_data_packet));
```

```
gps_data_packet * recieved;

i2c_SRecvBuffer( inputdata,
I2C_RBUFFSIZE);

recieved = inputdata;
```

“GPS error code, GPS number of satellites, GPS time, GPS longitude, GPS latitude, GPS altitude, GPS direction, GPS position”
E.g.

GPS char*: “1, 7, 140716, 9.648758, 59.682964, 184, 5.1, 3.6”\n (1.2)

3.4. IMU – Core

Communication between the two is done over a COM to USB virtual COM port using a FT232. The IMU sends data every second, and the Core fetches data from the virtual COM port when there is a data request using a Serial read method:

```
String^ Serial_Read::read_gps(){
        try{
                _serialPort1->Open();
                String^ gps_data_buffer =_serialPort1->ReadLine();
                return(gps_data_buffer);
        }
        catch (...){
                return "0";
        }
}
```

Data format:

“GPS error code, GPS number of satellites, GPS time, GPS longitude, GPS latitude, GPS altitude, GPS direction, GPS position”

E.g.

GPS char*: “1,7,140716,9.648758,59.682964,184,5.1,3.6\n” (1.3)

3.5. Core-GUI

The data is transferred through a socket, within ServerSocket::ListenOnPort() method, under network event write and read the methods:

```
if (NetworkEvents.lNetworkEvents & FD_WRITE){
  if (NetworkEvents.iErrorCode[FD_WRITE_BIT]){
    cout<<"Could not write, error:\n" <<NetworkEvents.iErrorCode[FD_WRITE_BIT];
  }
  tempdata = mySerialData->Complete_data_send();
  buff = tempdata.c_str();
  send(SocketArray[Index - WSA_WAIT_EVENT_0], buff, 200, 0);
}
```

The GUI reads the receive network event and receives the transferred data string:

```
if (NetworkEvents.lNetworkEvents & FD_READ){
      if (NetworkEvents.iErrorCode[FD_READ_BIT]!= 0){
            MessageBeep(MB_ICONERROR);
            break;
      }
      memset(buffer,0,sizeof(buffer));
      recv(s,buffer,sizeof(buffer),0);
       . . .
```

Data format:

Char* = " GPS error code, GPS number of satellites, GPS Time, GPS Longitude, GPS latitude, GPS Altitude, GPS direction, GPS position , ,RL error code, RL number of satellites, RL time, RL Longitude, RL latitude, RL Altitude, RL direction , RL pos dilution ,xAcc, yAxx, zAcc, xGyro, yGyro, zGyro, roll, pitch yaw, servo channel 1, servo channel 2, servo channel 3 \n"

GPS char* (see 1.3) + RL char*(see 1.1)

E.g.
"1,7,140716,9.648758,59.682964,184,5.1,3.6,1,940717,9.64927,59.68316,192,0.00,0.00,1.90,-1.76,-0.03,10.69,0.22,0.19,-0.15,-1.72,17.26,-20.88,149,178,148"

3.6. GUI-Core

The data is transferred through a socket, within ClientSocket::ConnectToHost () method, under network event write and read the method SendData(char*):

Data format:

1: Char* = "Req" is sent when the GUI is done processing the data it received and is ready for more.

2: Char* = "Launch" is sent when the launch button is pressed.

4. Release mechanism electronics

From	*To*	*How*	*Type*	*See:*
Radio board	**Servo board**	I2C	Char*	See 1.2.1
Radio module	**Radio board**	UART	Char[]	See #

4.1. Radio board –Servo board

The data is transferred using Atmel's own I2C "lib"

Data format:

1. Char* = '0' sent when servo is to be closed.
2. Char* = '1' sent when servo is to be opened.
3. Char* = '2' sent when servo is to be open (stays open for one second) and closed.

4.2. Radio module to radio board

The data transferred using Atmel's own UART "lib".

Data format:

Radio board receives a char array, usually 1 byte long. Char[0] is checked.

5. Avionic

From	*To*	*How*	*Type*	*See:*
GPS	**Radio board**	I2C	Char*	See #
Radio Board	**Radio module**	UART	String	See#

5.1. GPS- Radio board

GPS

Data format:

"RL error code, RL number of satellites, RL time, RL Longitude, RL latitude, RL Altitude, RL direction , RL pos dilution ,xAcc, yAxx, zAcc, xGyro, yGyro, zGyro, roll, pitch yaw "

E.g.

RL char*: "1, 940717, 9.64927, 59.68316, 192, 0.00, 0.00, 1.90, -1.76, -0.03, 10.69, 0.22, 0.19, -0.15, -1.72, 17.26, -20.88, 149, 178, 148\n"

HØGSKOLEN
i Buskerud

PROJECT EVALUATION

LocalHawk

Jon Bjørnland

Erik Myklebust

Lars Erik Røise

Lars Hallvard Sannes

May 29th 2010

1. Table of Contents

2. Introduction

Our project assignment was to further develop an already existing system. Our work area on the LocalHawk system was reduced to the Ground Station and the Launcher, because of the systems size and complexity.

The Ground Station was to be rewritten in C++, the existing Ground Station had problems running real time data manipulation. The Ground Station should have a GUI that displays information.

Position data should be collected from both the aircraft and the Ground Station. Use a long-range radio link to send telemetric and position data down to a receiving unit that sends strings of data to the Ground Station. The Ground Station needed a GPS module to be able to achieve position. This position should be used to improve the aircrafts GPS precision trough the use of a DGPS

The launcher needed some improvements regarding the possibilities to launch other aircraft and also a functional release mechanism. The Wagon that accelerates the aircraft was only able to launch aircrafts with back mounted propeller, and it was preferred that it could be more modular. The release mechanism that was used on the launcher did not function as it was intended to. Triggering the system could only be done manually, and the possibility for failure was high.

3. Reached Goals

3.1. Goals Launcher

Wagon

1. Make the wagon capable of launching Siren airframe (front mounted thrust).

Release mechanism

2. Make a semi automatic release mechanism that works.
3. Improve the safety regarding premature launch.

3.1.1. Results

Marked with X:

Goal	Fulfilled	Partly fulfilled	failed
1		X	
2	X		
3	X		

Wagon

A new assembly part has been added into the wagon system. The reason for this part is to make the launcher able to handle aircraft with the propeller in front. The wagon part has settings that can be adjusted, so it can handle different aircraft of this type. There have not been any changes to the main wagon frame, other than this part that can be mounted on.

The goal has not been fully archived because of some difficulties in the launch sequence. This is because of the old wagons behavior when it hits the launcher frame. The aircraft gets thrown into wrong angle of attack, causing the aircraft to almost fall into the grown. This could have been foreseen by the results from the high-speed test, by looking closer into the launch sequence.

Release mechanism

The release mechanism that has been developed fulfills all its goals at a very satisfying level. Using some of the circuit boards developed by KDA and code inspired by other groups, the system has been made from scratch, without the use of parts from the old mechanism. Safety has been seen as very important during the develop phase. That is why the system is almost completely secure, and prevent premature launch under every assumption. The system is also safe when launch is preformed manually, but safer with semi automatic launch. The system is also safe when launch is preformed manually, but safer with semi automatic launch, using implemented electronics.

3.2. Goals Ground Station

Core application

1. Make an application in C++ that reads from the GPS and radio link module and relays that information to the GUI.

Graphical user interface

2. Make a GUI that displays information sent from the Core.

Radio Frequency communication

3. Implement electronics in the aircraft to gather GPS information.
4. Use a long-range radio link to send telemetric data down to a receiving unit that sends strings of data to the Ground Station.

Navigation

5. Add a GPS module to the Ground Station.
6. Improve the accuracy of the GPS position of the UAV.

3.2.1. Results

Marked with X:

Goal	Fulfilled	Partly fulfilled	failed
1	X		
2	X		
3	X		
4	X		
5	X		
6	X		

Core application

The Ground Station has been totally reworked. The new Ground Station is written in C++. It is divided into two sub programs, the Core and GUI, and utilizes a socket connection between them.

Graphical user interface

The GUI is an interface that shows information collected from the UAV and the Core application that acts like a data relay, which forwards the COM port data to the GUI.

Radio Frequency communication

GPS position data is now sent from the aircraft, down to the Ground Station via a long-range radio link together with simulated IMU data. The IMU data is simulated because an IMU was not available.

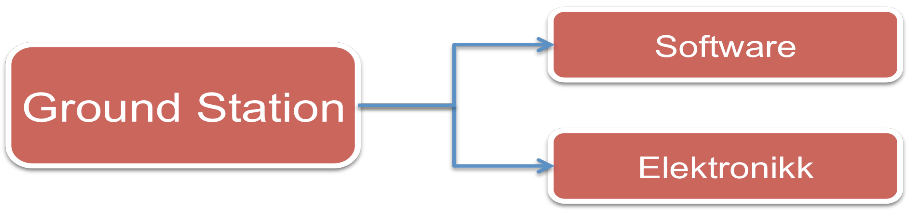
Navigation

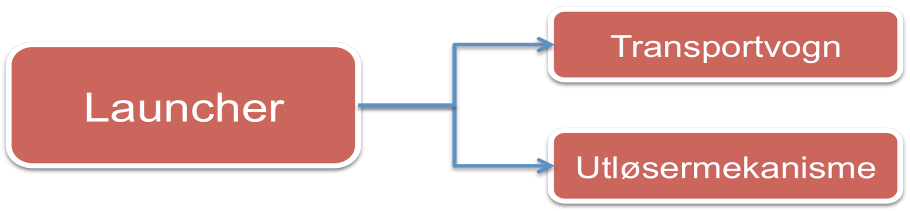
Position improvement has been achieved by implementing a DGPS system for the UAV and a position filtering to acquire the Ground Station's stationary position for the DGPS.

4. Project development

4.1. Working method

During the start phase of this project we divided the project into two main areas, with corresponding two fields.





4.1.1. Area of responsibility

Ground station:

Software – Lars Erik Røise and Jon Bjørnland

Electronic – Jon Bjørnland and Erik Myklebust

Launcher:

Wagon – Lars Hallvard Sannes

Release mechanism – Erik Myklebust and Lars Hallvard Sannes

Every group member has had responsibility for some areas, meaning that they are responsible for the task. This does not mean that this is the only things they are suppose to focus on, but that they had the superior responsibility.

4.1.2. Teamwork

In the early start phase of the project, we come to an agreement on how to execute the project. We also made some rules for the project participant. They were made for archiving best efficiency, and also give us some restrictions.

Example: If one of us came 10 minutes to late, this person had to pay 10 NKR in a common "Ice cream fund (NO: isfond) as a punishment.

Every group member had office together, and used this as the work place for the project. We found out that when we were sitting together, we cooperated better and everybody gave each other constructive feedback.

4.1.3. Work schedule

In the start of the project we made a time schedule, dividing the project into small tasks. We also made a document that showed everybody activities ahead. We did not manage to follow this exactly because of miscalculations etc. We also varied a bit in what we worked on. If we had started on one assignment, but had to wait before it could be finished, we continued on another task meanwhile. This method worked very well for us, and it provided that we had something to do at all time.

4.1.4. Teamwork with supervisors

Our external supervisors given from KDS have given us help and input on our work. We had several supervisors, with different qualification, covering almost every needed area. This has been very helpful. If something was not clear or we needed some help. On the other hand we have had some difficulties with AVR programming, which we used on the microcontroller. This is a area which we were very fresh on. We did not have any experience using this, and it was hard to find someone that could. So the execution of this took a lot of time, testing and failing.

We were a bit unhappy with the help from our internal supervisor, regarding help and input on the documentation.

5. Conclusion

The project was carried out as planned. Almost every goal was reached, and every part in the system did function together, as intended. The information from the aircraft could be seen at a computer at all time. The GPS showed a more accurate position. The launch could be performed from the Ground Station computer, by pushing a button for releasing the wagon.

We feel that during this year our competence has increased a lot and we know more about teamwork. Also learning new things and working on areas outside our education field. On the other hand we feel that documentation has been a too big part of the project, and this has led to less time on system development. This is something that we also think that our project supplier agrees on.

HØGSKOLEN i Buskerud

PROJECT PLANNING DOCUMENT

LocalHawk

Jon Bjørnland

Erik Myklebust

Lars Erik Røise

Lars Hallvard Sannes

January 8th 2010

Summary

This report presents the work done by Lars Erik Røise, Erik Myklebust, Jon Bjørnland and Lars Hallvard Sannes during the third year of the bachelor degree at Buskerud University College fall 2009.

This document contains most of the plans and information about the upcoming project. First, the objectives give an impression of what the different activities are meant to accomplish and why. This part also includes a prediction of the time spent on each activity.

The rest of the document shows an elementary activity network, a general idea over each activity's responsible person, the resource plan, budget, and information about the participating project group members.

There are three attachments to this document, which are; document overview, personal time schedule, and the carefully planned time schedule.

Table of Contents

Introduction

We contacted Kongsberg Defence System in search for an interesting bachelor assignment and they were enthusiastic to include us in the LocalHawk project. LocalHawk is a student project that has been in progress over several years and constantly developed.

The penultimate semester, our project group has been preparing for the main project. When dealing with a project of this magnitude, it is essential to be well prepared in advance to prevent unnecessary surprises along the way. As many obstacles as possible should be revealed before heavy processes have been started. Problems appearing later on are much harder to reverse as time goes by.

This project contains about 20 activities. Most project groups work by the Prototype project model, as does ours, often experience unforeseen difficulties that results in reconsideration and maybe also re-planning. The project group's ability to be dynamic to changes is greatly appreciated when being evaluated. A carefully planned schedule to early detect activities slipping is also highly valued. Therefore, nearly all the activities are divided into sub activities to make planning, follow-up, and revision easier.

Restrictions

This project has a few restrictions in terms of the execution.

It is time restricted, the project have to be finished by May 17th. It is also financially restricted in the sense that the project cost needs to correspond with the parameters set by the budget

The projects objectives is constrained by the specifications of requirements

KDS have given us a software restriction, they do not want us to use programs like Microsoft projects or other closed proprietary formats. They also do not want us to publicize some of our work and any of the program code on the Internet.

Broad objectives

The C++ compiled ground station should show all needed information such as position, velocity and heading of the aircraft as well as all of the today's parameters. This should be updated frequently. To make the GPS coordinates on the aircraft position more accurate; a D-GPS needs to be implemented. Using some form of connection, the ground station should talk to the launcher and sending the launch signal to the trigger on the launcher. The trigger system will need to be redesigned for more robustness and automatic use. The today's wagon system should be redesigned for the other airframes in the localhawk project.

Objectives

We have divided the project into three categories: Ground station, launcher and administrative. These categories are divided into activities. The following section will describe these individual activities in greater detail.

Under each of the activities you will find the person responsible for activity and the total man-hour sum of the activity. This is because some personnel are working part-time on several activities simultaneously.

Under each of the sub-activities you will find estimated start, which indicates the estimated time of when the activity will be initiated, when the activity is expected to be finished, and the time duration of the activity.

For reference in the time schedule and future timesheets the activities is numbered accordingly: main group; GRS – Ground station LSH – Launcher or SEG – Administrative. Followed by the objective name, then the activity number (1, 2, 3 etc) and at last a sub-activity letter (a, b, c etc).

I.e. the activity number of first sub-activity under the first activity on the ground station is: GRS-COR-1a.

Core application

Head of activity: Lars Erik Røise
Predicted total man-hours: 171 hours

Introduction

Today's application is only a prototype, it's written in Java, and has the necessary functions to fly the plane. However the application does not have the required real-time capabilities, to processes the information. In order to correct that, the core of the application is to be written in C++.

Activities

Remake application core.

GRS-COR-1

a. Estimated duration is Des. 16th to Jan. 13th (duration: 8 days; note: holidays).
   - Write a new version of the application in C++

b. Estimated duration is Jan. 14th to Feb. 8th (duration: 18 days).
   - Make the needed parts of the application work in real-time.

c. Estimated duration is Feb. 9th to Feb. 18th (duration: 8 days).
   - Implement the Java GUI.

d. Estimated duration is Feb. 19th to Feb. 28th (duration: 6 days).
   - Test the core application according to the test specification. The work done and the test ought to be documented.

Differential Global Positioning System (DGPS)

Head of activity: Jon Bjørnland
Predicted total man-hours: 324 hours

Introduction

Today there exists a GPS (Global Positioning System) unit on the aircraft. The accuracy of this unit is not on an acceptable level due to interference and GPS deviation. To achieve higher accuracy there were put forward a request to implement a "D"GPS ("Differential" Global Positioning System) to the ground station.

Concept

The solution we are to implement will not be a true differential GPS hence the quotation marks. A true differential GPS would use both the known position and the GPS position of the ground station to calculate the GPS deviation. Then use this information and the presumption that two GPS modules have the same deviation to subtract the same GPS deviation from the UAV's GPS position to get a more precise position.

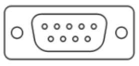
Our solution will only use the GPS position of the ground station and the GPS position of the UAV to calculate the UAV's position relative to ground station location. This will henceforth be the definition we will use of Differential Global Positioning System or DGPS without quotation marks.

This essentially will say that the ground station will be the origin in a coordinate system. And the UAV will get its coordinates from that system. This does not by it self provide a more accurate position. But when the runway and other targets are plotted into the same coordinate system we start to see improvement in waypoint execution and landing.

But before this can be done, we need to interface the GPS to the ground station computer. To interface the GPS antenna module that is going to be used, the EM406a (see picture), the signal needs to convert the signals to something a computer can understand.

Concept – Interface 1

One way to do this is to transform the output signal to a RS232 serial communication protocol through a DE-9 connector.



DE-9 connector

The problem with this solution is that the RS232 through a DE-9 connector is a outdate communication protocol that needs an adapter if you want to connect it to any modern computer

Concept – Interface 2

Another way is to transform the signal to a universal serial signal connected through a USB port.

A USB port

A USB port is found on all modern computers.

Activities

Due to the size of this objective and conflicting critical path, we have divided the problem into three sub activities. These also make it easier to estimate the time, work load, and follow up and correct deviation.

Creating a interface circuit for the GPS antenna

GRS-GPS-1

The first activity that needs to be accomplished is to make a GPS peripheral that connects to the ground station through some sort of I/O port. This task can be divided into four sub tasks:

a. Estimated duration is Des. 16th to Jan. 15th (duration: 10 days; note: holidays).
   - First the design the circuit. How does one get the GPS antenna to send signals a computer can understand?

b. Estimated duration is Jan. 18th to Jan. 20th (duration: 3 days).
   - Then there is the task of creating a physical circuit board, which can be used to solder the components onto.

c. Estimated duration is Jan. 21st to Feb. 2nd (duration: 9 days).
   - Solder the components onto the circuit board that was made.

d. Estimated duration is Feb. 3rd to Feb. 12th (duration: 10 days).
   - The circuit board needs to be analyzed, and fix faults to prevent a possible faulty board from frying the GPS antenna module when it is connected.

Interfacing the GPS

GRS-GPS-2

After the GPS is connected to the ground station, we have to make the GPS position data available in the ground station code. This cannot be done until the core application is finished.

a. Estimated duration is Feb. 15th to Mar. 3rd (duration: 13 days).
   - Write into the core application the option to read the I/O port, which the GPS peripheral is connected to.

Make the DGPS code

GRS-GPS-3

At last, it needs to be created a DGPS code that calculates the UAV's position relative to that of the ground station.

a. Estimated duration is Mar. 4th to Mar. 26th (duration: 17 days).
   - Write an algorithm that converts the coordinates into relative coordinates and implement this into the GUI and core application.

b. Estimated duration is Apr. 5th to Apr. 8th (duration: 4 days).
   - Test the DGPS according to the test specification. The work done and the test ought to be documented.

Wireless link

Head of activity: Erik Myklebust
Predicted total man-hours: 145 hours

Introduction

The communication between the ground station and the aircraft is done through the wireless link (RC1200MC), which is already in place. This is link sends all information containing coordinates, heading reference values, servo values, etc. The ground station and aircraft are always trying to send this information, but the different packages do not have the same priority.

The current solution has one major defect. The ground station and the aircraft are constantly sending the information produced through the wireless link. The result is heavy traffic that if not constrained. When the information flow gets too great, the wireless link may shut down. It is of most importance that this does not happen, since a crash into the ground might break costly components.

Activities

Prevent the wireless link from collapsing

GRS-WRL-1

a. Estimated duration is Des. 16th to Jan. 15th (duration: 10 days; note: holidays).
   - The different kinds of data that is being transferred needs to be sorted and categorized on importance into a queue.

b. Estimated duration is Jan. 18th to Feb. 2nd (duration: 12 days).
   - A suiting algorithm has to be found. This needs to be compatible with the ground station core.

c. Estimated duration is Feb. 3rd to Feb. 10th (duration: 6 days).
   - Implement the algorithm. The data vital to hold the aircraft safe in the air needs to be processed constantly. The other less vital information should be held back. The baud rate must be limited to just below the RC1200MC's baud rate.

d. Estimated duration is Feb. 11th to Feb. 18th (duration: 6 days).
   - Test the wireless link according to the test specification. The work done and the test ought to be documented.

Graphical User Interface (GUI)

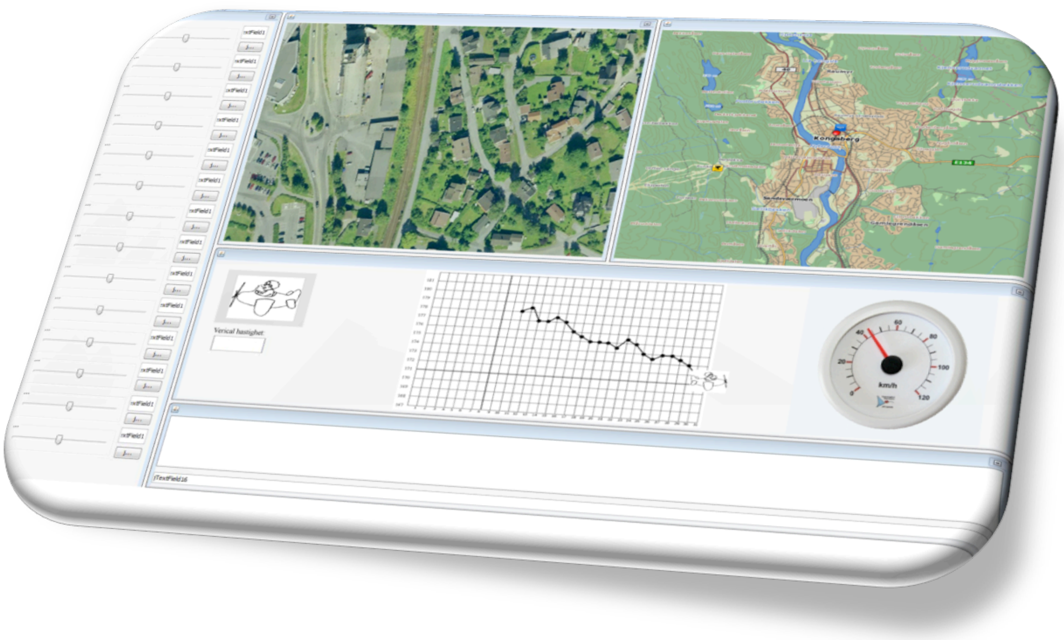
Head of activity: Lars Erik Røise
Predicted total man-hours: 172 hours

Introduction

One of the tasks the user interface has is to display information about the aircraft; display video sent from the aircraft and possibility to override the autopilot form the ground. The GUI given to us is only a first version and has great room for improvements in every way.

Concept

Our solution will display more information about the plane. The layout and overall look will be something like this:



Activities

Improving the GUI

GRS-GUI-1

a. Estimated duration is Mar. 1st to Mar. 30th (duration: 22 days).
    - Implement additional functions: a map, horizontal and vertical speed, vertical acceleration and height indicator.

b. Estimated duration is Apr. 21st to Apr. 28th (duration: 6 days).
    - Improve the layout of the GUI.

c. Estimated duration is Apr. 29th to May. 4th (duration: 4 days).
    - Test the GUI according to the test specification. The work done and the test ought to be documented.

Release mechanism

Head of activity: Erik Myklebust
Predicted total man-hours: 338

Introduction

The reason we have to make improvements is that the servo controlling the launch is broken and needs to be replaced by a stronger servo. Also, the weakness of the solution provided is that it is unsafe and not easy to use. When the wagon is to be locked in the trigger, this needs several difficult procedures. Additionally, it does not feel safe to be in front of, or close to, the trigger while the cords are in tension. This is because of the fear of unexpected release caused by a sliding lock mechanism. The trigger needs a safer, stronger and more permanent trigger.

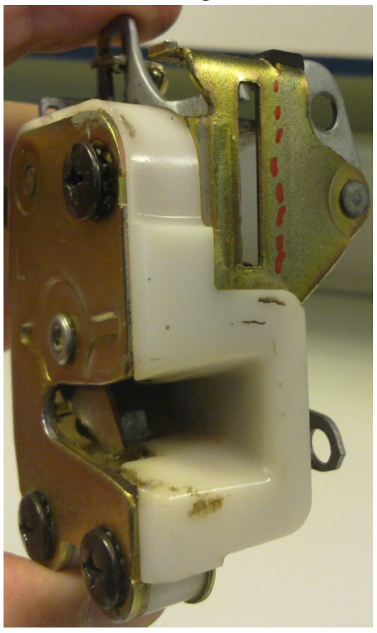
Concepts

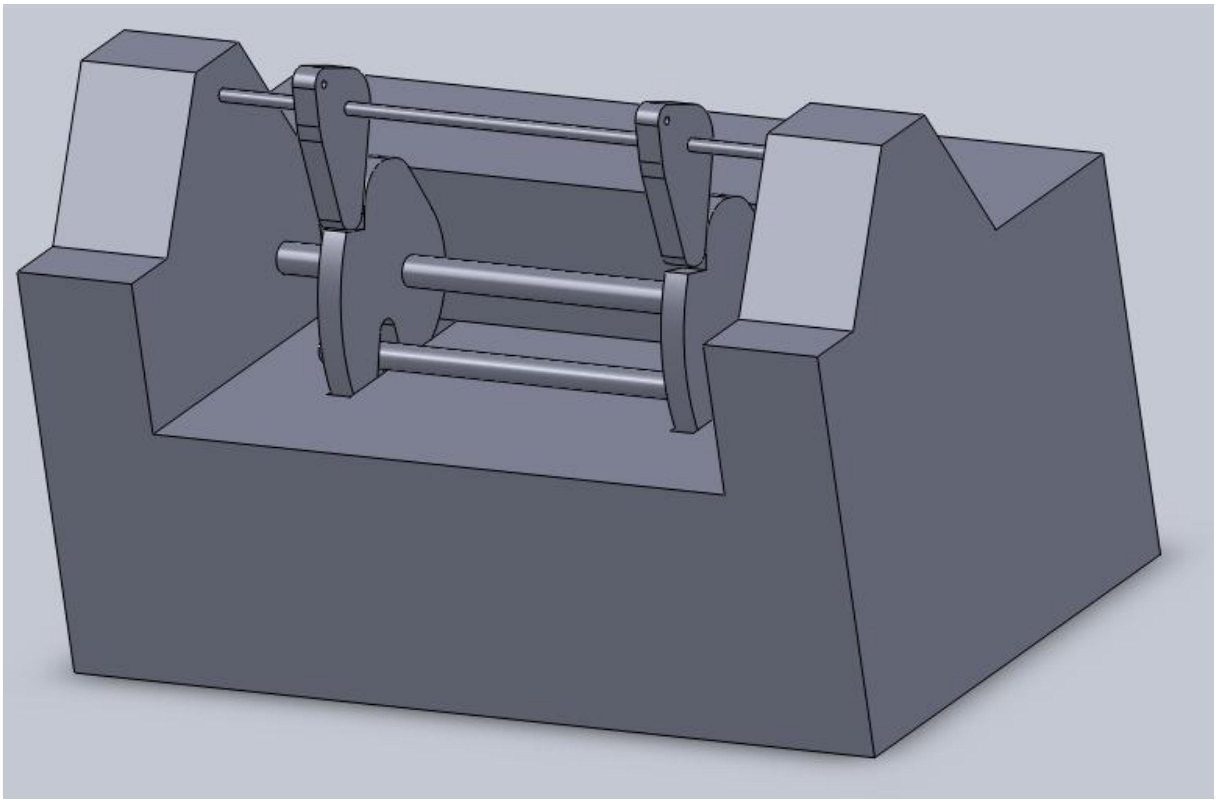
Most of the release mechanism will work like the solution from the project done summer of 2009:

- The wagon will be pushed back into the trigger.
- The trigger will hold the wagon in place while the bungee cords are winched into tension.

The difference will be the mechanism that holds the wagon in place. This will be greatly improved with a new and better technology. With the improved solution it will be easier to operate while launching. The solution will also make the release mechanism stronger and longer lasting.

All of the following concepts are based on a two-point holder placed in a line. When the bar is manually pulled back into place the holders are locked. This idea is greatly used in the car industry. In every car door there is a holding mechanism that is released when somebody uses the handle. Since the car industry is build upon reliance, usability and cost, this widely used concept is probably the most advantageous solution for holding and releasing medium weighted load. Using a servo requires that triggering the release takes little force. The mechanism is released either manually by hand or using a servo. If using a servo, the needed force necessary will have to be below the servo ability.


Shows a car door latch that can be used.



Shows a draft of how the release mechanism might be constructed. In short, it is a simplified car door latch with an option of connecting a servo (actuator). The holder is connected to a spring and will try to push upwards. When the servo pulls on the locking mechanism, this will release the wagon (the lower bar). The advantage with this concept is the low power required.

Concept 1

Using a manually triggered release mechanism will make an easier solution with good strength and usability. One way to accomplish this is to join a lever to the wagon holder. Pull the lever back to release the wagon.

Concept 2

The solution made in the previous project used a button to drive a servo that released the wagon. This system is easy to copy and improve. The figures on the right show some sketches that have been made of the holding device triggered by a servo.

Concept 3

This solution is more advanced making it possible to release the mechanism from ground station using a connection to the launcher. This is build upon the 2.concept. This solution eliminates the need for a person dedicated to launch. With this solution, more resources can be spent on timing, catching, recording, measuring, and controlling the aircraft.

Either concept; a strain measurer should be implemented to make correct readings on force from wagon, estimated energy built up, or predict exit velocity of the aircraft.

Activities

Make a release mechanism:

LSH-TRG-1

a. Estimated duration is Feb. 19th to Feb. 26th (duration: 6 days).
    - Look into different kinds of triggers. The trigger found needs to work well while under relatively high tension. It will hopefully later be controlled by a servo and therefore has to be triggered with low force.

b. Estimated duration is Mar. 1st to Mar. 30th (duration: 22 days).
    - Design the trigger. The trigger will have to take into account how the wagon is designed.

c. Estimated duration is Apr. 21st to Apr. 28th (duration: 6 days).
    - Build the release mechanism and install it on the launcher.

d. Estimated duration is Mar. 29th to May 4th (duration: 4 days).
    - Test the trigger according to the test specification. The work done and the test ought to be documented.

Wagon System

Head of activity: Lars Hallvard Sannes
Predicted total man-hours: 337 hours

Introduction

There are two airframe models for the Localhawk project. One is the Stryker F-27C by Parkzone; the other is a Siren model by Great Planes. The Launcher is specifically designed for the Stryker model, which is rear powered with a 6-inch (15.24cm) propeller. The Siren model is front powered with a huge 13-inch (33.02cm) propeller, this model is not possible to launch today with today's launcher.

We want to make it possible to launch both of these aircraft from this launcher. This means modifying or redesigning the wagon that holds the airframe during the launch sequence. We also want to make some small improve to some parts on the launcher to achieving an easier launch sequence.

Concept

Today's wagon solution is only a prototype; consequently it has room for some improvements. The wagon shall also, as previously stated, be able to fit the two very different types of aircrafts. To accommodate for this, the propel size needs to be taken into account. When the aircraft is launched, the propeller may strike the end of the launcher frame damaging the propeller and jeopardizing the flight. The aircrafts does not only have a different in placing and size of the propeller, the airframes are also very different. Therefore we have to come up with a good solution to the problem.

Concept – wagon 1

One of our solutions is to first optimizing the old wagon for the Stryker airframe, and then design an adapter you can mount on the wagon to make it possible to launch the Siren airframe. The problem with the Siren aircraft is the large wingspan and the large front mounted propel. Therefore the new part has to be wide, for keeping the aircraft steady. It also needs to raise the angle of attack, preventing the propeller from striking the frame.

Concept – wagon 2

Another way to solve the problem with two different aircraft types is to create a new wagon that can manage to launch them both. Then we need to come up with a completely new wagon design. A design that still can launch the Stryker F-27c but also manage the Siren without modifications in between the different launches.

Activities

There are two main activities to this objective. One will be to design and build a prototype of the wagon. The other will be to make some small adjustments to the launcher itself, like applying more break force, some more robust bungee cords and other easy modifications may also be performed, like changing to lock nuts.

Create a new wagon or modify the old wagon

LSH-WGN-1

a. Estimated duration is Jan. 15th to Jan. 22nd (duration: 6 days).
   - Find out what is needed to support each airframe and find a way to incorporate the needs of both airframes in one design.

b. Estimated duration is Jan. 25th to Feb. 2nd (duration: 7 days).
   - Create the new wagon design in a CAD.

c. Estimated duration is Feb. 15th to Mar. 11th (duration: 19 days).
   - Build a new wagon or modify the old.

d. Estimated duration is Mar. 12th to Mar. 18th (duration: 5 days).
   - Fit a more permanent vertical airframe support and find a new way to mount the rubber bands if the old wagon is chosen.

Improve parts on the wagon for better use and safety

LSH-WGN-2

a. Estimated duration is Feb. 3rd to Feb. 10th (duration: 6 days).
   - Calculate the needed break force to stop the wagon before hitting the launcher frame and come up with a solution that will prevent the wagon from hitting the frame.

a. Estimated duration is Feb. 11th to Feb. 12th (duration: 2 days).
   - That Look at new rubber bands that are stronger and more responsive when triggered. Find out if it will improve the system.

b. Estimated duration is Mar. 30th to Apr. 24th (duration: 5 days). Note: Holiday.
   - Do some test and find the spring-constant of the different kinds of rubber bands and compare it to the old rubber bands.

b. Estimated duration is Apr. 26th to May 4th (duration: 7 days).
   - Test the Wagon according to the test specification. The work done and the test ought to be documented.

Computer aided design (CAD)

Head of activity: Lars Hallvard Sannes
Predicted total man-hours: 45 hours

Introduction

There already exists a CAD drawing of the launcher. The existing drawing is made in a 3D design program called CATIA. The existing sketches are of an earlier version of the created launcher. Thus they are not accurate enough to do sufficient analysis of. That is why we need to create a new CAD drawing and do some simulated tests of it.

The old drawing is too imprecise and outdated to get information regarding its abilities and boundaries. Therefore we need to create a new and more precise CAD drawing of the launcher. This is get information about its boundaries regarding stresses and strength.

Activities

Create a new CAD drawing

LSH-CAD-1

a. Estimated duration is Des. 16th to Jan. 12th (duration: 7 days; note: holidays).
    - Measure the launcher and draw it in a CAD program.

b. Estimated duration is Jan. 13th to Jan. 14th (duration: 2 days).
    - Document the work done.

Administrative

Introduction

There is a lot of documentation and other administrative work that needs to be done in this project. All administrative activities have an activity number that starts with SEG. This section will explain all the administrative work that will be completed throughout the coming semester.

Activities

As this section does not describe a single activity, it has been divided into activity portions.

Create a user manual for the ground station and the launcher.

SEG-MAN-1/2

Head of activity SEG-MAN-1: Lars Erik Røise
Head of activity SEG-MAN-2: Lars Hallvard Sannes
Predicted total man-hours: 29 hours

The project group is going to make a step-by-step user manual for the ground station. The launcher already has a step-by-step assembly instruction but this needs to be greatly improved with respect to the work done by our group.

Why

The ground station as of November '09 has no explaining documentation to assist the in the further development or usage of the code. This is why KDS have requested a step-by-step user manual to the ground station.

To keep the launcher's instruction manual still valid, the project group need to update it with the latest changes.

Activities

1. Estimated duration is May 12th to May 14th (duration: 3 days).
   - Write a user manual for the ground station.

2. Estimated duration is May 12th to May 14th (duration: 3 days).
   - Write a user manual for the launcher.

Make a movie
SEG-MOV-1

Head of activity: Jon Bjørnland
Predicted total man-hours: 34 hours

The project group is going to make a movie, which depicts our work throughout the year.

Why
KDS have requested a movie, which they can use to promote the project to future student.

Activities
a. Estimated duration is May 12th to May 13th (duration: 2 days).
    - Gather movie clips taken throughout the year and find appropriate music. Edit together a final product.

b. Estimated duration is May 14th to May 14th (duration: 1 day).
    - Document the work done.

Create a poster
SEG-PST-1

Head of activity: Erik Myklebust
Predicted total man-hours: 34 hours

Design and create an A2 poster in style with previous KDS posters, which shows of our work and project group.

Why
The poster is part of the compulsory documentation the university college demands. Additionally KDS has expressed a desire for a poster to use as promotional material for the LocalHawk project.

Activities
a. Estimated duration is May 26th to May 27th (duration: 2 days).
    - Take pictures of the finished product and some of the work done and compose a poster in A2 format.

b. Estimated duration is May 14th to May 14th (duration: 1 day).
    - Document the work done.

Make a key number document
SEG-KEY-1

Head of activity: Jon Bjørnland
Predicted total man-hours: 4hours 48minutes

Create a document that contains key numbers for the entire project e.g. spring constant for the rubber bands, airframe, sizes, weights, lift and drag constants.

Why
Once we accepted this project assignment we got a "LocalHawk starter pack". This contains old reports, document guidelines, and an inventory list.

The project group finds that a key number document is something that would be very handy for future work, and should be included in the "LocalHawk starter pack".

As it is today, if one want to know, e.g. the spring constant of the rubber band, one has to scroll through several reports if one does not know exactly where to look. This is a waste of time.

Activities
a. Estimated duration is Apr. 26th to Apr. 26th (duration: 1/2 day).
    - Find all the data.

b. Estimated duration is Apr. 26th to Apr. 26th (duration: 1/2 day).
    - Write a document.

Presentation preparations
SEG-PRP-1

Head of activity: Jon Bjørnland
Predicted total man-hours: 160 hours

We are to prepare for the presentations in January and April/May.

This is not an objective that KDS have required but has been created as an activity to prevent unexplained voids in the time schedule.

Why
We are to inform the university college on our progress and plans for further work. The presentations are also part of the evaluation process and count for 25% of the grade given.

Activities
a. Estimated duration is Jan. 4th to Jan. 8th (duration: 5 days).
    - Prepare for the first presentation in January.

b. Estimated duration is Mar. 31st to Apr. 2nd (duration: 3 days).
    - Prepare for the second presentation in Mars.

Complete documentation
SEG-DOC-1

Head of activity: Jon Bjørnland
Predicted total man-hours: 184 hours

Write accountancy to document the financial result and complete the final report.

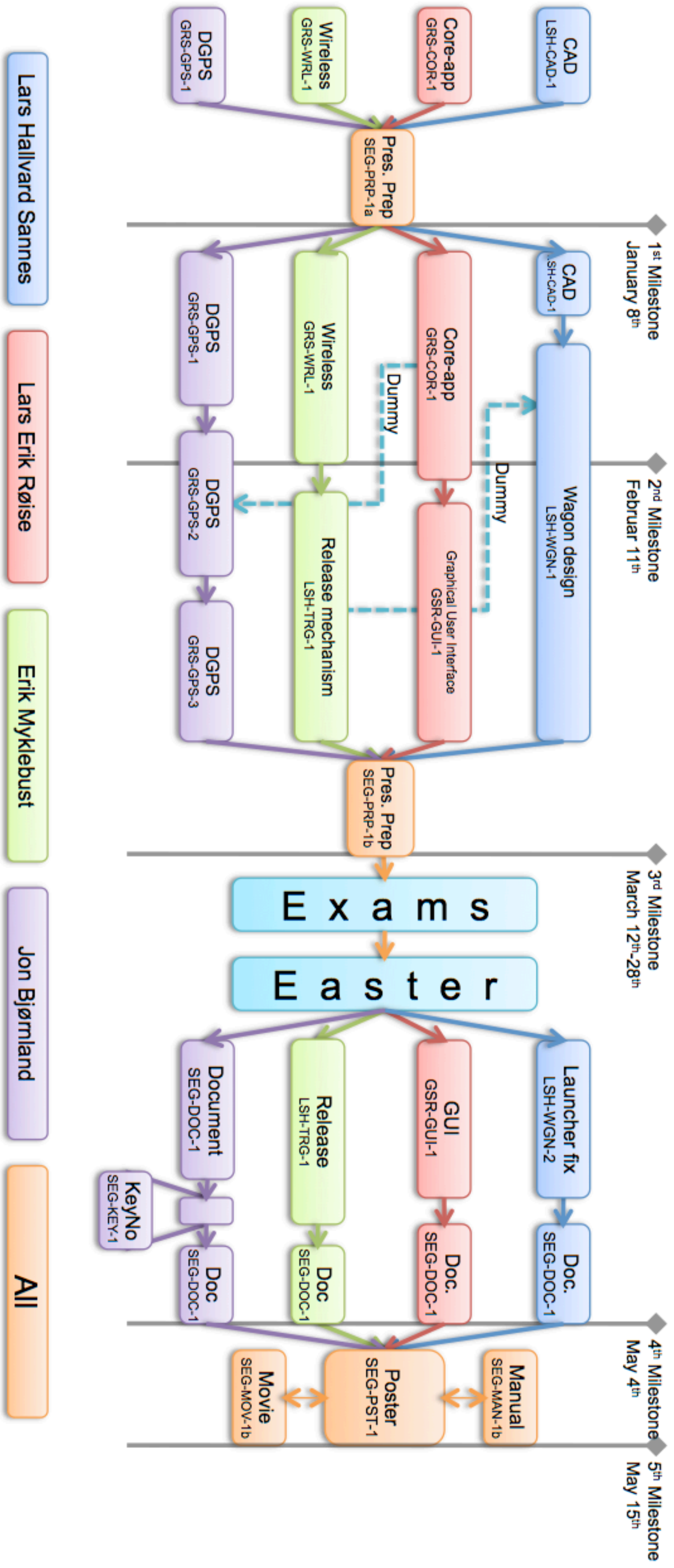
Why
To explain the work done for the next groups to work on this project we document the needed information about the products and the development of them. Documentation is also part of the evaluation process and the final report counts for a total 50% of the grade given.

Activities
a. Estimated duration is Apr. 21st to Apr. 23rd (duration: 3 days).
   - Write accountancy.

b. Estimated duration is Apr. 27th to May 11th (duration: 5 days).
   - Put all the documentation together in one final report.

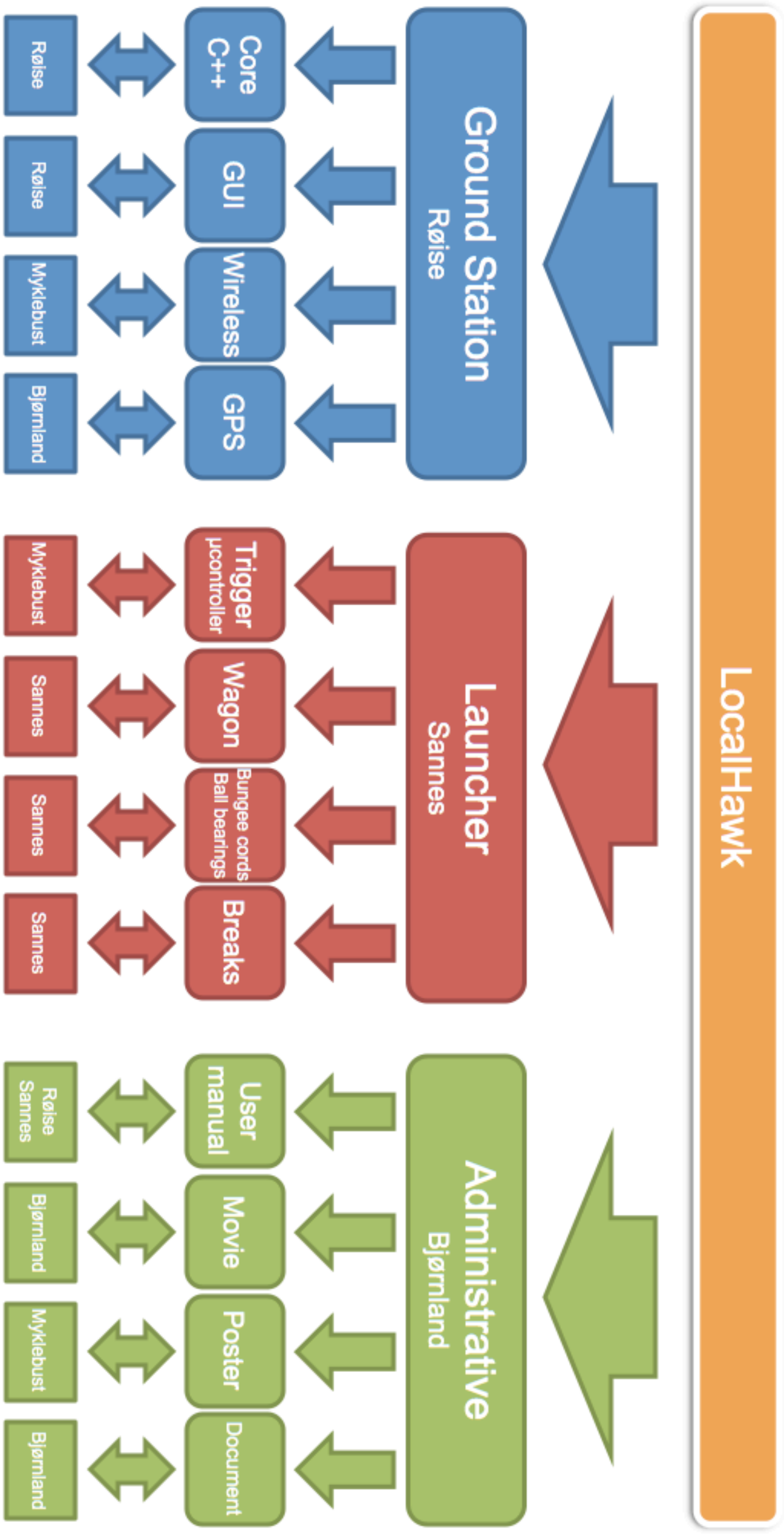
Project Network

This is a flow chart, which depicts the projects workflow.



Responsibility map

This is a graphical representation of responsibility distribution throughout the project.



Budget

This budget shows the expected project expenditures. There are two budget proposals one with wagon materials and one without wagon materials (left and right column respectively).

The reason for this is that KDS was uncertain of the amount of materials available is stock.

LOCALHAWK- BUDGET:

Post	Total
Budgeted, KDA	3000,00
Total	3000,00 NOK

Post	Total w/o materials	Total w /materials
Launcher		
Wagon materials	0,00	800,00
Rubber band	300,00	300,00
Bracket	50,00	50,00
Fasteners	30,00	30,00
Basic 16x2 Character LCD - 5V	175,00	175,00
Battery	20,00	20,00
Trigger		
Strain measurer	250,00	250,00
Cable between aircraft and Launcher	100,00	100,00
Metal Parts, trigger	300,00	300,00
Microcontroller (minimum 14 digital and 1 analog I/O)	50,00	50,00
LM1117 5V regulator	10,00	10,00
Ground Station		
GPS interface circuit	250,00	250,00
Team bulding	365,00	365,00
Total procurement	1900,00 NOK	2700,00 NOK
Tax base	1425,00	2025
Tax	475,00	675,00
Shipping	300,00	300,00
TOTAL, LOCALHAWK- PROJECT	**800,00 NOK**	**0,00** NOK

About us

Jon Bjørnland
Field of study: Mechatronics
Phone: 93609860
E-mail: bjornland@gmail.com
Area of expertise: Reverse engineering
Group function: Project leader – This person is in charge of the administrative work surrounding the project. He is also responsible for the GPS and all the activities associated with this system.

Erik Myklebust
Field of study: Mechatronics
Phone: 93838712
E-mail: erikmykl@gmail.com
Area of expertise: Theory
Group function: Mechatronical engineer – This person has the task of developing the release mechanism on the launcher as the main responsibility. After completing this, he will concentrate on the communication between launcher and ground station.

Lars Hallvard Sannes
Field of study: Product design
Phone: 90884799
E-mail: larshsann@hotmail.com
Area of expertise: Problem solving
Group function: Mechanical engineer – This person is responsible for the launcher as a whole. He has also been given the task of designing the improved wagon system.

Lars Erik Røise
Field of study: Simulation and game development
Phone: 40610744
E-mail: roise88@gmail.com
Area of expertise: Troubleshooting
Group function: Computer engineer – This person has the ground station as his main responsibility and will be converting the core application from JAVA to C++ as well as further development on the GUI.

HØGSKOLEN
i Buskerud

RISK ASSESSMENT REPORT

LocalHawk

Jon Bjørnland
Erik Myklebust
Lars Erik Røise
Lars Hallvard Sannes

March 10th 2010

Table of Contents

Introduction

This document contains the risk analysis of the LocalHawk project spring 2010 its purpose is to map out critical objectives and what the consequence would be if one failed to comply with that specific requirement.

It is built up with each requirement being assigned a threat value and a risk value. The Threat value is based on the importance of that requirement being met. The risk value is a representation of how difficult the requirement is to meet

Risk	Low	Medium	High
Threat	Low	Medium	High

Since avoiding risk is almost impossible, good planning and analysis is required to minimize it. A requirements that is both difficult and important to achieve, (I.e. Risk; High and Threat; High). Requires frequent and close supervision to assure a successful completion. But a low, low, on the other hand requirement require less frequent and close supervision

Ground Station

GRS

GRS-COR – Core Application

GRS-COR

REQ-COR-1	**The Core application design**	
Requirement	**Pri**	**Description**
REQ-COR-1.1	1	The Core shall be written in C++
REQ-COR-1.2	1	The core application shall not be dependent on any other part of the ground station software. *This means that the core shall continue to run even if the GPS and/or the Wireless link is disconnected and any software crashes, other software includes GPS, wireless link drivers and/or GUI.*
REQ-COR-1.3	1	There shall be defined a interface between the peripheral circuit and the core
REQ-COR-1.4	2	There shall be a socket interface between GUI and the core

Requirement	**Risk**	**Threat**	**Consequence**
REQ-COR-1.1	Low	High	Everything related to the Core application is dependent on this requirement being achieved
REQ-COR-1.2	Low	Low	REQ-COR-1.4 will become unachievable
REQ-COR-1.3	Low	Low	The next group will struggle to understand the communication layout
REQ-COR-1.4	Medium	Low	None

GRS-GUI – Graphical User interface

GRS-GUI

REQ-GRS-1 The GUI design		
Requirement	**Pri**	**Description**
REQ-GUI-1.1	1	It contains a map; the map shall enable the user to set waypoint for the aircraft.
REQ-GUI-1.3	2	It contains a console; the console shall give messages and error messages as well as take commands from the user.
REQ-GUI-1.4	2	It contains a graph that can show a statistics of different logged information chosen by the user.
REQ-GUI-1.5	2	It is divided into tabs for each task; *For example map, video, console will have its own tab.*
REQ-GUI-1.6	3	It contains possibilities for user defined settings; *Like the user shall be able to choose whether the map should follow the aircraft or not.*

Requirement	**Risk**	**Threat**	**Consequence**
REQ-GUI-1.1	Low	Low	The solution will not contain a graphical representation of the position
REQ-GUI-1.3	Low	Low	None
REQ-GUI-1.4	Medium	Low	None
REQ-GUI-1.5	Low	Low	More complex and difficult representation of data
REQ-GUI-1.6	Low	Low	None

REQ-GPS – The Differential Global Positioning System (DGPS)

GRS-GPS

REQ-GPS-1 Ground stations GPS position		
Requirement	**Pri**	**Description**
REQ-GPS-1.1	1	The ground stations GPS position (Coordinates) shall be available to the ground station software.
REQ-GPS-1.2	1	The position data available shall be at least the NMEA 0183$GPRMA/B data *Longitude, latitude, height,*

REQ-GPS-2. GPS failsafe		
Requirement	**Pri**	**Description**
REQ-GPS-2.1	1	The ground station shall handle a GPS disconnect without freezing, crashing or otherwise inhibit the continued operation of the core application.
REQ-GPS-2.2	1	The loss of satellites shall not bring the entire system down
REQ-GPS-2.3	2	Provide the code app with a warning/indication if all satellite connections are lost

Requirement	Risk	Threat	Consequence
REQ-GPS-1.1	Medium	High	The completion of the entire task is dependent of the achievement of this requirement
REQ-GPS-1.2	Medium	High	The minimum information necessary for the successful completion of this task
REQ-GPS-2.1	Medium	Medium	Unstable system
REQ-GPS-2.2	Low	Low	Unstable system
REQ-GPS-2.3	Low	Low	None

REQ-WRL – Wireless link

GRS-WRL

REQ-WRL-1. Communication		
Requirement	**Pri**	**Description**
REQ-WRL-1.1	1	The wireless link needs to communicate with the new ground station core application now written in C++
REQ-WRL-1.2	1	Contain CRC-16 error detection.

REQ-WRL-2. Package flow stability		
Requirement	**Pri**	**Description**
REQ-WRL-2.1	1	The microcontroller shall not send a package while receiving and vice versa.

REQ-WRL-3. Failsafe		
Requirement	**Pri**	**Description**
REQ-WRL-3.1	2	The system shall become aware if connection to aircraft is lost.
REQ-WRL-3.2	3	The signal strength shall be available.
REQ-WRL-3.3	2	An error in the wireless shall not freeze, crash or otherwise inhibit the continued operation of the core application.

Requirement	Risk	Threat	Consequence
REQ-WRL-1.1	Medium	High	Non-Functional system
REQ-WRL-1.2	Medium	Medium	Unstable system
REQ-WRL-2.1	Low	High	Radio link failure
REQ-WRL-3.1	Medium	Low	None
REQ-WRL-3.2	Medium	Low	None
REQ-WRL-3.3	Medium	Medium	Unstable system

Launcher

LSH

REQ-WGN – Wagon system

LSH-WGN

REQ-WGN-1. The Wagon		
Requirement	**Pri**	**Description**
REQ-WGN-1.1	1	The Wagon system shall manage to launch the two different airframes (Stryker and Siren) into stable flight.
REQ-WGN-1.2	1	The Wagon system shall manage to launch the two airframes with or without the propeller running.

REQ-WGN-2. Design requirements		
Requirement	**Pri**	**Description**
REQ-WGN-2.4	2	For obtaining the required takeoff speed (approximately 10 $^{m}/s$) *The length of the wagon shall be limited to max 400mm.*

Requirement	**Risk**	**Threat**	**Consequence**
REQ-WGN-1.1	Medium	High	Inability to comply with this requirement causes the entire task to be failed
REQ-WGN-1.2	Medium	Medium	Same as above
REQ-WGN-2.4	Medium	High	Failure to reach 10 m/s will cause the aircraft launched to crash land upon leaving the launcher

REQ-TRG – Release mechanism

LSH-TRG

REQ-TRG-1. Release mechanism force		
Requirement	**Pri**	**Description**
REQ-TRG-1.1	1	The release mechanism must be strong enough to hold the wagon in lock position, when max force (from the bungee cords)
REQ-TRG-1.2	1	The release mechanism must be strong enough to release the wagon immediately after triggered. *A very important thing regarding safety is to prevent premature launch.*

REQ-TRG-2 Safety		
Requirement	**Pri**	**Description**
REQ-TRG-2.1	1	The release mechanism shall have a safety device for preventing premature launch.

REQ-TRG-3. Design requirements		
Requirement	**Pri**	**Description**
REQ-TRG-3.1	2	The mechanism must be placed in the back of the launcher frame on the main frame witch is 50x50mm (it can build beyond, but not to much because of the launchers handiness)
REQ-TRG-3.2	2	The mechanism cannot be more than 150mm in the direction where the wagon travels. *This is for maintaining the traveling length of the wagon*
REQ-TRG-3.3	3	The mechanism contains a force measurement tool

Requirement	**Risk**	**Threat**	**Consequence**
REQ-TRG-1.1	Low	High	Failure to comply can lead to unintended launch when someone is tightening the bungee cords
REQ-TRG-1.2	Medium	High	Not achieving this requirement can lead to an inability to launch
REQ-TRG-2.1	Low	Medium	Unintentional launches
REQ-TRG-3.1	Low	Low	Unmanageability
REQ-TRG-3.2	Low	Medium	Reduced launch speed and possibility of breaking REQ-WGN-2.4
REQ-TRG-3.3	Low	Low	Inability to know the loaded force

Administrative

SEG

REQ-SEG-1	Movie	
Requirement	**Pri**	**Description**
REQ-SEG-1.1	1	We are to make a movie depicting the work done throughout the project period.

REQ-SEG-2	Poster	
Requirement	**Pri**	**Description**
REQ-MOV-2.1	1	We are to make a poster promotion our product.
REQ-SEG-2.2	1	The poster size must be A2, 420mm × 594mm.

REQ-SEG-3	Manual	
Requirement	**Pri**	**Description**
REQ-SEG-3.1	1	There ought to be a user manual for the ground station.
REQ-SEG-3.2	1	There ought to be a user manual for the launcher.

REQ-SEG-4	Documentation	
Requirement	**Pri**	**Description**
REQ-SEG-4.1	1	We shall to provide sufficient documentation so that the next group that works on this project doesn't have problems familiarizing with the changes we have made.
REQ-SEG-4.1	1	A concept document shall be written.

Requirement	Risk	Threat	Consequence
REQ-SEG-1.1	Low	Low	None
REQ-MOV-2.1	Low	Medium	Missing piece of final documentation
REQ-SEG-2.2	Low	Medium	None
REQ-SEG-3.1	Low	Medium	Difficult use of the Ground Station
REQ-SEG-3.2	Low	Medium	Difficult use of the Launcher
REQ-SEG-4.1	Low	High	Failure to complete the bachelor project
REQ-SEG-4.1	Low	Low	Bad start to the project

Conclusion

From this document we can conclude that we have a good understanding of the risks evolving this project and the consequences if a requirement is not reached.

Type	Level	Occurrence
Risk	Low	22
Risk	Medium	14
Risk	High	0
Threat	Low	16
Threat	Medium	10
Threat	High	10

Low	Low	12
Low	Medium	10
Low	High	4
Medium	Medium	4
Medium	High	6
High	High	0

From these two tables it is clear that the risk is overall low and zero double highs. It will be important to keeping an eye on the 6 requirements in the table below that are medium/high.

Requirement	Risk	Threat
REQ-GPS-1.1	Medium	High
REQ-GPS-1.2	Medium	High
REQ-WRL-1.1	Medium	High
REQ-WGN-1.1	Medium	High
REQ-WGN-2.4	Medium	High
REQ-TRG-1.2	Medium	High

HØGSKOLEN
i Buskerud

SPECIFICATION OF REQUIREMENTS

LocalHawk

Jon Bjørnland
Erik Myklebust
Lars Erik Røise
Lars Hallvard Sannes

March 10th 2010

Summary

This document contains the requirement and constraints for the entire project.
For each of the activities we have written a function part that describes the function of the requirement and the system. Then there is a short explanatory text about how the system is supposed to perform.

Table of Contents

Introduction

This document contains requirements to the entire project.

The number between the requirement number and the requirement is the priority of the requirement.

1	The requirement will be done before all other
2	Not critical.
3	Not important for the function of the unit, but is something that will be fulfilled if there is time.

GRS – Ground Station

GRS

The Ground Station is the system that communicates with the UAV and process and displays the information sent by the UAV. The ground station also allows simple control over the UAV. The system consists of the core

GRS-COR – Core Application

GRS-COR

Function

The Core application is the software that processes the information from the aircraft, what is meant by process is distribute data to the other parts of the ground station system, store data, and handle errors.

Today's application is only a prototype, it's written in Java, and has the necessary functions to fly the aircraft. However the application is too static and does not allow easy replacement of parts of the application, does not have the required real-time capabilities, to processes the information. In order to correct that, the core of the application is to be written in C++.

Performance

Defined by KDS:

REQ-COR-1 The Core application design		
Requirement	**Pri**	**Description**
REQ-COR-1.1	1	The Core shall be written in C++
REQ-COR-1.2	1	The core application shall not be dependent on any other part of the ground station software. *This means that the core shall continue to run even if the GPS and/or the Wireless link is disconnected and any software crashes, other software includes GPS, wireless link drivers and/or GUI.*
REQ-COR-1.3	1	There shall be defined a interface between the peripheral circuit and the core
REQ-COR-1.4	2	There shall be a socket interface between GUI and the core

GRS-GUI – Graphical User interface

GRS-GUI

Function

The GUI shows the information from the aircraft and GPS unit, and enables the user to give commands and overrides to the aircraft.

Performance

Self defined:

REQ-GRS-1 The GUI design		
Requirement	**Pri**	**Description**
REQ-GUI-1.1	1	It contains a map; the map shall enable the user to set waypoint for the aircraft.
REQ-GUI-1.3	2	It contains a console; the console shall give messages and error messages as well as take commands from the user.
REQ-GUI-1.4	2	It contains a graph that can show a statistics of different logged information chosen by the user.
REQ-GUI-1.5	2	It is divided into tabs for each task; *For example map, video, console will have its own tab.*
REQ-GUI-1.6	3	It contains possibilities for user defined settings; *Like the user shall be able to choose whether the map should follow the aircraft or not.*

REQ-GPS – The Differential Global Positioning System (DGPS)

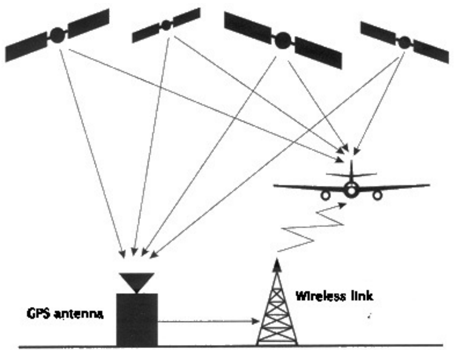
GRS-GPS

Function

Today there is a GPS[1] unit on the aircraft. The accuracy of this unit is not on an acceptable level due to interference and GPS error. To achieve higher accuracy there were forwarded a request to implement a DGPS[2].

This feature requires a GPS module on the ground station; when this is in place there will be possible to improving the accuracy by a DGPS or some other accuracy-improving feature (LAAS[3], RTK[4] or pseudorange time error estimation).

This requirement is in place to make sure that the GPS is working acceptably and is able to handle errors and singularities that might occur due to the inherent structure of the programming or the program itself.



Performance

REQ-GPS-1 Ground stations GPS position		
Requirement	**Pri**	**Description**
REQ-GPS-1.1	1	The ground stations GPS position (Coordinates) shall be available to the ground station software.
REQ-GPS-1.2	1	The position data available shall be at least the NMEA 0183$GPRMA/B data *Longitude, latitude, height,*
REQ-GPS-2. GPS failsafe		
Requirement	**Pri**	**Description**
REQ-GPS-2.1	1	The ground station shall handle a GPS disconnect without freezing, crashing or otherwise inhibit the continued operation of the core application.
REQ-GPS-2.2	1	The loss of satellites shall not bring the entire system down
REQ-GPS-2.3	2	Provide the code app with a warning/indication if all satellite connections are lost

Guiding requirement:

Defined by KDS:

REQ-GPS-3. Make use of the GlobalSat EM406 GPS receiver provided by KDS.

[1] Global Positioning System
[2] Differential Global Positioning System
[3] Local Area Augmentation System
[4] Real Time Kinematic

REQ-WRL – Wireless link

GRS-WRL

Function

When communicating through the wireless link, numerous problems may occur. This includes package loss and problems with transmitting and receiving data at the same time. Problems like these can result in that the ground station does not receive all the desired information like measurements etc. Working with a UAV, it will often lose contact with the ground station making further manual control impossible. When this is about to happen, the operator should get a notification.

Performance

Defined KDS:

REQ-WRL-1. Communication		
Requirement	**Pri**	**Description**
REQ-WRL-1.1	1	The wireless link needs to communicate with the new ground station core application now written in C++
REQ-WRL-1.2	1	Contain CRC-16 error detection.

REQ-WRL-2. Package flow stability		
Requirement	**Pri**	**Description**
REQ-WRL-2.1	1	The microcontroller shall not send a package while receiving and vice versa.
REQ-WRL-3. Failsafe		
Requirement	**Pri**	**Description**
REQ-WRL-3.1	2	The system shall become aware if connection to aircraft is lost.
REQ-WRL-3.2	3	The signal strength shall be available.
REQ-WRL-3.3	2	An error in the wireless shall not freeze, crash or otherwise inhibit the continued operation of the core application.

Launcher

LSH

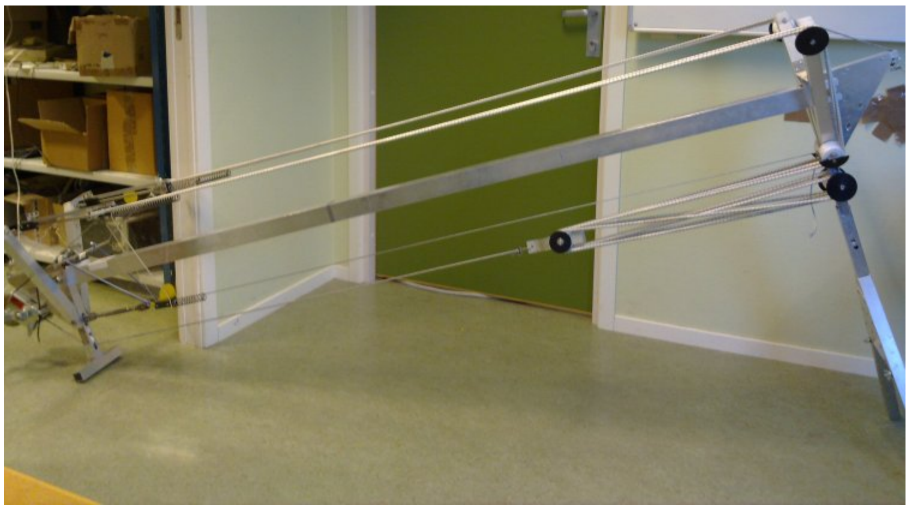
This system consists of the launcher frame and a corresponding wagon (WGN) that holds the aircraft. The wagon is held in place by a trigger (TRG). The function of this rig is to get the aircraft airborne

REQ-WGN – Wagon system

LSH-WGN

Function

To get the airplane in the air we use an already existing Launcher. This is a portable runway with bungee cords. It also has a wagon system for transporting the airframe. The bungee cords accelerate the Wagon system with the aircraft along 3 guidelines up to launch speed.



The existing wagon that we have has a big limitation in launching different types of aircrafts. At the moment it can only launch the Stryker F-27c airframe (The Local Hawk aircraft). It is requested that the wagon can transport and launch two different aircrafts, the F-27c and the Siren. These to airframes are very unlike in structure. The Siren aircraft have the propeller in the front and also a much larger wingspan than the Stryker. This means that the Wagon system needs to be rebuilt, so it can be able to launch both aircrafts. It is also possible that a solution containing two different Wagon will be the alternative.

Performance

Defined by KDS:

REQ-WGN-1. The Wagon		
Requirement	**Pri**	**Description**
REQ-WGN-1.1	1	The Wagon system shall manage to launch the two different airframes (Stryker and Siren) into stable flight.
REQ-WGN-1.2	1	The Wagon system shall manage to launch the two airframes with or without the propeller running.

Self defined:

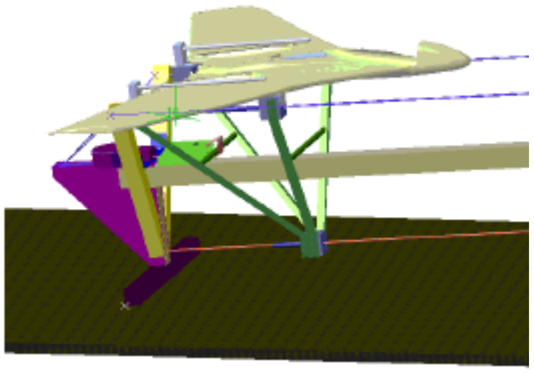
REQ-WGN-2. Design requirements		
Requirement	**Pri**	**Description**
REQ-WGN-2.4	2	For obtaining the required takeoff speed (approximately 10 ^{m}/s) *The length of the wagon shall be limited to max 400mm.*

REQ-TRG – Release mechanism

LSH-TRG

Function

The release mechanism will more than likely be a solution that works a lot like the solution from the summer of 2009. The wagon will be pushed back into the release mechanism. This will hold the wagon in place while the bungee cords are winched into tension. When the desired energy is built up; the mechanism is triggered and the wagon sets the aircraft in motion. Today the function of the release mechanism is limited to only release manually (with a thread). It is developed to release semi automatic using a servo, but this is not strong enough.



This figure shows a part of the launcher at an earlier stage of development done by last year's student project.

Therefore we need to create a new solution that works properly. It has to be able to hold and release the launcher.

Performance

Defined by KDS:

REQ-TRG-1. Release mechanism force		
Requirement	**Pri**	**Description**
REQ-TRG-1.1	1	The release mechanism must be strong enough to hold the wagon in lock position, when max force (from the bungee cords)
REQ-TRG-1.2	1	The release mechanism must be strong enough to release the wagon immediately after triggered. *A very important thing regarding safety is to prevent premature launch.*

Self defined:

REQ-TRG-2 Safety		
Requirement	**Pri**	**Description**
REQ-TRG-2.1	1	The release mechanism shall have a safety device for preventing premature launch.

REQ-TRG-3. Design requirements		
Requirement	**Pri**	**Description**
REQ-TRG-3.1	2	The mechanism must be placed in the back of the launcher frame on the main frame witch is 50x50mm (it can build beyond, but not to much because of the launchers handiness)
REQ-TRG-3.2	2	The mechanism cannot be more than 150mm in the direction where the wagon travels. *This is for maintaining the traveling length of the wagon*
REQ-TRG-3.3	3	The mechanism contains a force measurement tool

Administrative

SEG

In addition to the aforementioned requirements there is also some requirements regarding documentation of the project and as regards to the administrative work.

REQ-SEG-1 Movie		
Requirement	**Pri**	**Description**
REQ-SEG-1.1	1	We are to make a movie depicting the work done throughout the project period.

REQ-SEG-2 Poster		
Requirement	**Pri**	**Description**
REQ-MOV-2.1	1	We are to make a poster promotion our product.
REQ-SEG-2.2	1	The poster size must be A2, 420mm × 594mm.

REQ-SEG-3 Manual		
Requirement	**Pri**	**Description**
REQ-SEG-3.1	1	There ought to be a user manual for the ground station.
REQ-SEG-3.2	1	There ought to be a user manual for the launcher.

REQ-SEG-4 Documentation		
Requirement	**Pri**	**Description**
REQ-SEG-4.1	1	We shall to provide sufficient documentation so that the next group that works on this project doesn't have problems familiarizing with the changes we have made.
REQ-SEG-4.1	1	A concept document shall be written.

HØGSKOLEN i Buskerud

TEST SPECIFICATION

LocalHawk

Jon Bjørnland

Erik Myklebust

Lars Erik Røise

Lars Hallvard Sannes

March 10th 2010

Summary

This document contains the test specification, on which we base the verification of our completed taskt. The test specification describes each of the test procedure that will test the requirements found in specification of requirement.

For each of the activities we have written an introduction that describes the requirement, which is necessary to test. Then there is a short explanatory text about why we test the specific requirements, followed by a deduction of how to execute the test.

Table of Contents

TST-COR – Core Application

GRS-COR

Requirements

REQ-COR-1 The Core application design

REQ-COR-1.1	1	The Core shall be written in C++
REQ-COR-1.2	1	The core application shall not be dependent on any other part of the ground station software. *This means that the core shall continue to run even if the GPS and/or the Wireless link is disconnected and any software crashes, other software includes GPS, wireless link drivers and/or GUI.*
REQ-COR-1.3	1	There shall be defined a interface between the peripheral circuit and the core
REQ-COR-1.4	2	There shall be a socket interface between GUI and the core

Test for REQ-COR-1.2

TST-COR-1

The core application shall not be dependent on any other part of the ground station software.

Execution TST-COR-1.1

Disconnect the GUI from its socket end, if the core continues to run the test is considered a success.

Execution TST-COR-1.2

Run the following tests for the GPS: TST-GPS-2 and Wireless: TST-WRL-3, if these tests is passed this test is also considered a success.

Execution TST-COR-1.3

Send incorrect/destructive information in to the core, if the core handles the information and continues to run the test is considered a success.

Test for REQ-COR-1.3 and REQ-COR-1.4

TST-COR-2

There shall be a socket interface between GUI and the core

Execution TST-COR-2.1

Run simulated information trough the interface and if the information is correctly formatted after passing through the interface the test is considered a success.

TST-GUI – Graphical User Interface

GRS-GUI

Requirements

REQ-GRS-1 The GUI design

REQ-GUI-1.1	1	It contains a map; the map shall enable the user to set waypoint for the aircraft.
REQ-GUI-1.3	2	It contains a console; the console shall give messages and error messages as well as take commands from the user.
REQ-GUI-1.4	2	It contains a graph that can show a statistics of different logged information chosen by the user.
REQ-GUI-1.5	2	It is divided into tabs for each task. *For example map, video, console will have its own tab.*
REQ-GUI-1.6	3	It contains possibilities for user defined settings; *Like the user shall be able to choose whether the map should follow the aircraft or not.*

Test for REQ-GUI-1.3

TST-GUI-1

It contains a console; the console shall give messages and error messages as well as take commands from the user.

Execution TST-GUI-1.1

Run simulated messages and error trough the system, disconnect the GPS and the wireless. If the correct messages are displayed, the test is considered a success.

Execution TST-GUI-1.2

Type inn commands and read the output in the other end of the system. If the commands match, the test is considered a success.

Test for REQ-GUI-1.4

TST-GUI-2

It contains a graph that can show a statistics of different logged information chosen by the user.

Execution TST-GUI-2.1

Send simulated information, and if the information displayed is correct, the test is considered a success.

TST-GPS – Differential Global Positioning System

GRS-GPS

Requirements

REQ-GPS-1 Ground stations GPS position

REQ-GPS-1.1	1	The ground stations GPS position (Coordinates) shall be available to the ground station software.
REQ-GPS-1.2	1	The position data available shall be at least the NMEA 0183 $GPRMA/B data *Longitude, latitude, height,*

REQ-GPS-2 GPS failsafe

REQ-GPS-2.1	1	The ground station shall handle a GPS disconnect without freezing, crashing or otherwise inhibit the continued operation of the core application.
REQ-GPS-2.2	1	The loss of satellites shall not bring the entire system down
REQ-GPS-2.3	2	Provide the code app with a warning/indication if all satellite connections are lost

Test for REQ-GPS-1

TST-GPS-1

The test for this requirement is pretty strait forward, the requirement is mainly the to show and how the is shown.

Execution TST-GPS-1.1

Connect to the GPS the ground station and see if the ground station displays the GPS position. The test is considered a success if the position is showed correct.

Test for REQ-GPS-2

TST-GPS-2

The ground station should handle a GPS disconnect and satellite loss without freezing, crashing or otherwise inhibit the continued operation of the core application.

Execution TST-GPS-2.1

Connect up the GPS and wait for satellite lock, then disconnect the cable between the GPS antenna module and the circuit board. If the core application still operates, the test is considered a success.

Execution TST-GPS-2.2

Connect up everything as in 2.1, then go inside so that the GPS looses its satellite, simulating a satellite loss. If the core application still operates, the test is considered a success. If the ground station also shows an indication of satellite loss the REQ-GPS-2.3 is also a success.

TST-WRL – Wireless link

GRS-WRL

Requirements

The wireless link has these three requirements that have to be completed.

REQ-WRL-1.1	1	The wireless link needs to communicate with the new ground station core application now written in C++.
REQ-WRL-1.2	1	Contain CRC-16 error detection.
REQ-WRL-2.1	1	The microcontroller shall not send a package while receiving and vice versa.
REQ-WRL-3.1	1	An error in the wireless shall not freeze, crash or otherwise inhibit the continued operation of the core application.
REQ-WRL-3.2	2	The system shall become aware if connection to aircraft is lost.
REQ-WRL-3.3	3	The signal strength shall be made available.

Test for REQ-WRL-1

TST-WRL-1

This requirement is mainly rewriting the previous code to make it work with the new design.

Execution TST-WRL-1.1

Load the code onto the microcontroller and connect the antenna to the board together with the primary microcontroller. Try to send information through the wireless link. If the input data is equal to the received data, then the test is considered a success.

Test for REQ-WRL-2

TST-WRL-2

This requirement is set to make sure the radio module does not simultaneously transmit and receive at the same on the same channel, since this is not supported.

Execution TST-WRL-2.1

Try and transmit data from two radio modules simultaneously[1]. These two shall have established a connection in advance. If this is prevented so that only one transmits at a time, then the test is considered a success.

Test for REQ-WRL-3

TST-WRL-3

[1] One should be the ground station module, while the other one is LocalHawk, or similar.

This requirement is set to make the system more failsafe. When operating the LocalHawk; it is often necessary to know when one is allowed to communicate with the aircraft to make adjustments etc. This should be made available so that the ground station can give notification to operator that LocalHawk is out of range.

Execution TST-WRL-3.1

Hopefully, the GUI is able to show information about connection status. If not, a test to test this function will be too time consuming and will not be required.

1. Establish a connection with two radio modules (ground station and the aircraft).
2. Find a way to weaken the signal (either by distance, obstacle, Faraday cage, remove the antenna, etc.).

If the GUI shows a notification of lost contact, then the test is considered a success.

TST-WGN – Wagon system

LSH-WGN

Requirements

The wagon objective has two functional requirements.

REQ-WGN-1. The wagon

REQ-WGN-1.1	1	The Wagon system must manage to launch the two different airframes (Stryker and Siren) into stable flight.
REQ-WGN-1.2	1	The Wagon system must manage to launch the two airframes with or without the propeller running.

Test for REG-WGN-1

TST-WGN-1

One of the requirements given from KDS was create or modify the wagon so it can be used to launch both of the aircrafts. The solution we have decided for launching the airframes consist of one main wagon frame and two assembly parts we can mount on the main frame. Each of the airframes got its one . So basically the test needs to show that we are able to launch both of the airframes several times without any interruptions. It is important that the airframes gets enough initial velocity when they abandon the wagon. They need to be able to glide without propeller force. So for testing this we propose the following test:

Execute TST-WGN-1.1

Set up the launcher outdoors and perform two launch sequences with one of the top part mounted on the launcher. Capture the launch sequence using a high speed camera. With the data collected, calculate the wagons velocity at the end of the launcher frame. If the wagons velocity is equal to or more than 10 $^{m}/s$ the test is considered a success and we can continue the testing.

Execute TST-WGN-1.2

Now one of the assembly parts is ready to launch with the airframe that fit the wagon. So place the airframe on the wagon and prepare the launch sequence. First have one launch without propeller force, then one with the propeller running.

If the airframes get into stable flight both of the times the test is considered a success for one of the assembly parts. Then do the entire test again. This time with the other part mounted to the wagon with the other airframe.

TST-TRG – Release Mechanism

LSH-TRG

Requirements

This test will look at the following two functional requirements:

Defined by KDS:

REQ-TRG-1. Release mechanism force

REQ-TRG-1.1	1	The release mechanism must be strong enough to hold the wagon in lock position, when max force (from the bungee cords)
REQ-TRG-1.2	1	The release mechanism must be strong enough to release the wagon immediately after triggered.

Self defined:

REQ-TRG-2 Safety

REQ-TRG-2.1	1	The release mechanism should have a safety device for preventing premature launch.
REQ-TRG-2.2	2	The release mechanism must have implemented a electronic device for triggering the wagon. This is to prevent injure on operator

Test for REQ-TRG-1

TST-TRG-1

When using the launcher, different extensions of the bungee cords might be applied. It is therefore necessary that the trigger will not be damaged by too much cord extension.

Execution TST-TRG-1.1

The wagon will be pulled back and locked into the release mechanism. An operator uses the winch to extend the bungee cords to the maximum[2]. This will create max force from the bungee cords on the trigger. When area is safe, an operator triggers the release.

After the wagon has launched, the release mechanism and the launcher needs to be inspected for damages. The test will then be done one more time, with normal load. If the wagon is released without damages, the test is considered successful.

[2] Maximum is in this case when the wire is winded back as far as possible

Test for REQ-TRG-2

TST-TRG-2

The release mechanism should have a safety device for preventing premature launch.

Execution TST-TRG-2.1

Do the same as in REQ-TRG-1.1/1.2 but lock the wagon with a release mechanism. Then when we have max force from the bungee cords, we release the trigger without releasing the safety device. If the safety device is strong enough to hold the wagon, we consider the test a success.