Sensur av hovedoppgaver

Høgskolen i Buskerud og Vestfold
Fakultet for teknologi og maritime fag

Prosjektnummer: **2014-04**
For studieåret: **2013/2014**
Emnekode: **SFHO3201**

Prosjektnavn: Automated CROWS Testing
Automatisert CROWS Testing

Utført i samarbeid med: Kongsberg Protech Systems.

Ekstern veileder: Dag Christian Nygaard.

Sammendrag: ACT er et system som gjennomfører automatiske funksjonstester på CROWS ved hjelp av en industrirobot.

Stikkord:
- Robot
- Verifisering
- Automatisering

Tilgjengelig: JA

Prosjektdeltagere og karakterer:

Navn	Karakter
August Kind Svendsen	
Eirik Lien Roa	
Henrik Berge Sørum	
Ståle Rudin	

Dato: 12. Juni 2014

_____	_____	_____
Daniel Larsson	**Karoline Moholth**	**Hallvard Murberg**
Intern veileder	Intern sensor	Ekstern sensor

ACT AUTOMATED CROWS TESTING

BACHELOROPPGAVE - GR 4

AUGUST KIND SVENDSEN
EIRIK LIEN ROA
HENRIK BERGE SØRUM
STÅLE RUDIN

KONGSBERG

HBV
HØGSKOLEN I BUSKERUD OG VESTFOLD · PROFESJONSHØGSKOLEN ·

Dokumentnavn:	Nummer:
Prosjektplan	1
Kravspesifikasjon	2
Software designdokument	3
Komponentdiagram	4
Teknologidokument - Robot	5
Teknologidokument - XML	6
Teknologidokument - Software	7
Teknologidokument - Verifikasjon	8
Testspesifikasjon	9
Testrapport	10
Sluttrapport	11
Fremtidsdokument	12
Etteranalyse	13
Installasjonsguide	14
Iterasjonsdokument	15

ACT AUTOMATED CROWS TESTING

PROSJEKTPLAN

AUGUST KIND SVENDSEN
EIRIK LIEN ROA
HENRIK BERGE SØRUM
STÅLE RUDIN

KONGSBERG

HØGSKOLEN I BUSKERUD OG VESTFOLD · PROFESJONSHØGSKOLEN · HBV

Prosjektplan			
PROSJEKT	ACT – Automated CROWS Testing		
OPPDRAGSGIVER	Kongsberg Protech Systems		
UTFØRT VED	HBV – Høgskolen i Buskerud og Vestfold		
GRUPPE	August Kind Svendsen, Eirik Lien Roa, Henrik Berge Sørum og Ståle Rudin		
VERSJON	3.0		
ANTALL SIDER	28		
DOKUMENTHISTORIE	VERSJON	UTGITT	BESKRIVELSE
	1.0	17.02.14	Første utgivelse
	2.0	24.03.14	Andre utgivelse
	3.0	26.05.14	Tredje utgivelse

INNHOLDSFORTEGNELSE

LISTE OVER TABELLER

LISTE OVER FIGURER

1 Om Dokumentet

1.1 Dokumenthistorie

VERSJON	DATO	ENDRING	SIGNATUR
0.1	27.1.2014	Dokument opprettet	ELR, SR
1.0	17.02.14	Første utgivelse	AKS, ELR, HBS, SR
1.1	03.03.14	• Endret prosjektmodell • Endret prosjektmodellbeskrivelse • Endret aktivitetsliste • Ryddet opp i dokumentet	AKS, HBS, SR
1.2	07.03.14	• Endret fremdriftsplan • Endret prosjektfaser • Lagt til iterasjoner	HBS, SR
2.0	21.03.14	• Lagt inn nye gantt-diagrammer • Oppdatert prosjektmodell • Rettet på milepæler • Rettskriving og småfiks	HBS
2.1	22.04.14	• Endret sluttdato på konstruksjons iterasjon #1 • Endret sluttdato på konstruksjons iterasjon #2 • Endret sluttdato på konstruksjons iterasjon #3 • Fjernet konkluderende iterasjon #1 • Endret konkluderende iterasjon #2 så den inneholder begge iterasjonene • Endret datoer på milepæler	HBS
3.0	26.05.14	• Satt inn ny fremdriftsplan • Satt inn nye gantt-diagrammer • Endret aktiviteter og ID	SR

Tabell 1: Dokumenthistorie

1.2 Definisjoner og forkortelser

Liste som beskriver ord og uttrykk som er brukt for å beskrive oppgaven

UTTRYKK	FORKLARING
HBV	Høgskolen i Buskerud og Vestfold
KPS	Kongsberg Protech Systems (oppdragsgiver)
ACT	Automated CROWS Testing
Industrirobot	Universal Robots 5
CROWS	Common Remotely Operated Weapon Station
RWS	Remote Weapon Station
Konfigurasjon	Forskjellige CROWS oppsett
DCP	Display Control Panel
MPU	Main Processing Unit
UP	Unified Process

Tabell 2: Definisjoner og forkortelser

2 Innledning

Prosjektplanen bruker vi som et styringsdokument som legger frem hvordan vi ønsker å utføre det avsluttende hovedprosjektet. Prosjektet utføres ved Høgskolen i Buskerud og Vestfold, fakultet for teknologi, studieåret 2013/14.

Arbeidsgiver er Kongsberg Protech Systems, en bedrift som jobber med å produsere og levere fjernstyrte våpenstasjoner for den amerikanske hæren. Oppgaven de har delt ut omhandler å utvikle et system som kan utføre automatiske funksjonstester på våpenstasjonene. Denne funksjonstesten innebærer fysiske oppgaver som å trykke på knapper og dra i spaker som styrer våpenstasjonen, og deretter se på utfallet av disse aksjonene. Så må en sjekke om utfallet stemmer overens med det ønskede resultatet.

Tidligere har denne veldig repetitive og tidkrevende oppgaven blitt gjort manuelt av ansatte. KPS ønsker derfor å ta i bruk en industrirobot fra Universal Robots som kan utføre de fysiske arbeidsoppgavene på løpende bånd, samt lage et system for å verifisere resultatene på en troverdig måte.

I dette dokumentet vil det presenteres temaer som aktiviteter og ansvarsfordelinger, hvilke mål og rammer som har blitt satt, risikoanalyse og kvalitetssikring, organisering med aktivitetsoversikt, og flere mindre temaer som er greit å få dekket for å unngå usikkerhet senere i prosjektfasen.

3 Mål og rammer for prosjektet

3.1 Bakgrunn

I dag er testing av CROWS utført manuelt, som er en tidskrevende prosess på nokså enkle og praktiske funksjonstester. Ved hver utgivelse av software til våpenstasjonen må den gjennom tre funksjonstester som hver tar fem arbeidsdager. Dette koster arbeidsgiver mye ressurser og vil derfor finne ut om dette kan automatiseres. Arbeidsgiver har kjøpt inn en industrirobot hvor tanken er at de enkle testene skal automatiseres. Hvorvidt dette lar seg realiseres og hvor mye ressurser som må settes av er vanskelig å beregne, noe som gjør at det egner seg godt som en bacheloroppgave.

3.2 Problemstilling

Hvordan kan vi produsere et godt rammeverk for et program som skal automatisere funksjonstester på CROWS konfigurasjoner med hjelp av en industrirobot, og kunne loggføre hendelser og verifisere sekvenser?

3.3 Målsetting

Hovedmålet med prosjektet er å planlegge og gjennomføre et utviklingsprosjekt fra begynnelse til slutt. Resultatet av prosjektet skal holde seg innenfor kravene som er satt av gruppa og arbeidsgiver. I tillegg skal dette prosjektet evalueres underveis som vil utlede til en konklusjon som skal gjenspeile gruppedeltakernes evne innen tekniske fag og prosjektkoordinering.

3.3.1 Resultatmål

Målet for prosjektet vårt er å lage et brukervennlig system som skal kunne utføre noen enkle funksjonstester. Disse funksjonstestene skal automatiseres via en industrirobot.

Etter at prosjektet vårt er ferdig skal vi ha et produkt som skal være lett for bedriften å jobbe videre på. Målet vårt er da og sette opp en god og oversiktlig plattform som vil utfylle dette kravet. Et av kriteriene fra bedriften var at de skulle kunne legge inn nye tester på en enkel måte.

I tillegg er målet å lage et enkelt, men brukervennlig program som skal styre hele prosessen. Dette programmet skal også håndtere resultater og skrive ut en rapport med henhold til de resultatene den kommer frem til.

3.3.2 Gevinstmål

I tiden etter prosjektet er målet at mye av CROWS funksjonstesting skal foregå automatisk. En industrirobot står for de fysiske arbeidsoppgavene på systemet i stedet for at en person sitter og gjør det manuelt. Dette vil spare bedriften ressurser i form av tid og penger.

3.3.3 Mål for gruppa

- Utvikle et godt produkt som gruppa og arbeidsgiver er fornøyd med.
- Oppnå en sluttkarakter som gruppemedlemmene er fornøyd med.
- Videreutvikle kunnskap om prosjektarbeid.
- Forbedre kompetanse innen software og systemutvikling.
- Forbedre kompetanse innen database og nettverk.
- Utvikle større kunnskap om dokumentering og analysering av resultater.

3.5 Prosjektrammer

Ved endt godkjent prosjekt blir det tildelt 20 studiepoeng per deltaker. Ett studiepoeng tilsvarer cirka 25-30 effektive arbeidstimer, det vil si at det forventes totalt litt over 2000 arbeidstimer fra prosjektgruppen. Forskjellen fra tidligere år er tidsrammen for dette prosjektet kun under vårsemesteret 2014.

Prosjektet består i hovedsak av tre presentasjoner med innleveringer. Den første presentasjonen skal gjøres tidlig i prosjektet, og er ment som en fremlegging for hvordan vi planlegger å jobbe videre. Dette innebærer en gjennomgang av prosjektplan, kravspesifikasjon og testspesifikasjon.

Etter at omtrent 40 % av prosjektet er utført skal den andre presentasjonen gjøres. Her blir vi litt mer tekniske og presenterer design og hvordan vi har tenkt å teste. I tillegg anføres det hvordan vi ligger an i forhold til prosjektplanen, og hva som skal gjøres videre.

I den siste og mest avgjørende presentasjonen skal alt som har blitt gjort til da presenteres og legges frem for veiledere, sensorer og eventuelle andre som er interessert i prosjektet.

Før hver presentasjon skal tilhørende prosjektdokumentasjon leveres inn til KPS og HBV minst to arbeidsdager før presentasjonsdato.

Oppdragsgiver er pålagt til å utnevne en ekstern veileder for prosjektet. Arbeidsoppgaver for denne veilederen innebærer å presisere oppgaven ved uklarhet, samt stå til rådighet når grupper trenger assistanse for materielle goder og andre ressurser.

I tillegg til veileder skal oppdragsgiver tildele prosjektgruppen en ekstern sensor. Denne sensoren er ment for å gjøre en evaluering fra arbeidsgiverens standpunkt, og vil ha betydning for den endelige karakteren.

Vi tildeles ikke lønn for arbeidet, men arbeidsgiver dekker prosjektgruppes utgifter som er direkte relatert til oppgaven. Disse skal på forhånd godkjennes av arbeidsgiver.

3.6 Prosjektmodell

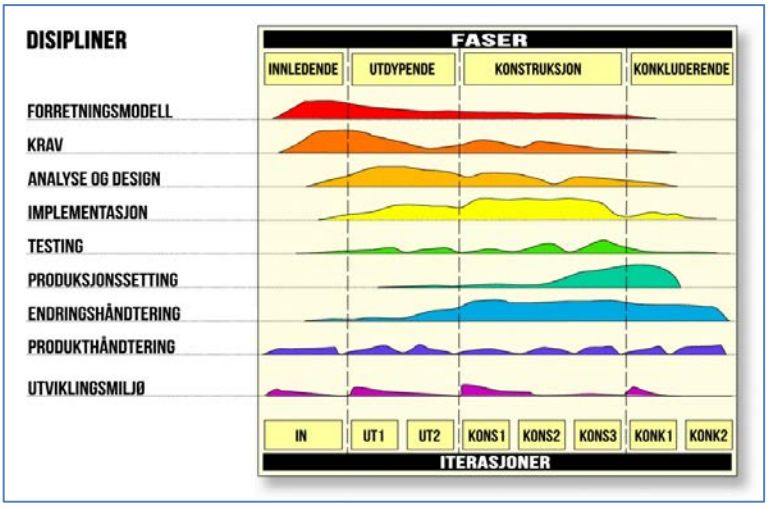
En god prosjektmodell er viktig for større prosjekter. Prosjektmodellen gir et godt bilde av hvordan arbeidet skal organiseres og hvilken rekkefølge aktivitetene skal gjennomføres i. Modellen deler prosjektet opp i faser som gir en oversiktlig struktur som gjør at prosjektgruppa får en felles forståelse om hvordan arbeidet skal utarte seg.

Vi har valgt å gå for den iterative prosjektmodellen Unified Process, eller bare UP. Modellen deles hovedsakelig inn i to dimensjoner, der den horisontale aksen er «faser», mens den vertikale viser «disipliner». Faser beskriver hvordan utviklingen av produktet er i forhold til tidsplan, mens disipliner er grener som beskriver hvilken aktivitet det er fokus på.

Iterasjoner er bestemte tidspunkt der vi går over hver og en av disiplinene sammen i gruppa, og analyserer hvordan vi ligger an. Dette gir et helhetlig bilde av utvikling og kan være med å oppdage ubalanser som vi da rekker å rette opp før fasen er over.

Det som gjør denne modellen egnet til våres prosjekt er at en UP modell legger veldig fokus på de mer krevende aktivitetene. Det vil si at når vi starter en iterasjon så skal de aktivitetene som er krevende eller vanskelige fokuseres først, dette gjøres for å redusere risikoen for at aktiviteter ikke skal bli ferdige tidsnok og at man skal ha tid til å fikse uforutsette utfordringer. En annen egenskap som gjør at denne modellen passer godt for oss er at disiplinene strekker seg over alle fasene, men hvor hvilken grad av fokus varierer. For eksempel kan vi se at vi helt fra den innledende fasen har vi noe fokus på «Analyse og Design», men ikke på langt nær så mye som når vi har kommet litt inn i den utdypende fasen. Dette gjør modellen proaktiv mot feil som kan oppstå, og lar oss tilpasse andre disipliner dersom en forandring i krav eller implementasjon dukker opp.

Det er vanlig å benytte seg av artefakter i UP og et eksempel på dette kan være et UML diagram. Dette er noe vi har lært å lage på HBV og ser på det som en fordel at vi kan bruke disse kunnskapene som en del av prosjektmodellen.



Figur 1: Prosjektmodell (UP)

4 Avgrensninger og forutsetninger

4.1 Forutsetninger

Det forutsettes at studentene har visse forkunnskaper for kunne fullføre prosjektet. Kunnskapene skal studentene ha opparbeidet i løpet av sin skolegang på HBV. Det innebærer kunnskap innen programmering, datavitenskap med spesiell vekt på nettverk, evne til å kunne jobbe i gruppe, og planlegge et større prosjekt med tilhørende dokumentasjon.

Fra arbeidsgivers side forutsettes det at tildelt oppgave er mulig å realisere, i hvert fall til en viss grad. Om oppgaven krever spesielt utstyr som ikke er mulig å anskaffe som student, forutsettes det at arbeidsgiver hjelper til her.

4.2 Avgrensninger

Målet vårt er å skape en god plattform som arbeidsgiver enkelt kan utvikle videre om de ønsker flere funksjoner i systemet. Vi avgrenser derfor oppgaven, i tillegg til å være et godt utgangspunkt for videre utvikling, til å få systemet til å kunne utføre enkle, praktiske funksjonstester.

5 Prosjektgruppa og oppdragsgiver

5.1 Kontaktinfo arbeidsgiver

INFORMASJON
Dag Christian Nygaard **(Dag.christian.nygaard@kongsberg.com, Dag_christian_nygaard@hotmail.com)** Kongsberg Protech Systems Prosjektrolle: Ekstern veileder
Hallvard Murberg **(Hallvard.murberg@kongsberg.com)** Kongsberg Protech Systems Prosjektrolle: Ekstern sensor

Tabell 3: Kontaktinfo arbeidsgiver

5.2 Om oppdragsgiver

Kongsberg Protech Systems (KPS) er en bedrift som er en del av Kongsberg Gruppen. KPS leverer teknologi og produkter innenfor forsvarsmateriell. Bedriften er verdensledende leverandør av Remote Weapon Stations, også kalt våpenstasjoner.

En våpenstasjon et apparat som er montert på utsiden av et kjøretøy. Dette apparatet består hovedsakelig av kameraer og skytevåpen slik at personen som styrer dette kan forsvare seg uten å utsette seg for fare ved å forlate innsiden av kjøretøyet.

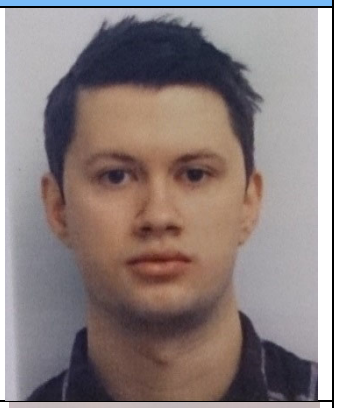
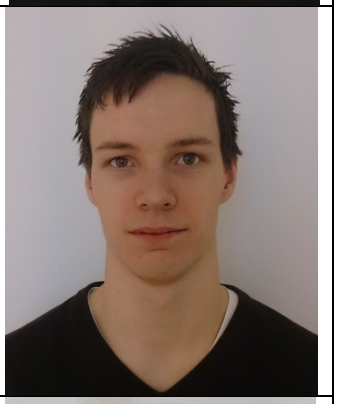
I 1987 ble Kongsberg Våpenfabrikk restrukturert og all sivil virksomhet ble solgt. Da ble selskapet Norsk Forsvarsteknologi etablert som igjen dannet grunnlaget for Kongsberg Defence & Aerospace.

Kongsberg Defence & Aerospace er delt opp i to deler, Kongsberg Defence Systems (KDS) og Kongsberg Protech Systems (KPS). KDS utvikler og produserer forsvars og romfarts relaterte systemer, mens KPS utvikler og produserer våpenstyringssystemer.

Om leser ønsker mer informasjon om KPS eller KDS finnes det på deres hjemmesider:

KPS - http://www.kongsberg.com/en/kps/
KDS - http://www.kongsberg.com/en/kds/

5.3 Prosjektmedlemmer

INFORMASJON	ANSVARSOMRÅDE	BILDE
August Kind Svendsen **(Augustkinds@gmail.com)** Dataingeniør, Virtuelle systemer	- Dokumentasjon - Krav - Software design - Backup	
Eirik Lien Roa **(Erooa9@gmail.com)** Dataingeniør, Virtuelle systemer	- Prosjektleder - Økonomi - Verifikasjon	
Henrik Berge Sørum **(Henrikbsorum@gmail.com)** Dataingeniør, Virtuelle systemer	- Robotskripting - Sekretær - Test	
Ståle Rudin **(Staale_rud@hotmail.com)** Dataingeniør, Virtuelle systemer	- Integrasjon - Kommunikasjon - System kontroller	

Vi har delt de forskjellige arbeidsområdene i prosjektet inn i ansvarsområder. Dette gjør det enklere å kunne fordele oppgaver og at en person har hovedansvar over et bestemt område. Så lenge det er mulig så vil gruppa samarbeide på de forskjellige områdene. Alle gruppedeltakerne må ha en viss forståelse av alle områdene slik at eventuell sykdom/fravær ikke gjør at deler av prosjektet stopper opp.

5.4 Ansvarsområder

Ansvar	Beskrivelse	Hvem
Prosjektleder	Har det overordnede ansvaret for at prosjektet blir gjennomført.	ELR
Dokumentasjon	Har ansvaret for at all dokumentasjone blir laget, at all dokumentasjon stemmer i forhold til maler og at versjonene blir fulgt og oppdatert.	AKS
Økonomi	Har ansvaret for å identifisere eventuelle kostnader og føre regnskap.	ELR
Software design	Har ansvaret for software design til oppgaven.	AKS
Sekretær	Har ansvaret for å skrive møtereferater fra alle interne og eksterne møter og Oppfølgingsdokumenter.	HBS
Test	Har ansvaret for all testing og at testspesifikasjonene blir oppfulgt.	HBS
Integrasjon	Har ansvaret for integrasjon av systemet.	SR
Verifikasjon	Har ansvaret for verifikasjonen til systemet(Bildegjenkjenning og data fra RWS).	ELR
Kommunikasjon	Har ansvaret for kommunikasjon mellom komponenter.	SR
Robotskripting	Har ansvaret for skripting til robot.	HBS
Krav	Skal utarbeide og oppdatere prosjektkrav og kravspesifikasjon.	AKS
Backup	Ta backup hver andre dag av gruppens dokumenter og filer.	AKS
Systemkontroller	Har ansvaret for at systemet utfører funksjoner i riktig rekkefølge (kontroller).	SR

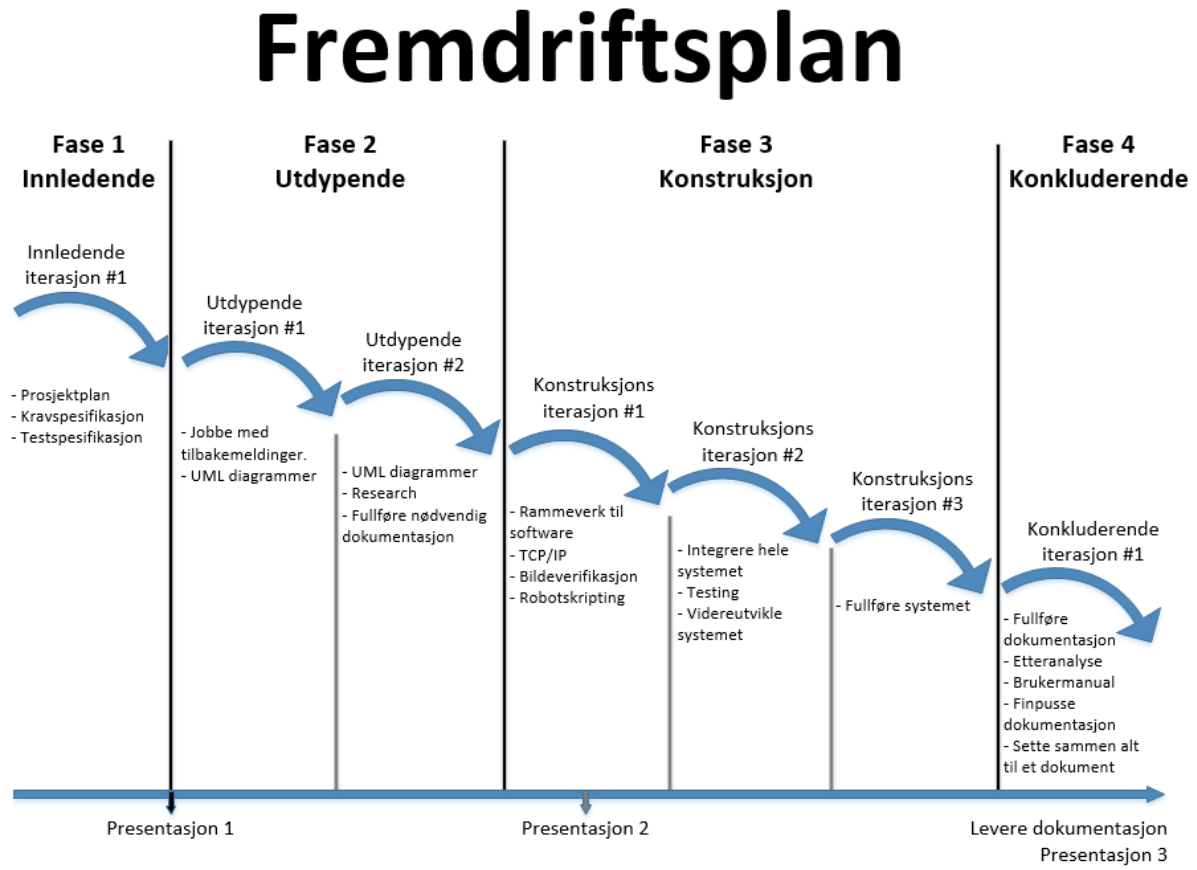
Tabell 4: Ansvarsområder

6 Gjennomføring

6.1 Prosjektplan

Overordnet plan over oppgaveløsningen.

6.1.1 Prosjektfremdrift



Figur 2: Fremdriftsplan

6.1.2 Prosjektfaser

6.1.2.1 Innledende fase

Som navnet tilsier er dette fasen vi setter oss inn i prosjektet og får satt de grunnleggende tingene på plass. Arbeidsplass med nødvendig utstyr, spesifisering av oppgave, og valg av veiledere og sensorer vil ta opp tid. Prosjektgruppa kommuniserer med arbeidsgiver og blir enige om hva oppgaven innebærer. Gruppa gjør analyse på egenhånd for å kunne planlegge arbeidsfasen nøyere. Hvordan gruppa planlegger å arbeide senere og hvordan prosjektet organiseres blir dokumentert i prosjektplanen. I tillegg til prosjektplan så blir testspesifikasjon og kravspesifikasjon skrevet. Disse dokumentene har prioritet i denne fasen. Dokumentene må være ferdige til første presentasjon og derfor viktig at disse ikke blir nedprioritert av andre aktiviteter som research, og lignende.

6.1.2.2 Utdypende fase

Denne fasen starter når presentasjon én er gjennomført. Planen er først og fremst å lage alle dokumenter og diagrammer som er nødvendig for konstruksjonsfasen. I denne fasen skal det lages UML diagrammer og designdokument. Det skal også gjøres en del research om de forskjellige delene i systemet som vi skal starte og programmere i konstruksjonsfasen. I denne fasen er det designdokumentet som er hovedprioritet. Designdokumentet er grunnlaget for konstruksjonsfasen og det er viktig at blir gitt ut en første versjon av dokumentet før konstruksjonsfasen starter.

6.1.2.3 Konstruksjons fase

Nå som alt er planlagt er det bare for gruppa å sette i gang med arbeidet som vi har sett frem til å ta fatt på. Det er nå vi legger rammeverket og sjekker at den planlagte infrastrukturen fungerer, det innebærer oppsett av TCP/IP, få kontroll på industriroboten, oppsett av bildeuthenting og gjenkjenning.

Det må lages er parser som kan tolke og gjøre om informasjon som blir sendt over TCP socket. Siden denne informasjonen ikke er "lesbar" så trenger man en parser i hver ende slik at den kan "oversette" informasjonen til noe som kan brukes.

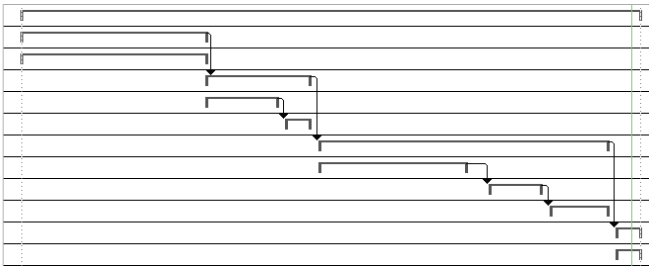
Når alle individuelle deler av systemet har et fungerende utkast, skal alt integreres sammen til et system. En stor del blir da å teste systemet, finne feil og rette disse. Vi skal også finpusse systemet mot slutten av denne fasen. Finpussing blir gjort ettersom hvor god tid det er igjen av fasen. Så lenge vi har et fungerende system i slutten av fasen så er det viktigst, men all ekstrajobb som å forbedre er bra hvis det er tid.
Hovedprioritet i denne fasen vil være de delene av systemet vi ser på som krevende eller vanskelig. Kommunikasjon med parsere og bildegjenkjenning er noe vi tror blir tidskrevende og vanskelig så dette får høy prioritet i starten av denne fasen. Etter den første iterasjonen er ferdig vil det bli integrasjon som er hovedfokus og igjen tilbake til finpussing av alle individuelle deler i den siste iterasjonen.

6.1.2.4 Konkluderende fase

I denne fasen vil mye av tiden gå av til å skrive hovedrapport og annen tilhørende dokumentasjon. En etteranalyse skal gjøres i plenum. Denne analysen innebærer å se tilbake på hva som har blitt gjort og hvorfor ting gikk som de gjorde. Det er viktig at vi har dokumentert prosessen og resultater godt, slik at etteranalysen ikke vil bli problematisk. Brukermanual skal også skrives. I denne fasen er det resterende dokumentasjon som skal bli prioritert høyt. Det er viktig at dette er ferdig til innlevering og derfor får den hovedprioritet.

6.1.3 Iterasjoner

ACT	95 days	Mon 13.01.14	Fri 23.05.14
Innledende	29 days	Mon 13.01.14	Thu 20.02.14
Innledende iterasjon 1	29 days	Mon 13.01.14	Thu 20.02.14
Utdypende	16 days	Fri 21.02.14	Fri 14.03.14
Utdypende iterasjon 1	11 days	Fri 21.02.14	Fri 07.03.14
Utdypende iterasjon 2	5 days	Mon 10.03.14	Fri 14.03.14
Konstruksjon	45 days	Mon 17.03.14	Fri 16.05.14
Konstruksjons iterasjon 1	23 days	Mon 17.03.14	Wed 16.04.1
Konstruksjons iterasjon 2	9 days	Tue 22.04.14	Fri 02.05.14
Konstruksjons iterasjon 3	10 days	Mon 05.05.14	Fri 16.05.14
Konkluderende	5 days	Mon 19.05.14	Fri 23.05.14
Konkluderende iterasjon 1	5 days	Mon 19.05.14	Fri 23.05.14



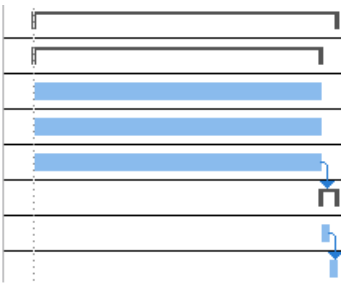
Figur 3: Gantt-diagram for alle iterasjoner

6.1.3.1 Innledende iterasjon

Den første iterasjonen er den innledende delen i prosjektet. Første iterasjon er fra prosjektstart til første presentasjon. Aktivitetene som skal jobbes med i denne delen er dokumentasjon. Der første versjon av prosjektplan, kravspesifikasjon og testspesifikasjon skal lages. Hardware skal kobles opp så alt er klart til neste iterasjon. Mot slutten av denne iterasjonen skal vi forberede en presentasjon og fremføre denne. Iterasjonen går over hele fasen og vil ha samme prioriteter som skrevet i fasen over.

Iterasjonen er ferdig 20.02.14

Innledende iterasjon 1	**29 days**	**Mon 13.01.14**	**Thu 20.02.14**
Dokumentasjon	**27 days**	**Mon 13.01.14**	**Tue 18.02.14**
Prosjektplan	27 days	Mon 13.01.14	Tue 18.02.14
Kravspesifikasjon	27 days	Mon 13.01.14	Tue 18.02.14
Testspesifikasjon	27 days	Mon 13.01.14	Tue 18.02.14
Presentasjon 1	**2 days**	**Wed 19.02.14**	**Thu 20.02.14**
Forberede presentasjon 1	1 day	Wed 19.02.14	Wed 19.02.1
Fremføre presentasjon 1	1 day	Thu 20.02.14	Thu 20.02.14

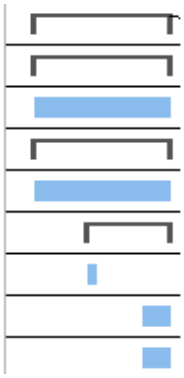


Figur 4: Gantt-diagram for innledende iterasjon

6.1.3.2 *Utdypende iterasjon #1*

Vi skal starte med å oppdatere dokumentasjon og jobbe med tilbakemeldinger fra sensorene. Vi skal også lage første versjon av nødvendige UML diagrammer(Use-case, Requirement, class, package). Iterasjonen er ferdig 07.03.14. Jobbe mod tilbakemeldinger og starte å planlegge konstruksjonsfasen(designdokument) vil ha høy prioritet.

Utdypende iterasjon 1	**11 days**	**Fri 21.02.14**	**Fri 07.03.14**
Dokumentasjon	**11 days**	**Fri 21.02.14**	**Fri 07.03.14**
Prosjektplan	11 days	Fri 21.02.14	Fri 07.03.14
Analyse og design	**11 days**	**Fri 21.02.14**	**Fri 07.03.14**
UML diagrammer	11 days	Fri 21.02.14	Fri 07.03.14
Research	**7 days**	**Thu 27.02.14**	**Fri 07.03.14**
Bazaar	1 day	Thu 27.02.14	Thu 27.02.14
TCP/IP	3 days	Wed 05.03.14	Fri 07.03.14
Bildegjenkjenning	3 days	Wed 05.03.14	Fri 07.03.14

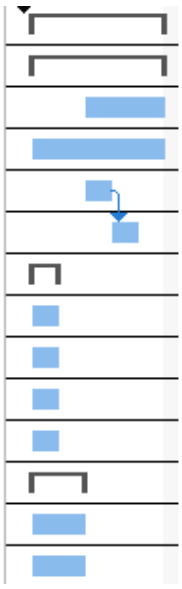


Figur 5: Gantt-diagram for utdypende iterasjon #1

6.1.3.3 *Utdypende iterasjon #2*

Alle gruppemedlemmene skal sette seg inn i sine aktiviteter og planlegge videre hvordan de skal gå fram når software skal implementeres. Alle UML diagrammer og dokumenter som skal være ferdig før konstruksjonsfasen må gjøres ferdig. Det er disse dokumentene som har hovedfokus. Iterasjonen er ferdig 14.03.14, men helgen kan brukes hvis noe gjenstår.

Utdypende iterasjon 2	**5 days**	**Mon 10.03.14**	**Fri 14.03.14**
Dokumentasjon	**5 days**	**Mon 10.03.14**	**Fri 14.03.14**
Iterasjonsdokument	3 days	Wed 12.03.14	Fri 14.03.14
Prosjektplan	5 days	Mon 10.03.14	Fri 14.03.14
Kravspesifikasjon	1 day	Wed 12.03.14	Wed 12.03.1
Testspesifikasjon	1 day	Thu 13.03.14	Thu 13.03.14
Research	**1 day**	**Mon 10.03.14**	**Mon 10.03.1**
TCP/IP	1 day	Mon 10.03.14	Mon 10.03.1
Mutual exclusion	1 day	Mon 10.03.14	Mon 10.03.1
Parsing	1 day	Mon 10.03.14	Mon 10.03.1
Bildegjenkjenning	1 day	Mon 10.03.14	Mon 10.03.1
Analyse og design	**2 days**	**Mon 10.03.14**	**Tue 11.03.14**
UML diagrammer	2 days	Mon 10.03.14	Tue 11.03.14
Design dokument	2 days	Mon 10.03.14	Tue 11.03.14

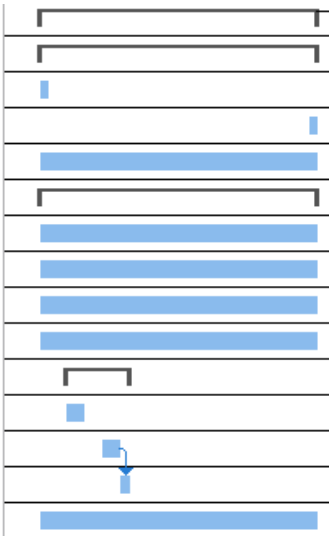


Figur 6: Gantt-diagram for utdypende iterasjon #2

6.1.3.4 Konstruksjonsiterasjon #1

Her starter gruppa med softwaredelen av prosjektet. Det første som skal gjøres er å lage rammeverket til software. Vi skal sette opp TCP/IP kommunikasjon, hente tekst fra et bilde og starte på skripting av robot. Rammeverket skal ha et fungerende utkast innen iterasjonen er ferdig. Iterasjonen er ferdig 16.04.14, men helgen kan brukes hvis noe gjenstår. De vanskeligste delene i systemet skal bli prioritert: bildegjenkjenning og kommunikasjon med parser. På grunn av fremflytting av presentasjon 2 vil software bli nedprioritert første uken i denne iterasjonen. Denne uken må vi prioritere dokumentasjon som skal være ferdig til andre presentasjon.

Konstruksjons iterasjon 1	**23 days**	**Mon 17.03.14**	**Wed 16.04.1**
Dokumentasjon	**23 days**	**Mon 17.03.14**	**Wed 16.04.1**
Iterasjonsplan	1 day	Mon 17.03.14	Mon 17.03.1
Iterasjonsrapport	1 day	Wed 16.04.14	Wed 16.04.1
Teknologidokument	23 days	Mon 17.03.14	Wed 16.04.1
Implementasjon	**23 days**	**Mon 17.03.14**	**Wed 16.04.1**
Kommunikasjon	23 days	Mon 17.03.14	Wed 16.04.1
Robot skripting	23 days	Mon 17.03.14	Wed 16.04.1
Verifikasjon	23 days	Mon 17.03.14	Wed 16.04.1
GUI	23 days	Mon 17.03.14	Wed 16.04.1
Presentasjon 2	**5 days**	**Thu 20.03.14**	**Wed 26.03.1**
Lage et fungerende utkast av system	2 days	Thu 20.03.14	Fri 21.03.14
Forberede presentasjon 2	2 days	Mon 24.03.14	Tue 25.03.14
Fremføre presentasjon 2	1 day	Wed 26.03.14	Wed 26.03.1
Testing	23 days	Mon 17.03.14	Wed 16.04.1

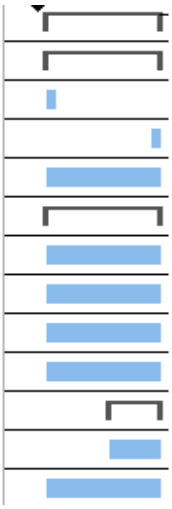


Figur 7: Gantt-diagram for konstruksjonsiterasjon #1

6.1.3.5 Konstruksjonsiterasjon #2

I denne iterasjonen skal alle de individuelle delene til systemet integreres sammen. Når alle delene er koblet sammen skal alt testes og videreutvikles. Vi skal ha et fungerende utkast av hele systemet når denne iterasjonen er ferdig. Integrasjon som er prioritert i denne iterasjonen.
Iterasjonen er ferdig 02.05.14.

Konstruksjons iterasjon 2	**9 days**	**Tue 22.04.14**	**Fri 02.05.14**
Dokumentasjon	**9 days**	**Tue 22.04.14**	**Fri 02.05.14**
Iterasjonsplan	1 day	Tue 22.04.14	Tue 22.04.14
Iterasjonsrapport	1 day	Fri 02.05.14	Fri 02.05.14
Teknologidokument	9 days	Tue 22.04.14	Fri 02.05.14
Implementasjon	**9 days**	**Tue 22.04.14**	**Fri 02.05.14**
Kommunikasjon	9 days	Tue 22.04.14	Fri 02.05.14
Robot skripting	9 days	Tue 22.04.14	Fri 02.05.14
Verifikasjon	9 days	Tue 22.04.14	Fri 02.05.14
GUI	9 days	Tue 22.04.14	Fri 02.05.14
Integrasjon	**5 days**	**Mon 28.04.14**	**Fri 02.05.14**
Sette sammen hele systemet	5 days	Mon 28.04.14	Fri 02.05.14
Testing	9 days	Tue 22.04.14	Fri 02.05.14

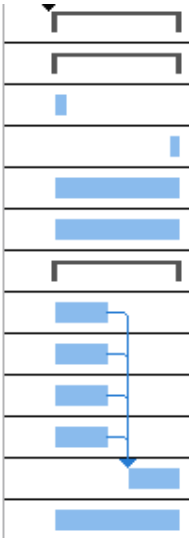


Figur 8: Gantt-diagram for konstruksjonsiterasjon #2

6.1.3.6 Konstruksjons iterasjon #3

I denne iterasjonen skal vi videreutvikle og forbedre helheten til systemet. Når denne iterasjonen er ferdig så skal vi ha laget et system som utfører minimum en test automatisk, og skrive ut en testrapport av de verifiserte resultatene. Løsningen må være robust slik at det blir mulig for KPS å jobbe videre på systemet.
Iterasjonen er ferdig 16.05.14.

Konstruksjons iterasjon 3	**10 days**	**Mon 05.05.14**	**Fri 16.05.14**
Dokumentasjon	**10 days**	**Mon 05.05.14**	**Fri 16.05.14**
Iterasjonsplan	1 day	Mon 05.05.14	Mon 05.05.1
Iterasjonsrapport	1 day	Fri 16.05.14	Fri 16.05.14
Teknologidokument	10 days	Mon 05.05.14	Fri 16.05.14
Testplan	10 days	Mon 05.05.14	Fri 16.05.14
Implementasjon	**10 days**	**Mon 05.05.14**	**Fri 16.05.14**
Kommunikasjon	5 days	Mon 05.05.14	Fri 09.05.14
Robot skripting	5 days	Mon 05.05.14	Fri 09.05.14
Verifikasjon	5 days	Mon 05.05.14	Fri 09.05.14
GUI	5 days	Mon 05.05.14	Fri 09.05.14
Finpusse software	5 days	Mon 12.05.14	Fri 16.05.14
Testing	10 days	Mon 05.05.14	Fri 16.05.14

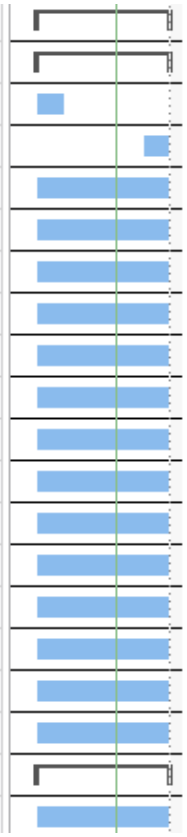


Figur 9: Gantt-diagram for konstruksjonsiterasjon #3

6.1.3.7 Konkluderende iterasjon

Dette er den siste iterasjonen i prosjektet. Her skal alt som mangler gjøres: skrive brukermanual, lage etteranalyse, finpusse dokumentasjon og sette sammen alt til et dokument. Når denne iterasjonen er ferdig skal all dokumentasjon leveres. Alt som gjenstår vil få prioritet i denne iterasjonen.
Iterasjonen er ferdig 23.05.14

Konkluderende iterasjon 1	**5 days**	**Mon 19.05.14**	**Fri 23.05.14**
Dokumentasjon	**5 days**	**Mon 19.05.14**	**Fri 23.05.14**
Iterasjonsplan	1 day	Mon 19.05.14	Mon 19.05.1
Iterasjonsrapport	1 day	Fri 23.05.14	Fri 23.05.14
Prosjektplan	5 days	Mon 19.05.14	Fri 23.05.14
Kravspesifikasjon	5 days	Mon 19.05.14	Fri 23.05.14
Testspesifikasjon	5 days	Mon 19.05.14	Fri 23.05.14
Testrapport	5 days	Mon 19.05.14	Fri 23.05.14
Komponentdokument	5 days	Mon 19.05.14	Fri 23.05.14
Software designdokumnet	5 days	Mon 19.05.14	Fri 23.05.14
Tek. Dok - Software	5 days	Mon 19.05.14	Fri 23.05.14
Tek. Dok - XML	5 days	Mon 19.05.14	Fri 23.05.14
Tek. Dok - Verifikasjon	5 days	Mon 19.05.14	Fri 23.05.14
Tek. Dok - Robot	5 days	Mon 19.05.14	Fri 23.05.14
Etteranalyse	5 days	Mon 19.05.14	Fri 23.05.14
Sluttrapport	5 days	Mon 19.05.14	Fri 23.05.14
Fremtidsdokument	5 days	Mon 19.05.14	Fri 23.05.14
Brukermanual	5 days	Mon 19.05.14	Fri 23.05.14
Software	**5 days**	**Mon 19.05.14**	**Fri 23.05.14**
Test av software og krav	5 days	Mon 19.05.14	Fri 23.05.14



Figur 10: Gantt-diagram for konkluderende iterasjon

6.1.4 Milepæler

DATO	BESKRIVELSE
20.02.14	**Første presentasjon** Den første presentasjonen til prosjektet. Prosjektplan, kravspesifikasjon og testspesifikasjon skal ha et utkast levert.
26.03.14	**Andre presentasjon** Den andre presentasjonen til prosjektet. Her skal det framføres hva som er gjort fram til nå og hva som skal gjøres fremover. Her er det fokus på den tekniske løsningen.
16.05.14	**Systemet skal være ferdig** Sluttproduktet skal være ferdig.
26.05.14	**Innlevering av prosjekt og dokumentasjon**
05.06.14	**Tredje presentasjon** Den siste presentasjonen til prosjektet. Det er på denne presentasjonen vi skal vise fram vårt endelige produkt. Dette er avslutningen på prosjektet.

Tabell 5: Milepæler

6.2 Møter

6.2.1 Ukentlige prosjektmøter

Prosjektgruppen skal ha ukentlige møter. Dette møtet vil holdes hver mandag morgen. Møtets hensikt er å oppsummere forrige uke og planlegge prosjektet videre.

6.2.2 Ukentlige møter med intern veileder

Det skal holdes ukentlige oppfølgingsmøter med intern veileder. Disse møtene vil holdes hver onsdag kl. 12.00 på HBV. Vår veileder Daniel Larsson skal ha et oppfølgningsdokument i løpet av mandag førstkommende uke. Disse møtene brukes til å gjennomgå oppfølgningsdokumentet og eventuelle spørsmål gruppa har rundt hovedprosjektet. Det skal skrives et referat fra disse møtene som skal leveres til alle deltagere innen 24 timer.

6.2.3 Møter ekstern veileder

Møter med ekstern veileder skal ikke holdes fast. Disse møtene vil avholdes etter behov. Ekstern veileder skal godkjenne alle prosjektdokumentene.

6.3 Nettside

Prosjektgruppa skal opprette en egen nettside for prosjektet. Her skal det legges ut nyheter og dokumenter fra prosjektarbeidet.

6.4 Timer

Det er laget en egen timeplan for hvert gruppemedlem i prosjektet. Denne brukes til å notere timer brukt hver uke på de forskjellige aktivitetene. Dette gjør at det er lett å holde oversikt over fremdriften på hver aktivitet i prosjektet og hvor mye tid hvert gruppemedlem har brukt. Timeplanen skal oppdateres senest fredag hver uke slik at timeplanen kan brukes på prosjektmøtet hver uke.

6.4.1 Aktivitetsliste

Dette er aktivitetene vi skal føre timer etter.

ID	BESKRIVELSE	ANSVAR-LIG	TIDS-ESTIMAT	STATUS
1xxx	**Administrativt**			
1000	Møter	ELR	40	Ferdig
1100	Web	SR	40	Ferdig
1200	Prosjektplan	HBS	200	Ferdig
1300	Timelister	SR	40	Ferdig
1400	Oppfølgingsdokument	HBS	20	Ferdig
1500	Iterasjonsdokument	SR	80	Ferdig
		SUM TID	420	

ID	BESKRIVELSE	ANSVAR-LIG	TIDS-ESTIMAT	STATUS
2xxx	**Presentasjoner**			
2000	Presentasjon 1	ELR	60	Ferdig
2100	Presentasjon 2	ELR	60	Ferdig
2200	Presentasjon 3	ELR	160	Ferdig
2300	Prosjektplakat	AKS	20	Ferdig
		SUM TID	300	

ID	BESKRIVELSE	ANSVAR-LIG	TIDS-ESTIMAT	STATUS
3xxx	**System**			
3000	Kravspesifikasjon	AKS	70	Ferdig
3100	Systemoppsett	ELR	20	Ferdig
3200	Software designdokument	AKS	100	Ferdig
3300	Teknologidokument - Software	SR	50	Ferdig
3400	Teknologidokument - XML	AKS	25	Ferdig
3500	Teknologidokument - Robot	HBS	50	Ferdig
3600	Teknologidokument - Verifikasjon	ELR	50	Ferdig
3700	Komponentdokument	AKS	20	Ferdig
3800	Fremtidsdokument	AKS	25	Ferdig
3900	Eksternt måleutstyr	HBS	40	Ferdig
		SUM TID	450	

ID	BESKRIVELSE	ANSVAR-LIG	TIDS-ESTIMAT	STATUS
4xxx	**Software**			
4000	Implementasjon	SR		Ferdig
4010	GUI	SR	20	Ferdig
4020	Kommunikasjon	SR	40	Ferdig
4030	Bildebehandling	ELR	150	Ferdig
4040	Verifikasjon og sammenligning av resultater	ELR	150	Ferdig
4050	Resultatdatabase med rapport	HBS	30	Ferdig
4060	XML støtte	AKS	60	Ferdig
4070	Systemkontroller	SR	60	Ferdig

4080	Trådbehandling	SR	30	Ferdig
4090	Feilhåndtering	ELR	30	Ferdig
4100	Annen implementasjon	ALLE	40	Ferdig
4200	Robotprogrammering	HBS	100	Ferdig
4300	Integrasjon	SR	40	Ferdig
4400	Sette seg inn i software	ALLE	60	Ferdig
		SUM TID	810	

ID	BESKRIVELSE	ANSVAR-LIG	TIDS-ESTIMAT	**STATUS**
5xxx	**Testing**			
5000	Test av software	SR	100	Ferdig
5100	Testspesifikasjon (Gjelder for 3000)	HBS	70	Ferdig
5200	Testrapport	HBS	10	Ferdig
5300	Test av krav	HBS	20	Ferdig
		SUM TID	200	

ID	BESKRIVELSE	ANSVAR-LIG	TIDS-ESTIMAT	STATUS
6xxx	**Ferdigstilling**			
6000	Sluttrapport	ELR	20	Ferdig
6100	Etteranalyse	AKS	30	Ferdig
6200	Brukermanual	SR	10	Ferdig
		SUM TID	60	

Tabell 6: Aktivitetsliste

7 Risiko

Ved å utføre en risikoanalyse kan vi identifisere eventuelle risikoer som kan oppstå slik at vi kan handle pro aktivt.

KONSEKVENS	PROSJEKTETS FREMGANG
1. Svært liten konsekvens	Prosjektfremdriften er lite påvirket
2. Liten konsekvens	Prosjektdriften opplever motgang, uten stans
3. Middels stor konsekvens	Prosjektdriften begrenses merkbart og tiltak bør vurderes
4. Stor konsekvens	Prosjektdriften stanser og tiltak er nødvendig
5. Svært stor konsekvens	Prosjektdriften blir avlyst

Tabell 7: Konsekvenskategorier

SANNSYNLIGHET	BESKRIVELSE
1. Lite sannsynlig	Sjeldnere en 1 per 1000 hendelser/time
2. Mindre sannsynlig	Gjennomsnitt 1 per 1000 hendelser/time
3. Sannsynlig	Gjennomsnitt 1 per 100 hendelser/time
4. Meget sannsynlig	Gjennomsnitt 1 per 10 hendelser/time
5. Svært sannsynlig	Oftere enn 1 per 10 hendelser/time

Tabell 8: Sannsynlighetskategorier

	KONSEKVENS				
SANNSYNLIGHET	1. Svært liten	2. Liten	3. Middels	4. Stor	5. Svært stor
5. Svært sannsynlig	5	10	15	20	25
4. Meget sannsynlig	4	8	12	16	20
3. Sannsynlig	3	6	9	12	15
2. Mindre sannsynlig	2	4	6	8	10
1. Lite sannsynlig	1	2	3	4	5

Tabell 9: Risikomatrise

Lav	Akseptabel risiko. Ingen tiltak nødvendig
Middels	Akseptabel risiko, men tiltak bør vurderes
Høy	Uakseptabel risiko, tiltak må iverksettes

7.1 Risikoanalyse

RISIKOFAKTORER	ÅRSAK	S	K	R	RISKOREDUSERENDE TILTAK
Tap av data	Feil på hardware og software.	1	5	5	Vi reduserer denne risikoen ved å ha siste versjon lagret på minst en minnebrikke, vi har et dokumenthåndteringssystem der vi regelmessig tar sikkerhetskopi av dokumenter og filer.
Analyse av data	Mangel på kunnskap rundt software.	3	3	9	Planlegge loggfiler og data osv. imot programmer vi kjenner.
Frafall i gruppa	Alvorlig sykdom, dødsfall.	1	4	4	Minst to personer jobber med de forskjellige prosjektdelene.
Dårlig oppmøte	Transport, forsoving, ferie, dødsfall i familie.	2	1	2	Ingen tiltak nødvendig under normale omstendigheter.
Skader på omgivelsene, mennesker	Ukontrollerte robot bevegelser.	2	5	10	Sette opp god sikkerhet rundt roboten. Inngjerding av risikoområde.
Mangel på kompetanse	Vanskeligere en først antatt.	3	3	9	Skaffe hjelp på et så tidlig tidspunkt som mulig, raskt vurdere andre muligheter. God kommunikasjon med veiledere.
Mangel på nødvendig testutstyr	Defekt testing.	2	4	8	Reservere testutstyr i god tid.
Tap av arbeidsplass/liv	Terrorangrep.	1	5	5	Lite kan gjøres.
Sykdom	N/A.	3	2	6	Ingen tiltak nødvendig under normale omstendigheter.
Dårlig motivasjon	Uenigheter, skjev arbeidsfordeling, dårlig planlegging.	2	2	4	Ingen tiltak nødvendig under normale omstendigheter.

Tabell 10: Risikoanalyse

(S = sannsynlighet, K = konsekvens, R = risiko. R = S×K)

Formålet med en risikoanalyse er å kartlegge farer og problemer for å få bakgrunn til å vurdere risiko, samt utarbeide planer og tiltak for å redusere risikoforholdene. En risikoanalyse skal inneholde følgende punkter:

- Hva som kan gå galt
- Hva sannsynligheten er for at det går galt
- Hvilke konsekvenser det utløser
- Tiltak får redusere konsekvensene

Ved å gjøre risikoanalysen er vi bedre forberedt på hva som kan gå galt og hva vi skal gjøre dersom dette inntreffer. Gruppens risikoanalyse fins i tabell 13 Risikoanalyse.

8 Økonomi

I møter med bedriften har vi tatt opp temaet økonomi. Vi ble enige om at dersom vi trenger noe spesielt utstyr for å gjennomføre prosjektet, så går bedriften til innkjøp av nødvendig utstyr.

Kostnader som faller på prosjektgruppa er:

BESKRIVELSE	KOSTNAD
Servering ved presentasjon	~1000kr
Prosjektplakat	~1000kr
Printing av prosjektdokumentasjon til siste presentasjon	~1000kr

Tabell 11: Kostnader

9 Software-oversikt

Under prosjektet vil vi benytte oss av flere software-applikasjoner for ulike arbeidsoppgaver:

Program	Hensikt
Visual Studio	Koding av programvare
Visual Paradigm	UML-design
Dropbox	Oppbevaring og felles tilgjengelighet av prosjektfiler
Microsoft Excel	Prosjektorganisering
Microsoft Word	Dokumentskriving
Microsoft Project	Gantt-diagrammer
Microsoft SQL Management Studio 2012	Database til rapport
Adobe Dreamweaver	Webside
FileZilla	FTP server (bildeuthenting fra våpenstasjon)
Notepad	XML dokumenter
Solidworks	Illustrasjon av mobilt komponentbord

Tabell 12: Software liste

10 Dokumentasjon

En stor del av bachelorprosjektet er dokumentasjonen som beskriver alt som er gjort i prosjektet. Dokumentene skal inneholde problemstilling, prosjektprosessen, resultater og konklusjon.

10.1 Versjonshåndtering

For å lettere finne endringer i prosjektets dokumenter har prosjektgruppa valgt å benytte seg av et versjonshåndteringssystem. Alle versjoner av et dokument skal tildeles et versjonsnummer. Versjonsnummereringen består av to tall, for eksempel 1.0. Tallet før punktum er offisielle utgivelser, mens tallet bak punktum beskriver interne versjoner.

Det vil være en offisiell utgivelse for hver presentasjon. Første presentasjon vil ha versjon 1.0, andre presentasjon vil ha versjon 2.0 og tredje presentasjon vil ha versjon 3.0.

Alle versjoner vil loggføres i en tabell i starten av alle dokumentene. Her vil det stå hvilken versjon dokumentet er og hvilke endringer som er gjort fra forrige versjon. Det vil også være en signatur fra den personen som har gjort endringene på dokumentene. Endringer i dokumenter vil ikke logges før etter første utgivelse, før dette vil all dokumentasjon anses som utkast.

10.2 Dokumentoversikt

Tabellen under er alle dokumentene listet opp. Det er også informasjon om hvem som er ansvarlige for de forskjellige dokumentene og planlagt utgivelse samt første utgivelse. Listen vil bli oppdatert under hele prosjektperioden.

DOKUMENT	ANSVARLIG	PLANLAGT FERDIG	FØRSTE GANG UTGITT
Prosjektplan	HBS	Første presentasjon	17.02.14
Kravspesifikasjon	AKS	Første presentasjon	17.02.14
Testspesifikasjon	HBS	Første presentasjon	17.02.14
Software designdokument	AKS	Andre presentasjon	24.03.14
Iterasjonsdokument	SR	Andre presentasjon	24.03.14
Testrapport	HBS	Tredje presentasjon	23.05.14
Sluttrapport	HBS	Tredje presentasjon	25.05.14
Brukermanual	ELR	Tredje presentasjon	25.05.14
Etteranalyse	AKS	Tredje presentasjon	25.05.14
Fremtidsdokument	AKS	Tredje presentasjon	25.05.14
Komponentdokument	AKS	Tredje presentasjon	22.05.14
Teknologidokument - XML	AKS	Tredje presentasjon	22.05.14
Teknologidokument - Robot	HBS	Tredje presentasjon	22.05.14
Teknologidokument - Software	SR	Tredje presentasjon	25.05.14
Teknologidokument - Verifikasjon	ELR	Tredje presentasjon	25.05.14

Tabell 13: Dokumentoversikt

ACT AUTOMATED CROWS TESTING

KRAVSPESIFIKASJON

AUGUST KIND SVENDSEN
EIRIK LIEN ROA
HENRIK BERGE SØRUM
STÅLE RUDIN

KONGSBERG

HBV
HØGSKOLEN I BUSKERUD OG VESTFOLD · PROFESJONSHØGSKOLEN ·

Kravspesifikasjon

PROSJEKT	ACT - Automated CROWS Testing		
OPPDRAGSGIVER	Kongsberg Protech Systems		
UTFØRT VED	HBV - Høgskolen i Buskerud og Vestfold		
GRUPPE	August Kind Svendsen, Eirik Lien Roa, Henrik Berge Sørum og Ståle Rudin		
VERSJON	2.0		
ANTALL SIDER	7		
DOKUMENTHISTORIE	VERSJON	UTGITT	BESKRIVELSE
	1.0	17.02.2014	Første utgivelse
	2.0	24.02.2014	Andre utgivelse

INNHOLDSFORTEGNELSE

LISTE OVER TABELLER

1 Om dokumentet

1.1 Dokumenthistorie

VERSJON	DATO	ENDRING	SIGNATUR
0.1	27.01.14	• Dokumentet ble opprettet	AKS, HBS
1.0	17.02.14	• Første utgivelse	AKS, ELR, HBS, SR
1.1	12.03.14	• Total renovering i samarbeid med KPS	AKS, ELR, HBS, SR
1.2	20.03.14	• Fortsettelse på renovering med KPS	AKS, ELR, HBS, SR
2.0	21.03.14	• Oppdatert dokumentet og sett over skrivefeil osv.	ELR

Tabell 1: Dokumenthistorie

1.2 Definisjoner og forkortelser

Liste som beskriver ord og uttrykk som kan være brukt i dette dokumentet:

UTTRYKK	FORKLARING
HBV	Høgskolen i Buskerud og Vestfold
KPS	Kongsberg Protech Systems (oppdragsgiver)
ACT	Automated CROWS Testing
Gruppa	Prosjektgruppa
Industrirobot	Universal Robots 5
CROWS	Common Remotely Operated Weapon Station
DCP	Display Control Panel
MPU	Main Processing Unit
RWS	Remote Weapon Station

Tabell 2: Definisjoner og forkortelser

2 Innledning

Dette dokumentet er en kravspesifikasjon laget av prosjektgruppa ACT.

Dokumentet dekker de kravene vi har satt for at vi skal oppnå ønskede resultater ved endt prosjekt. Kravene er satt opp av prosjektgruppa og arbeidsgiver.

Hensikten med dette dokumentet er å skape en felles forståelse mellom prosjektgruppe og arbeidsgiver om hva resultatet av prosjektet skal bli.

Dokumentet inneholder alle krav til prosjektet og er delt inn i følgende kategorier: Rammekrav, sikkerhetskrav, funksjonelle krav, utstyrskrav og andre krav.

Kravspesifikasjonen legger grunnlaget for testspesifikasjonen [1], som beskriver hvordan man skal kunne teste at de forskjellige kravene blir oppfylt og opprettholdt.

3 Krav

Vi har delt opp kravene i fem kategorier, under står de nærmere beskrevet:

- Rammekrav (Rammebetingelser som må oppfylles for at oppgaven skal betraktes som gjennomført, se tabell 3)
- Sikkerhetskrav (Krav som tar vare på sikkerheten for personer og utstyr, se tabell 4)
- Funksjonelle krav (Hva systemet skal gjøre, se tabell 5)
- Utstyrskrav (Krav som gjør arbeidsflyten enklere, se tabell 6)
- Andre krav (Krav som skal opprettholdes for å forbedre resultatet av prosjektet, se tabell 7).

3.1 Kravprioritet

Kravene er delt inn i disse prioriteringene:

- A-krav: Krav som må oppfylles for at prosjektet skal kunne fullføres, eller som er viktig med tanke på sikkerhet.
- B-krav: Viktige krav som bør være på plass, men ikke er nødvendige for at prosjektet skal fullføres.
- C-krav: Krav som vil heve kvaliteten på systemet, men som kun oppfylles om tiden strekker til.

3.2 Rammekrav

KRAV	PRIO	DATO	UTSTEDER	BESKRIVELSE
RK.1	A	12.03.14	KPS	Selvstendig system med minst mulig manuell assistanse. Systemet skal gjennomføre testsekvens selvstendig etter kalibrering og igangsetting.
RK.2	A	12.03.14	KPS	Systemet skal håndtere feilsituasjoner på en forutsigbar og sikker måte.
RK.3	B	12.03.14	KPS	Skal være mulig å legge til flere tester uten å endre kompilert kode.
RK.4	A	12.03.14	KPS	Testsekvenser skal loggføres og testresultat dokumenteres.

Tabell 3: Rammekrav

3.3 Sikkerhetskrav

KRAV	PRIO	DATO	UTSTEDER	BESKRIVELSE
SK.1	A	12.03.14	KPS	Systemet skal avbryte kjøring ved fatal feil under test.
SK.2	A	12.03.14	KPS	Software må kunne måle systemtilstander.
SK.3	A	12.03.14	KPS	Systemet skal kunne avbrytes manuelt.

Tabell 4: Sikkerhetskrav

3.4 Funksjonelle krav

KRAV	PRIO	DATO	UTSTEDER	BESKRIVELSE
FK.1	A	27.01.14	KPS	Systemet skal utføre tester på CROWS automatisk.
FK.2	A	12.03.14	KPS	Systemet skal verifisere resultatene iht. CROWS software-krav.
FK.2.1	B	12.03.14	KPS	Systemet skal kunne verifisere CROWS software-krav ved bruk av bildegjenkjenning.
FK.2.2	A	12.03.14	KPS	Systemet skal kunne verifisere CROWS software-krav ved bruk av data mottatt fra våpenstasjon.
FK.3	B	12.03.14	KPS	Systemet må kunne lese av elektriske tilstander(strøm/spenning) via eksternt utstyr på våpenstasjonen.
FK.4	B	12.03.14	KPS	Systemet må ha en hensiktsmessig oppdateringsfrekvens til ulike deler av systemet.

Tabell 5: Funksjonelle krav

3.5 Utstyrskrav

KRAV	PRIO	DATO	UTSTEDER	BESKRIVELSE
UK.1	B	27.01.14	Gruppa	Må ha tilgang til samme Protector CROWS konfigurasjon gjennom hele prosjektet.
UK.2	A	14.02.14	Gruppa	Får arbeide på samme versjon av software til CROWS systemet gjennom hele prosjektet. Versjonen som vil bli brukt er CROWS 3.12.
UK.3	A	20.03.14	KPS	Må bruke vernesko med ESD.
UK.4	A	20.03.14	Gruppa	Ha tilgang til robot gjennom hele prosjektet.
UK.5	A	20.03.14	Gruppa	Ha nødvendig tilgang til utstyr og fasiliteter.

Tabell 6: Utstyrskrav

3.6 Andre krav

KRAV	PRIO	DATO	UTSTEDER	BESKRIVELSE
AK.1	C	27.01.14	KPS	Lage brukerhåndbok for bruk av systemet.
AK.2	C	20.03.14	KPS	Lage system og design dokumentasjon.

Tabell 7: Andre krav

4 Referanser

REFERANSE	DOKUMENTTITTEL	VERSJON
[1]	Testspesifikasjon	3.0

Tabell 8: Referanser

ACT AUTOMATED CROWS TESTING

SOFTWARE DESIGNDOKUMENT

AUGUST KIND SVENDSEN
EIRIK LIEN ROA
HENRIK BERGE SØRUM
STÅLE RUDIN

KONGSBERG

HBV
HØGSKOLEN I BUSKERUD OG VESTFOLD
PROFESJONSHØGSKOLEN

Software designdokument			
PROSJEKT	ACT - Automated CROWS Testing		
OPPDRAGSGIVER	Kongsberg Protech Systems		
UTFØRT VED	HBV - Høgskolen i Buskerud og Vestfold		
GRUPPE	August Kind Svendsen, Eirik Lien Roa, Henrik Berge Sørum og Ståle Rudin		
VERSJON	2.0		
ANTALL SIDER	22		
DOKUMENTHISTORIE	VERSJON	UTGITT	BESKRIVELSE
	1.0	24.03.14	Første utgivelse
	2.0	26.05.14	Andre utgivelse

INNHOLDSFORTEGNELSE

LISTE OVER TABELLER

LISTE OVER FIGURER

1 Om dokumentet

1.1 Dokumenthistorie

VERSJON	DATO	ENDRING	SIGNATUR
0.1	14.03.14	• Dokumentet ble opprettet	AKS
1.0	21.03.14	• Dokument ferdig utfylt	AKS, HBS
2.0	23.05.14	• Endelig versjon ferdig	AKS

Tabell 1: Dokumenthistorie

1.2 Definisjoner og forkortelser

Liste som beskriver ord og uttrykk som kan være brukt i dette dokumentet:

UTTRYKK	FORKLARING
HBV	Høgskolen i Buskerud og Vestfold
KPS	Kongsberg Protech Systems (oppdragsgiver)
ACT	Automated CROWS Testing
Industrirobot	Universal Robots 5
CROWS	Common Remotely Operated Weapon Station
RWS	Remote Weapon Station
DCP	Display Control Panel
MPU	Main Processing Unit
Konfigurasjon	Forskjellige CROWS oppsett
Firing controller	Eksternt instrument som måler fyringsspenningen

Tabell 2: Definisjoner og forkortelser

2 Innledning

Hensikten med dette dokumentet er å beskrive den planlagte arkitekturen til ACT-systemet som skal utvikles i sammenheng hovedprosjekt for KPS. Det vil bli brukt som et planleggingsdokument, og bli brukt som et veiledende verktøy underveis i utviklingen.

Når vi nærmer oss prosjektslutt og et ferdig produkt, vil vi gå tilbake til dette dokumentet for å analysere og sammenligne resultatet med hva vi hadde planlagt. Dette vil dokumenteres i kapitellet «Implementasjonsperspektivet».

En grundigere beskrivelse av implementasjonen og systemets endelige oppbygning vil dokumenteres i de forskjellige teknologidokumentene[2] og Sluttrapporten[3].

2.1 Avgrensning

Oppsummert skal systemet vi lager gjøre enkle funksjonstester på CROWS produsert av KPS. Planen er at en industrirobot skal stå for det fysiske arbeidet som tidligere har blitt gjort manuelt. Systemet består av flere hardware og software-komponenter som må kommunisere sammen for at funksjonene skal fungere. Det må også til en hver tid holde kontroll på hvor i testsekvensen man er, og verifisere resultater og dokumentere disse resultatene.

3 Bakgrunn og oversikt

I denne delen går vi over grunner og årsaker til hvorfor designet har blitt formet slik det har. Dette er preget av systemkrav, avgrensninger og funksjoner systemet forventes å ha.

3.1 Funksjonelle krav

Arkitekturen til systemet er tydelig påvirket av kravene, spesielt de funksjonelle kravene. Her er en oppsummering av de krav som har vært drivende for arkitekturen, hentet fra kravspesifikasjonen[1]:

FK.1 – Systemet skal utføre tester på CROWS automatisk.
FK.2 – Systemet skal verifisere resultatene iht. CROWS software-krav.
FK.3 – Systemet skal kunne verifisere ved bruk av bildegjenkjenning.
FK.4 – Systemet skal kunne verifisere ved bruk av data mottatt fra våpenstasjon.
FK.5 – Systemet må kunne måle elektriske tilstander (strøm/spenning) på våpenstasjonen.
FK.6 – Systemet må ha en hensiktsmessig oppdateringsfrekvens til ulike deler av systemet.

3.2 Ikkefunksjonelle krav

Selv om de funksjonelle kravene veier mest på hvordan arkitekturen settes opp, er det også noen ikkefunksjonelle krav som vil ha en påvirkning. Dette har vi satt opp som rammekrav og sikkerhetskrav, og står listet opp under.

RK.1 – Selvstendig system med minst mulig manuell assistanse. Systemet skal gjennomføre testsekvens selvstendig etter kalibrering og igangsetting.
RK.2 – Systemet skal håndtere feilsituasjoner på en forutsigbar og sikker måte.
RK.3 – Skal være mulig å legge til flere tester uten å endre kompilert kode.
RK.4 – Testsekvenser skal loggføres og testresultat dokumenteres.

SK.1 – Systemet skal avbryte kjøring ved fatal feil under test.
SK.2 – Software må kunne måle systemtilstander.
SK.3 – Systemet skal kunne avbrytes manuelt.

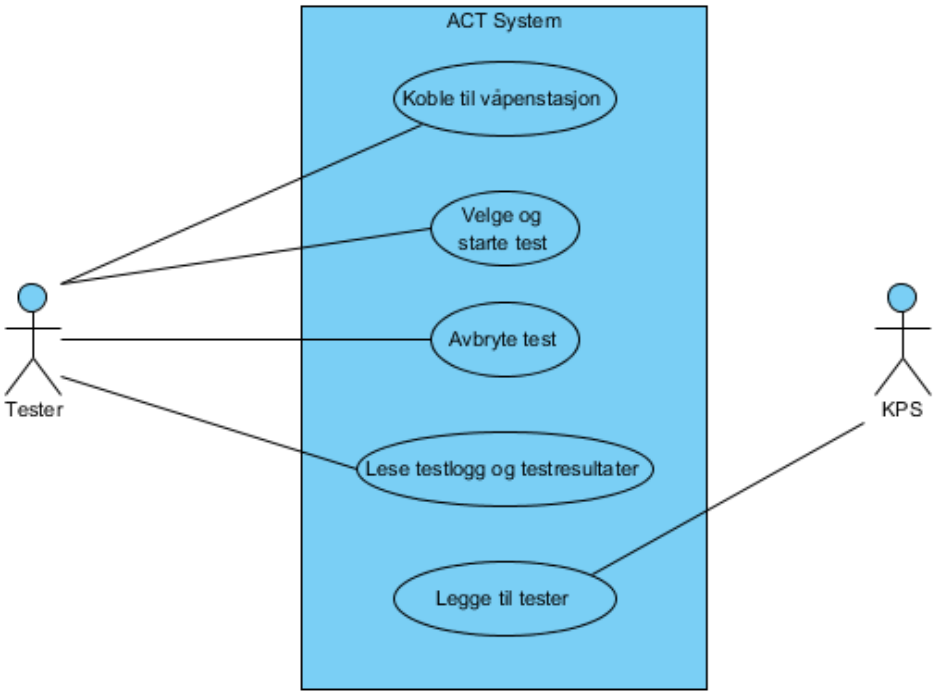
Som vi kan se er det flere av de ikkefunksjonelle kravene som vil spille en viktig rolle i utviklingen av systemet. Rammekrav 3 krever at vi lager et modulært system, der en enkelt kan legge til og endre tester til systemet. Dette har vi tenkt å løse ved å la testene være egne filer som blir lest av programmet vi lager, for eksempel som .xml-filer. På denne måten kan brukerne av systemet enkelt lage sin egen testsekvens uten å måtte kode i selve programmet.

Rammekrav 4, testsekvenser skal loggføres og testresultat dokumenteres, er et krav som også setter sitt preg på arkitekturen. Dette krever at vi lager programmet slik at vi til en hver tid kan lese hvilken posisjon og status våpenstasjonen er i, slik at vi kan dokumentere at testen faktisk har funnet sted.

3.3 Use case diagram

Her ser vi de use case-ene som har vært bestemmende for arkitekturen. Vi ser hva brukeren kan forvente av use case-ene ved bruk av systemet. Et typisk bruk av systemet innebærer å koble seg til våpenstasjonen, deretter velge den ønskede testprosedyren. Fra dette punktet vil en automatisk testing av våpenstasjonen skje. Skulle det oppstå en situasjon der brukeren ønsker å avbryte testsekvensen vil personen ha mulighet til å avbryte testen når som helst. Når ACT-systemet er ferdig med å teste våpenstasjonen vil det printes ut en testlogg som har dokumentert hva som har skjedd underveis i testen, inkludert testresultater.

Om arbeidsgiver ønsker å legge til flere tester, eventuelt modifisere en test, så kan vi se på modellen at de skal ha muligheten til dette. Siden dette ikke nødvendigvis er en som bruker systemet til testing ellers, så har vi vist dette ved å sette opp en egen aktør for dette.



Figur 1: Use case diagram

3.4 Avhengigheter

ACT-systemet er hovedsakelig avhengig av to andre eksterne systemer. Det ene er systemet som utgjør industriroboten består av robotarmen og en tilhørende PC som holder på et program som kan kjøre scriptet som styrer armen.

Det andre systemet vi er avhengig av er CROWS oppsettet. Dette er et system bestående av en DCP (monitor med kontrollpanel), MPU (signalbehandler mellom DCP og WS) og selve våpenstasjonen. Vi kommer til å hente ut forskjellige signaler fra alle nevnte komponenter, ettersom hva testen krever.

4 Arkitekturbeskrivelsen

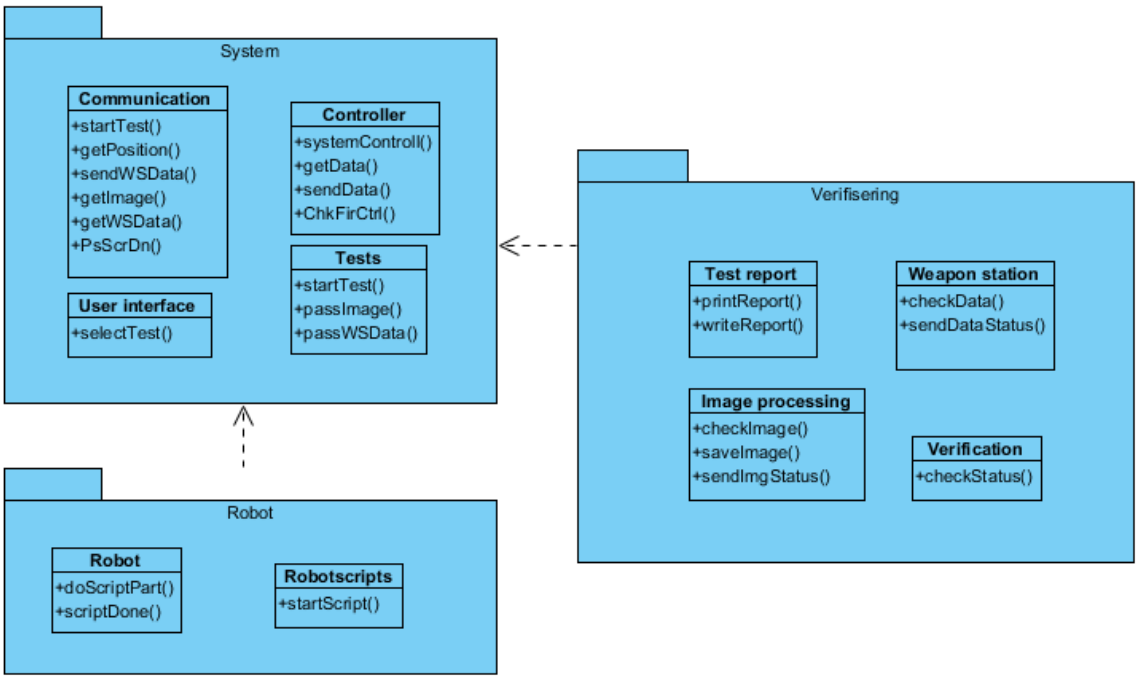
I dette kapittelet skal vi beskrive den planlagte arkitekturen for systemet. For å klargjøre planleggingen kommer vi til å beskrive dette fra forskjellige perspektiver. Først vil vi se på det logiske perspektivet, deretter prosess- og deploymentperspektivet, og helt tilslutt fra et implementasjonsperspektiv. Implementasjonsperspektivet vil føres inn ettersom vi implementerer, og her vil vi sammenligne det endelige produktet med designet vi planla i dette dokumentet. Hva de forskjellige perspektivene innebærer vil bli beskrevet nærmere i hvert sitt underkapittel.

4.1 Logisk perspektiv

Her beskriver vi organiseringen av programvaren i form av subsystemer, pakker, klasser og grensesnitt. Vi vil gå over det arkitektoniske mønstret og beskrive hvorfor vi har valgt å bygge opp programmet slik vi har gjort, og hvordan denne oppbygningen er med på å tilfredsstille kravene.

For å få et innblikk i organiseringen av klassene starter vi med pakkediagrammet. Her ser vi de forskjellige klassene som er planlagt å bruke, der de er delt inn i forskjellige pakker sammen nærliggende klasser. Det vil si klasser som har et tett samarbeid med hverandre.

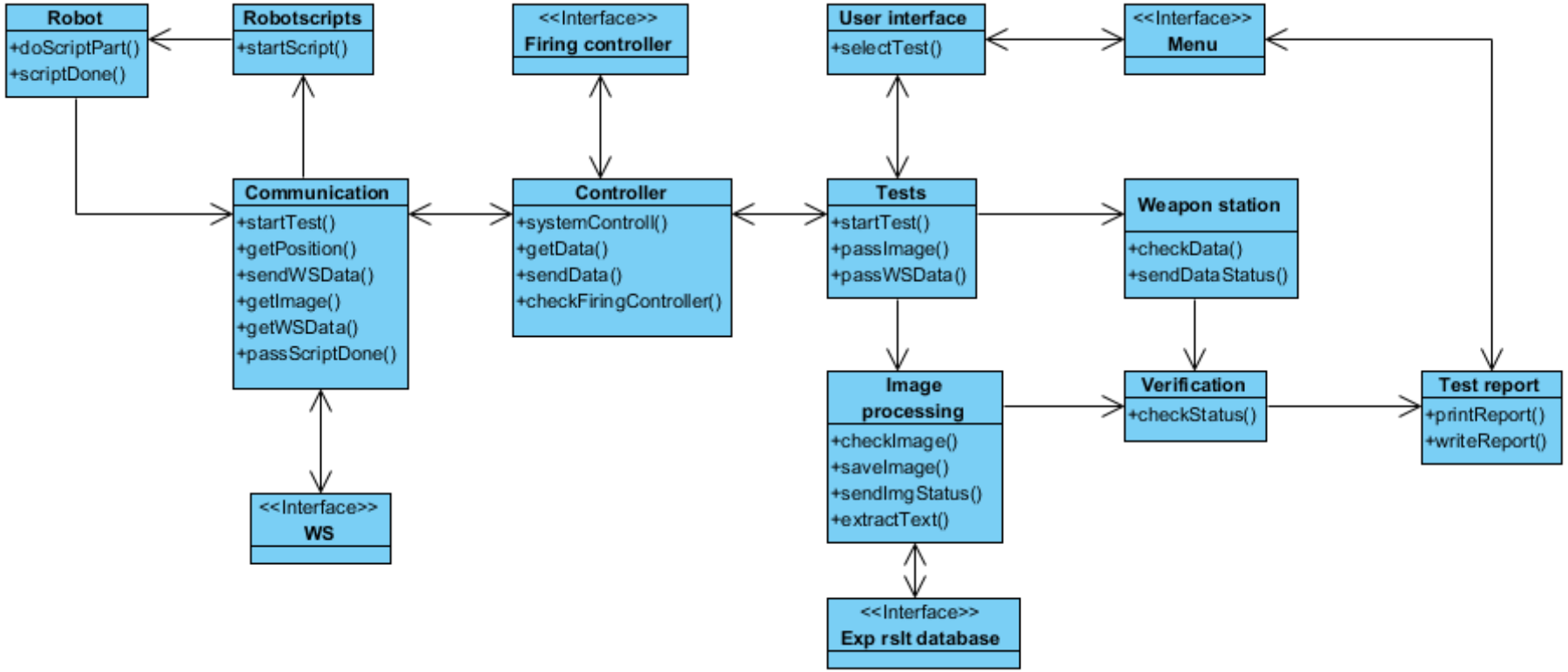
Vi har hovedsakelig delt opp systemet inn i tre pakker; system, verifisering og robot. Systempakken kan sees på som hovedpakken der de grunnleggende klassene ligger. Her finner vi klasser som har ansvar for kommunikasjon, testoversikt, kontroll, og det relatert til grensesnittet. Verifiseringspakken holder på de klassene som relatert til verifisering av testresultater, samt den klassen som setter sammen resultater til noe brukeren kan lese. Til slutt har vi robotpakken som har ansvar for å sende bevegelsesbeskjeder til roboten, samt holde på de scripts som skal kjøres.



Figur 2: Pakkediagram

4.1.1 Overordnet klassemodell

Klassediagrammet viser et foreløpig design av klassene til systemets software. Vi får også frem hvordan klassene er bygget opp og hvilke metoder de inneholder. UML diagrammer lages i hovedsak for å vise en modell av hvordan software skal se ut, der klassediagrammer er en av de mest sentrale diagrammene. Klassemodellen vil endres underveis i konstruksjonen hvis vi finner bedre løsninger eller finner ut at noe ikke fungerer som planlagt. Det vil også bli fylt inn mer detaljert hva hver klasse inneholder når vi vet mer spesifikt hva som trengs i de forskjellige klassene.



Figur 3: Klassediagram

4.1.1.1 Detaljert beskrivelse av «User interface»-klassen

Hensikt: Denne klassen er å holde på de grafiske elementene som vises i brukergrensesnittet.
Pakke: Denne klassen tilhører systempakken.
Operasjoner: Bruker metoden «select test» for å velge ut en test, som videreformidler dette til testklassen.

4.1.1.2 Detaljert beskrivelse av «Test»-klassen

Hensikt: Denne klassen skal holde på alle tester, og de egenskapene som hører med der. Denne klassen kan sees om en klasse som videreformidler informasjonen de andre klassene trenger for å vite hvilken test de holder på med.
Pakke: Denne klassen tilhører verifiseringspakken.
Operasjoner: startTest() metoden er den metoden som starter en test. Den skal da starte den testen som ble valgt i "User inferface" klassen og sende nødvendig informasjon videre.
passImage() metoden skal videresende bilde fått fra Controller-klassen til Image processing-klassen.
passData() metoden skal videresende data mottatt fra Controller-klassen til Weapon station-klassen.

4.1.1.3 Detaljert beskrivelse av «Controller»-klassen

Hensikt: Dette er selve kontrolløren til softwaren vi skal kode. Denne klassen skal vite om all informasjon som blir sendt mellom klasser så den hele tiden kan følge med på hvor man er i en test. Da kan den videreformidle informasjon til klasser som skal bruke dette. All informasjon som skal bli sendt gjennom systemet våres må innom denne klassen før den blir sendt videre. Dette gjøres så denne klassen alltid vet hvor i testsekvensen programmet er og derfor kan styre hvilken klasse som kan kjøre. Implementasjon av «mutual exclution» er viktig i denne klassen.
Pakke: Denne klassen tilhører systempakken.
Operasjoner: systemControll() metoden er metoden som står får kontrollering av trafikkflyt. Denne metoden vil bestemme når klassen skal videresende data og når klasser kan starte på sine operasjoner. getData() metoden henter nødvendig data fra Tests- eller Communication-klassen.
sendData() metoden sender nødvendig data til Tests- eller Communication-klassen.
checkFiringController() metoden skal sjekke status på fyringsmekanismen når dette er nødvendig.

4.1.1.4 Detaljert beskrivelse av «Communication»-klassen

Hensikt: Hensikten med denne klassen er å kommunisere med eksterne enheter. Klassen inneholder funksjoner som skal sende og motta informasjon fra våpenstasjon og robot.
Pakke: Denne klassen tilhører systempakken.
Operasjoner: startTest() metoden skal sende beskjed til robot om hvem test som skal startes, denne vil da inneholde hvilke scripts som må trengs for å utføre den bestemte testen.
getPosition() metoden skal hente ut posisjonen til våpenstasjonen.
sendWSData() metoden skal sende informasjon som er nødvendig til våpenstasjonen (WS-interface).
getImage() metoden skal hente ut bilde fra våpenstasjonen når dette er nødvendig.
getWSData() metoden skal hente ut data fra våpenstasjonen når dette er nødvendig.
passScriptDone() metoden sender beskjed til Controller-klassen når klassen har mottatt beskjed fra Robot-klassen at en bevegelse er utført.

4.1.1.5 Detaljert beskrivelse av «Robot scripts»-klassen

Hensikt: Hensikten med denne klassen er å holde på bevegelsesmønstrene til roboten. Det vil si at den holder på en sekvens av navn på bevegelser som utgjør en testsekvens. Disse deles inn i deler, der hver del tilsvarer den bevegelsen som kreves til deltesten som utføres.
Pakke: Denne klassen tilhører robotpakken.
Operasjoner: startScript() metoden inneholder alle bevegelsene til roboten. Denne mottar en beskjed fra Communication-klassen om hvilke bevegelse som skal utføres. Når den vet hvilken bevegelse som skal utføres sender klassen bevegelsen til Robot-klassen.

4.1.1.6 Detaljert beskrivelse av «Robot»-klassen

Hensikt: Denne klassen holder på alle bevegelser roboten kan gjøre. Hvilke bevegelser som skal gjøres får den beskjed fra robot scripts- klassen. Alle bevegelsene som er nødvendig for at den skal utføre en test må utføres i riktig sekvens.
Pakke: Denne klassen tilhører robotpakken.
Operasjoner: doScriptPart() metoden utfører en og en bevegelse som den har mottatt fra robot script-klassen. scriptDone() metoden sender en beskjed til Communication-klassen om at scriptet er utført.

4.1.1.7 Detaljert beskrivelse av «Image processing»-klassen

Hensikt: Hensikten med denne klassen er å prosessere bilde. Det er her bildeverifikasjon blir utført. Denne klassen skal hente ut tekst fra bilde og sammenligne teksten med en "korrekt" tekst.
Pakke: Denne klassen tilhører verifiseringspakken.
Operasjoner: checkImage() metoden skal hente ut tekst fra mottatt bilde og sjekke denne teksten opp mot en korrekt tekst. All den korrekte teksten vil ligge i en database.
sendImgStatus() metoden sender verifikasjonsstatus til Verification-klassen.
extractText() metoden henter ut den tekst fra databasen som skal sammenlignes med teksten som er tatt ut fra bildet. saveImage() metoden skal lagre bildene hentet fra våpenstasjon til et lagringsmedium.

4.1.1.8 Detaljert beskrivelse av «Weapon station»-klassen

Hensikt: Hensikten med denne klassen er å verifisere data som er hentet fra våpenstasjonen. Data som posisjon og status på knapper er data som skal verifiseres i denne klassen.
Pakke: Denne klassen tilhører verifiseringspakken.
Operasjoner: checkData() metoden skal verifisere data som er hentet fra våpenstasjonen opp mot riktige data. sendDataStatus() metoden sender verifikasjonsstatus til Verification-klassen.

4.1.1.9 Detaljert beskrivelse av «Verification»-klassen

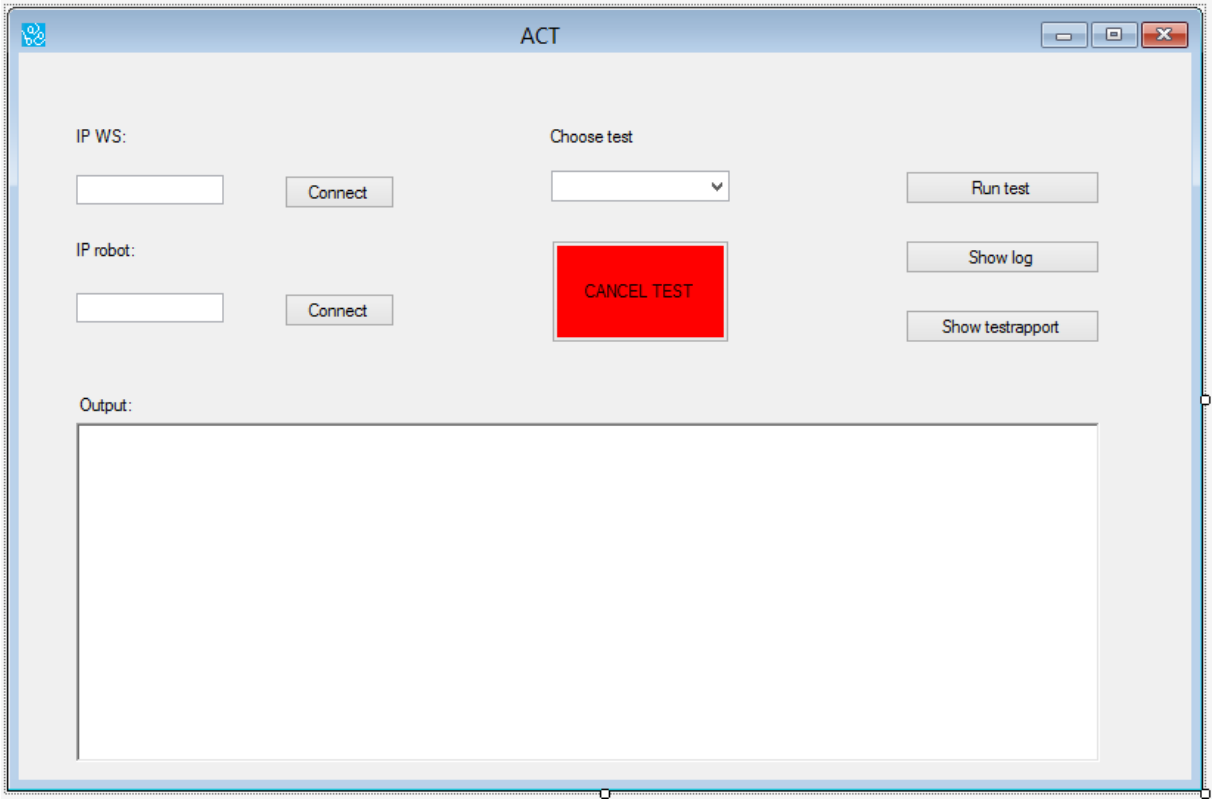
Hensikt: Hensikten med denne klassen er å motta status fra Weapon station-klassen og Image processing-klassen og sjekke om disse resultatene er «ok» eller «ikke ok».
Pakke: Denne klassen tilhører verifiseringspakken.
Operasjoner: checkStatus() metoden skal sjekke om resultater mottatt fra Weapon station-klassen og Image processing-klassen er ok eller ikke. Metoden skal deretter sende denne beskjeden videre til Test report-klassen.

4.1.1.10 Detaljert beskrivelse av «Test report»-klassen

Hensikt: Denne klassen skal fylle inn resultater i en rapport og skrive ut disse.
Pakke: Denne klassen tilhører verifiseringspakken.
Operasjoner: writeReport() metoden skal fylle in resultater mottatt fra Verification-klassen i rapporten. printReport() metoden skal skrive ut rapporten når den er ferdig utfylt.

4.1.2 Brukergrensesnittet

Her ser vi et skjermbilde av hvordan vi planlegger å lage brukergrensesnittet. Vi ser her at brukeren enkelt kan taste inn IP-adresse til både våpenstasjon og robot. Deretter beveger brukeren seg til høyre i vinduet der den ønskede testen velges. Tilslutt presser personen bare på «Run test», og testsekvensen vil starte. Om det skulle oppstå en situasjon der brukeren ønsker å avbryte testen er det bare å trykke på «Cancel test»-knappen, da vil testen avbrytes momentant. Ved endt fullført testsekvens kan brukeren velge å trykke på «Show log» eller «Show test report», avhengig om brukeren vil se logg eller rapport. Dette vil da vises i «Output» vinduet, men vil også lagres som et eget dokument.



Figur 4: Utkast av grensesnitt

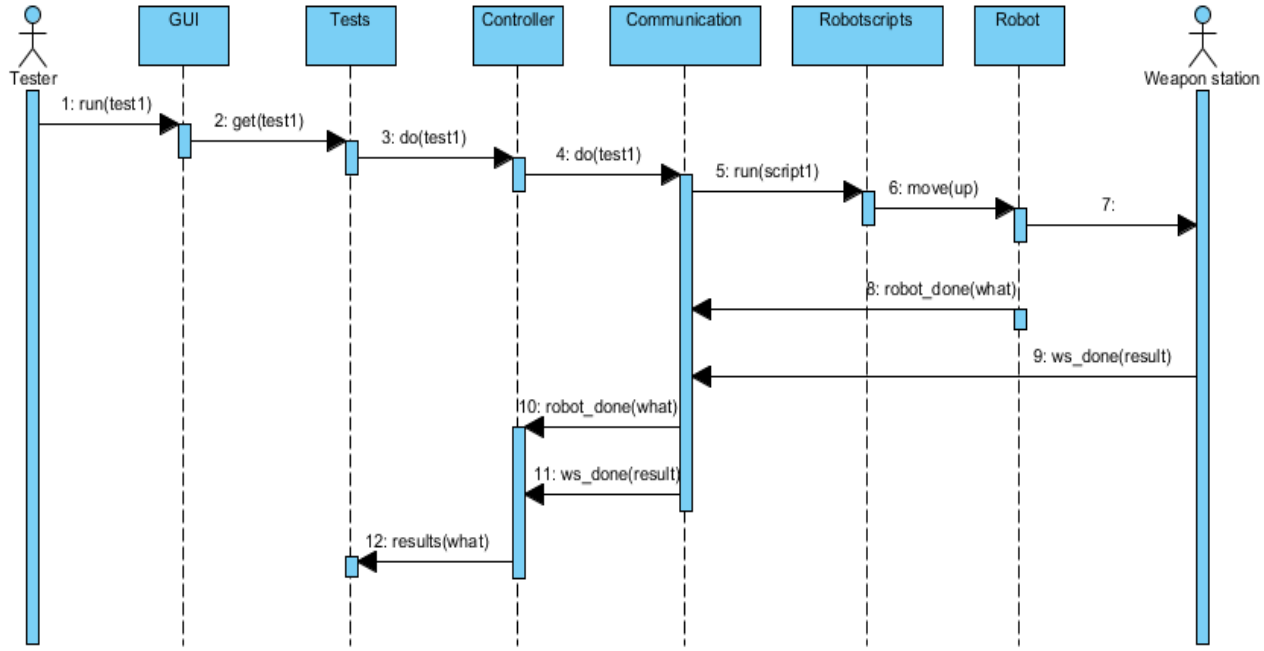
4.2 Prosessperspektivet

I denne delen skal vi ta for oss hvordan de forskjellige klassene kommuniserer mellom hverandre, interfacet til våpenstasjonen og til «firing controller». I tillegg til hva slags informasjon som sendes mellom partene vil det også komme frem i hvilken rekkefølge eller sekvens de forskjellige metodene vil opptre. Først tar vi for oss sekvensdiagrammene.

4.2.1 Sekvensdiagrammer

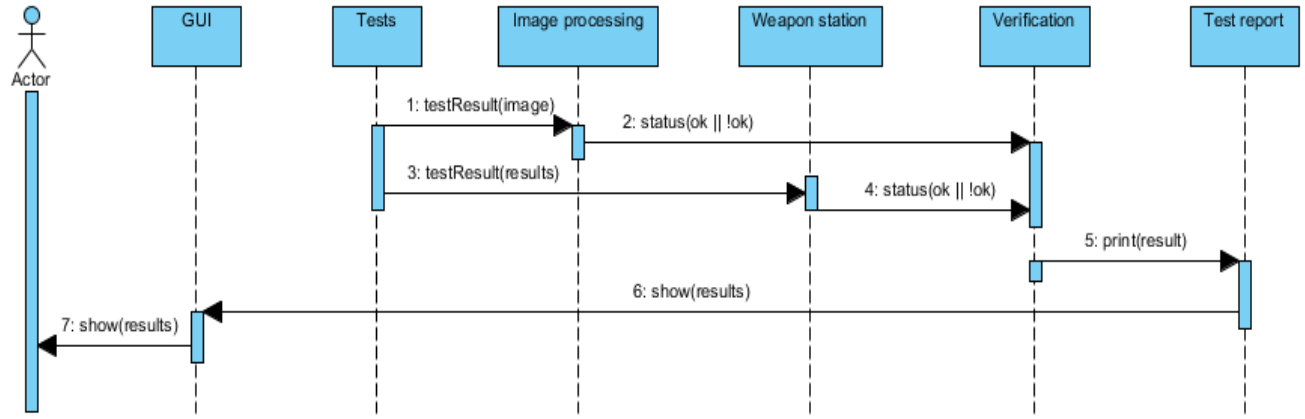
Sekvensdiagrammene viser oss hvordan en typisk testsekvens vil foregå. Det hele starter med at aktøren, her testeren, velger en test som skal starte. Deretter kan vi følge pilene ettersom de beveger seg fra klasse til klasse. Over pilene står det en tekst som viser i pseudokode hvilken metode som blir kjørt, og hvilken parameter denne inneholder. Vi har delt inn sekvensdiagrammet i to deler på grunn av at ett ville blitt veldig stort og rotete.

Første sekvensdiagram viser prosessen frem til der resultatene blir sendt til «Tests».



Figur 5: Sekvensdiagram 1

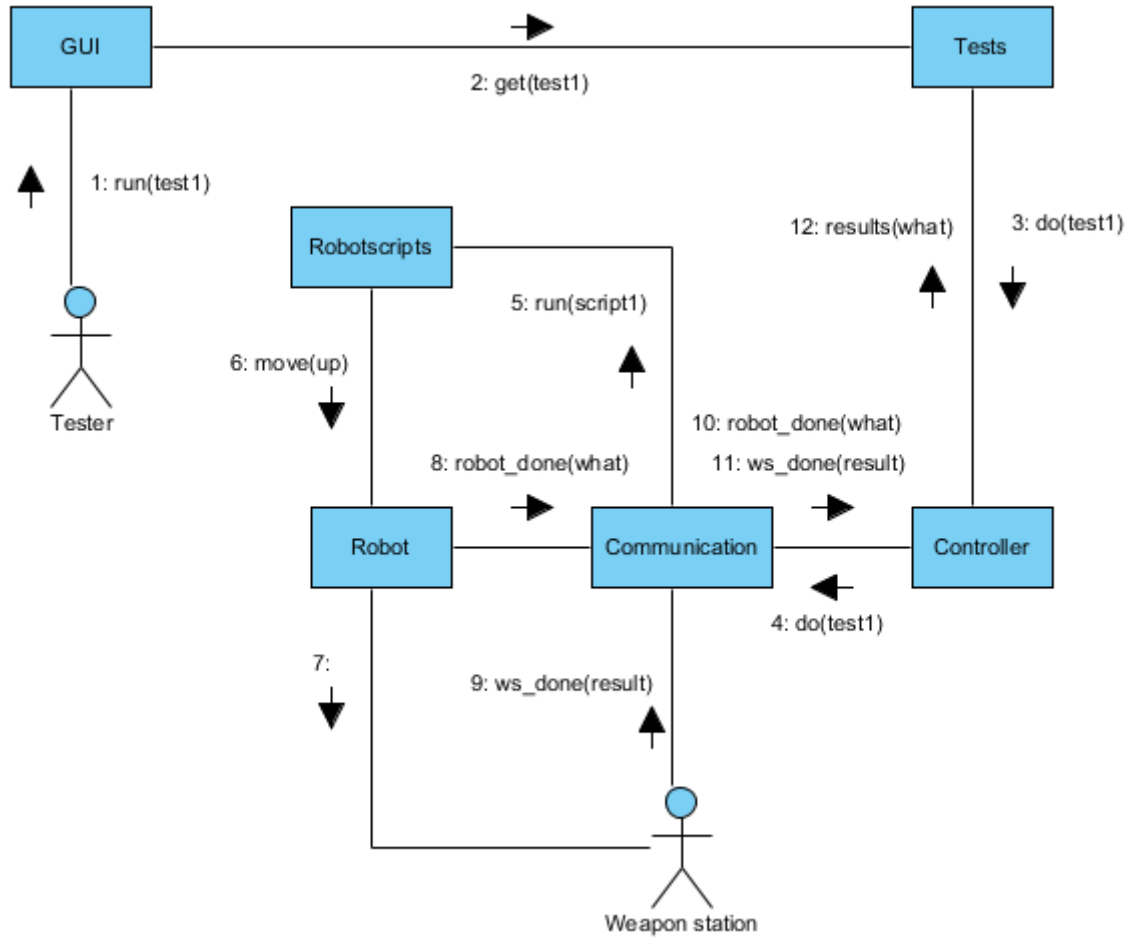
I andre del av sekvensdiagrammet fortsetter det der det første slapp. Testresultatene går gjennom verifikasjonsbiten av programmet, og vil tilslutt bli presentert for brukeren i brukergrensesnittet.



Figur 6: Sekvensdiagram 2

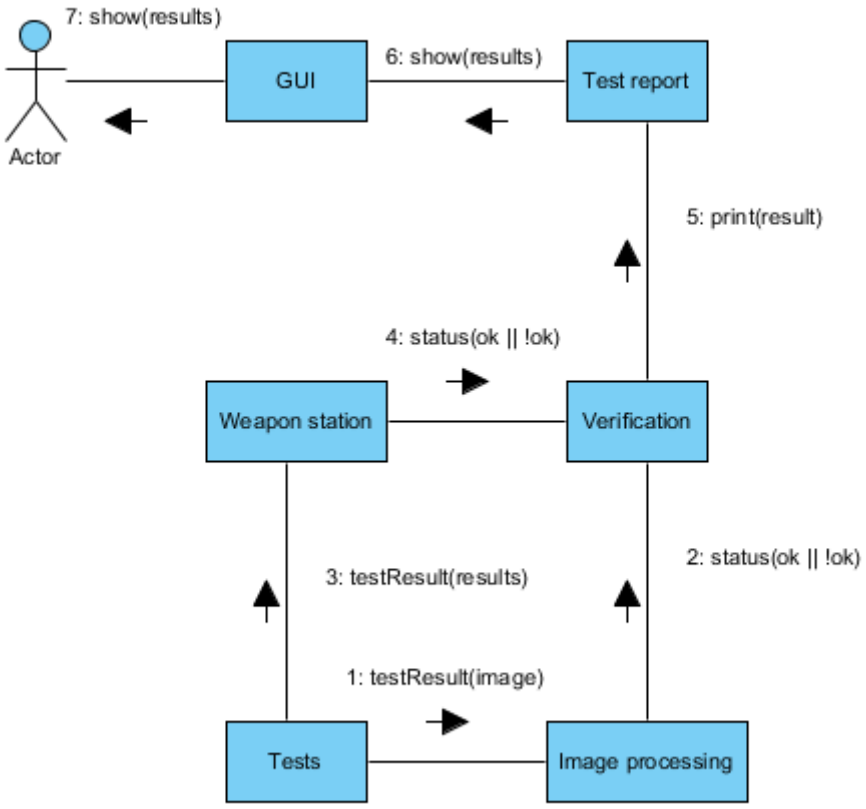
4.2.2 Kommunikasjonsdiagrammer

Kommunikasjonsdiagrammet er en forenklet versjon av sekvensdiagrammet, med samme use case som utgangspunkt. Her får vi en enklere oversikt over hvilke klasser som kommuniserer med hverandre, og hvilken vei kommunikasjonen foregår. Også her har vi to diagrammer, et til hvert sekvensdiagram som vist tidligere. Tallene viser i hvilken rekkefølge kommunikasjonen går, der «1» er starten.



Figur 7: Kommunikasjonsdiagram 1

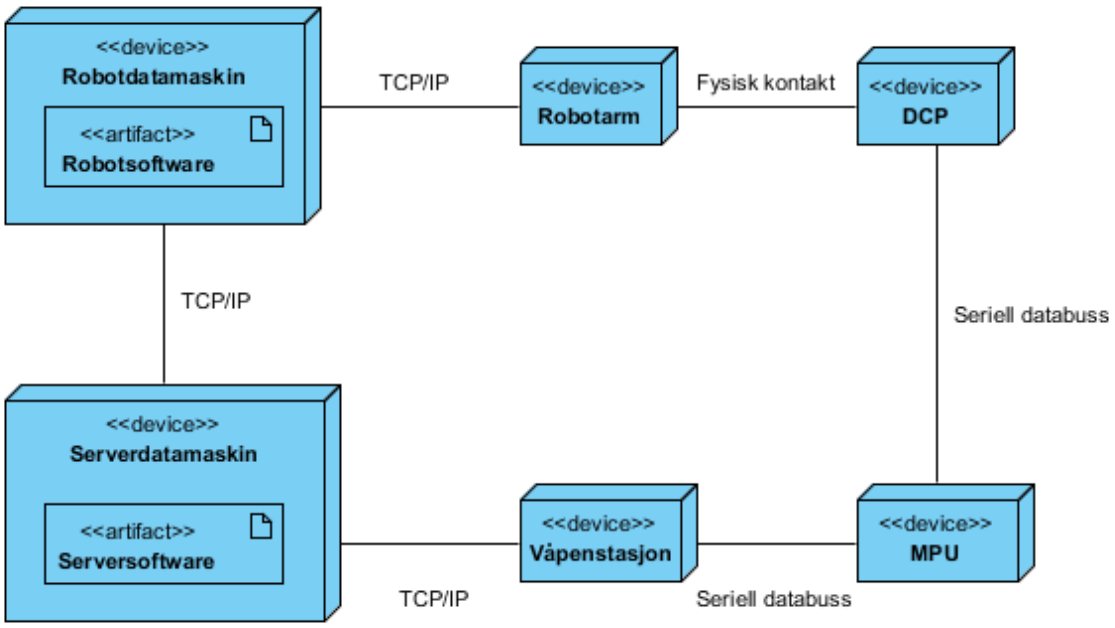
Kommunikasjonsdiagram del to viser kommunikasjonen fra der «Tests»-klassen får resultatene fra «Controller»-klassen.



Figur 8: Kommunikasjonsdiagram 2

4.3 Deployment-perspektivet

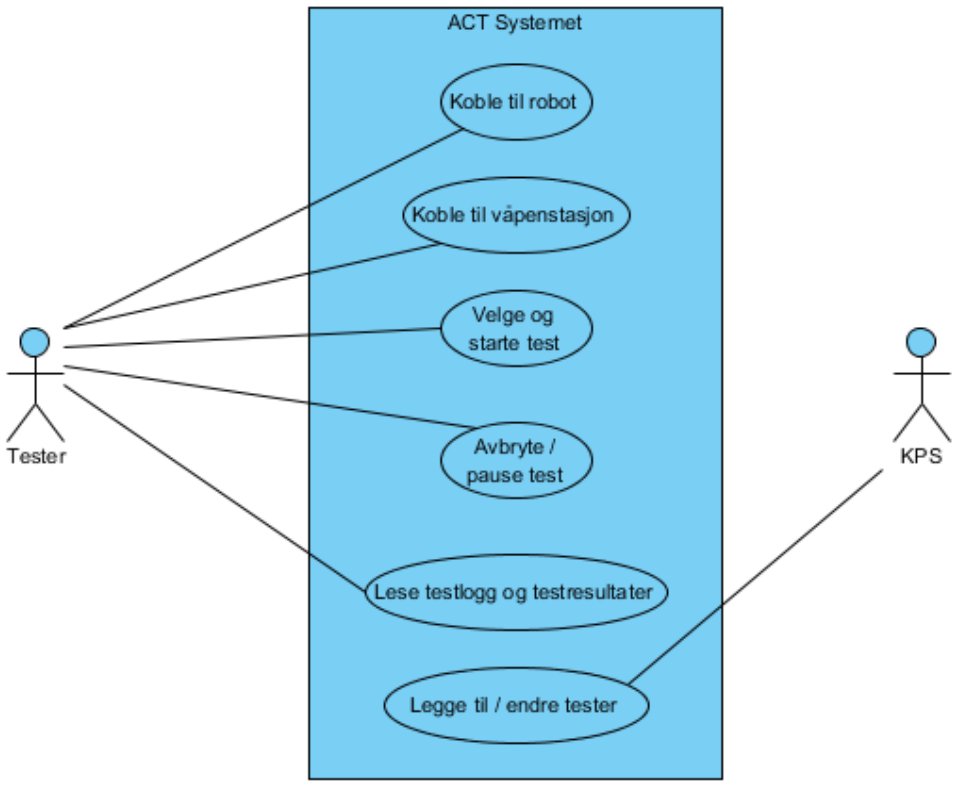
Dette brukes for å beskrive den fysiske arkitekturen til det distribuerte systemet vårt. Det viser hvordan prosesser og komponenter er fordelt på prosesseringsnoder i nettverket, og mellom nodene har vi ført opp hva slags type kommunikasjonsvei som benyttes.



Figur 9: Deployment-diagram

4.4 Implementasjonsperspektivet

I denne delen av dokumentet skal vi nå ta for oss hvordan implementasjonen og integrasjonen ble i forhold til hvordan vi hadde det planlagt. Selv om mye ble gjennomført slik det er forklart tidligere, så har det oppstått situasjoner som har ført til at noe har blitt gjort annerledes. Vi starter med å ta en titt på hvordan det nye use case diagrammet endte opp med å bli.

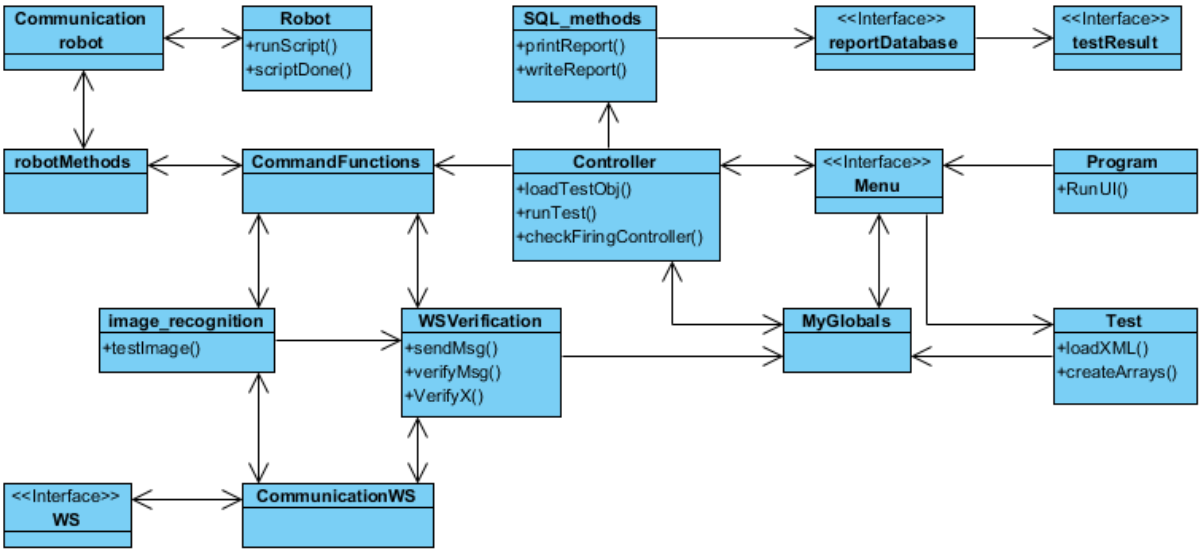


Figur 10: Oppdatert use case diagram

Her ser vi endringer i form av nye use cases og modifikasjoner på noen av de gamle. «Koble til robot» er en av de nye use casene. Vi hadde ikke med dette i det første diagrammet siden vi tenkte at dette kunne være noe som skjedde automatisk. Det vi ikke tenkte på da var tilkoblingspartene ikke alltid vil ha samme IP-adresse, og derfor må dette konfigureres før en tilkobling kan skje.

Går vi over til pakkediagrammet så kan vi si at dette stemmer godt overens med hvordan det har endt opp, med noen små endringer her og der. Selv om dette diagrammet ikke ble spesielt mye brukt under selve implementasjonen, så forklarer det likevel hvilke klasser som har nære relasjoner til hverandre. De forskjellene som er er blant annet at vi har noen fler klasser, og noen er fjernet. Det har også skjedd noen endringer i hva de forskjellige klassene gjør og hvordan de kommuniserer, og det kommer vi inn på i neste avsnitt som tar for seg klassediagrammet.

4.4.1 Klassediagrammet



Figur 11: Oppdatert klassediagram

Hvis vi sammenligner klassediagrammene før og etter implementasjon ser vi at selve strukturen er nokså lik, med noen klasser borte og noen nye innført. I det nye klassediagrammet har vi kun tatt med de viktigste kommunikasjonsveiene, da det å illustrere hver eneste vei ville blitt uoversiktlig og forvirrende. De veiene som er illustrert gir likevel et godt bilde av hvordan klassene i programmet kommuniserer.

Alle metodene som finnes i de forskjellige klassene finner en godt dokumentert i teknologidokumentet for software[2].

Vi fikk forenklet det i robotpakken ved å bruke en applikasjon som ligger på robotdatamaskinen som gjør det mulig å lage scripts og kommunisere med andre komponenter på en enkel måte. Alt man trenger å gjøre nå er å starte applikasjonen på robotdatamaskinen, så vil den være klar til å opprette en kommunikasjonsvei. Denne applikasjonen holder også på alle scripts roboten kan gjøre, og bare venter på en beskjed som bestemmer hvilken som skal starte.

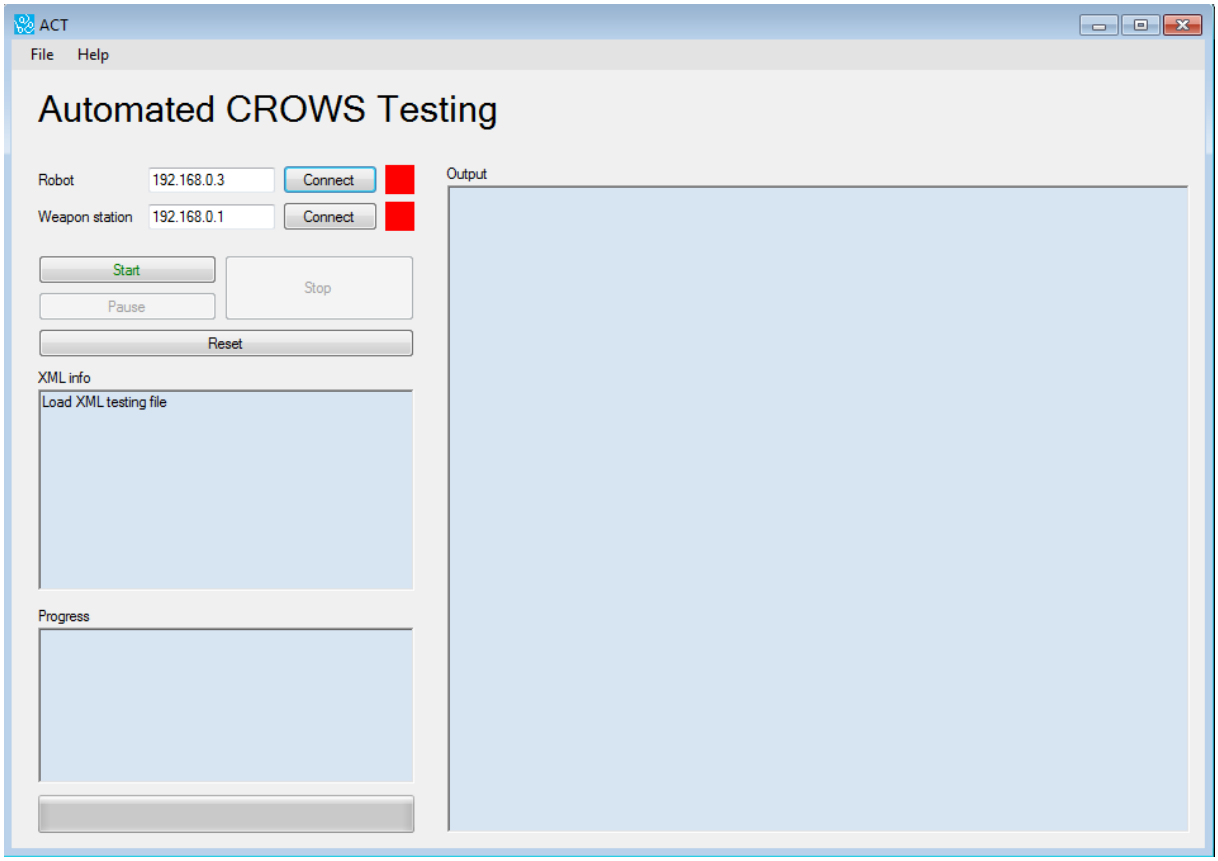
Klassen som var ment for kommunikasjon endte vi opp med å dele i to. En for våpenstasjonen og en for roboten. Dette gjorde vi for å skape bedre oversikt i programmet, og på grunn av kommunikasjonsmetodene er såpass forskjellige.

En klasse vi ikke har med i første klassediagram, som skulle vise seg å spille en sentral rolle, er klassen «MyGlobals». Denne bruker vi for å kunne jobbe med viktige variable som brukes av flere klasser rundt om i programmet. På denne måten kan vi være sikre på at alle klassene bruker samme variable med riktig verdi.

Klassen «Verification» fant vi ut at det var like greit å integrere i «WSVerification». Arbeidet de gjør er veldig likt og relatert, og verifikasjonsprosessen blir enklere ved å sammenligne resultatene direkte her.

Som nevnt tidligere så har datautvekslingen i programmet blitt nokså avansert. En kan fortsatt trekke likhetstrekk til sekvensdiagrammet, men det har bygget på seg en hel del siden da. Siden vi jobber med så mange komponenter der ting må skje i riktig rekkefølge blir beskjeder og bekreftelser sendt på kryss og tvers i programmet til en hver tid. Å illustrere dette i et sekvensdiagram vil ende opp med et veldig uoversiktlig diagram med piler på kryss og tvers, og vi har derfor valgt å la være å lage dette. For å forstå hvordan klassene og interfacene kommuniserer må en først hensikten deres, så kan en bruke det oppdaterte klassediagrammet for å skape seg et bilde på kommunikasjonen. Dette vil også variere etter hvilken testsekvens som kjøres, og er sjelden likt fra sekvens til sekvens.

4.4.2 Brukergrensesnittet



Figur 12: Oppdatert grensesnitt

Brukergrensesnittet har estetisk sett blitt bedre. Vi har lagt til flere tekstbokser som vil vise forskjellige typer output, og fjernet noen knapper som nå er tilgjengelig via verktøylinjen. «Output»-boksen viser informasjon om hva som skjer under testen. Sånn som når koblinger oppstår, når noe blir sendt eller mottatt og hva som blir sendt og mottatt. Dette er for å gi brukeren innsikt i hva programmet driver med i sanntid.

«XML info»-boksen viser informasjon om testen som er lagret i XML dokumentet. Dette kan være navnet på testen, og hvor mange sekvenser og kommandoer den består av. «Progress» viser som navnet tilsier progresjonen etter testen har blitt startet. Her kan brukeren lese hvilken sekvens som er under eksekvering, samt hva den aktuelle kommandoen inneholder. Under finner vi en «progress bar» som fylles opp ettersom testen gjennomføres.

Ved siden av «Connect»-knappene har vi nå puttet på en farget boks som indikerer om det er en kobling til våpenstasjon og robot. Grønn betyr tilkobling, mens rød betyr det motsatte.

Øverst har vi lagt til en verktøylinje, for å unngå å fylle opp hele vinduet med massevis av knapper. Fra «File»-tabben er det mulig å bla gjennom filene på datamaskinen og finne testen man vil kjøre. Der finner en også en knapp som vil lukke programmet. I «Help»-tabben har vi tre knapper. En som åpner en «Quick guide» for hvordan bruke programmet. En annen åpner «About», som forteller kort om hvem som står bak programmet og hva det er ment til. Og til slutt har vi en «Quick XML Guide» som brukeren kan åpne for enkel tilgang hvordan lage og bruke XML for ACT-systemet.

4.4.3 Konklusjon på implementasjonen

Som en konklusjon kan vi si at software designdokumentet har vært en god veileder på hva og hvordan vi implementerte systemet. Vi fikk bruk for mange av ideene som ble lagt til rette her, og har brukt dette til å legge rammeverket for noe vi kunne bygge videre på. Å forutse de forskjellene som faktisk ble til tror vi ikke ville vært mulig med tanke på den kunnskapen vi satt med da vi lagde dette dokumentet. Derfor vil vi si oss godt fornøyd med det vi planla.

5 Referanser

REFERANSE	DOKUMENTTITTEL	VERSJON
[1]	Kravspesifikasjon	2.0
[2]	Tek. dok. – Robot / XML / Software / Verifikasjon	1.0
[3]	Sluttrapport	1.0

Tabell 3: Referanser

ACT AUTOMATED CROWS TESTING

KOMPONENTDOKUMENT

AUGUST KIND SVENDSEN
EIRIK LIEN ROA
HENRIK BERGE SØRUM
STÅLE RUDIN

KONGSBERG

HØGSKOLEN I BUSKERUD OG VESTFOLD · HBV · PROFESJONSHØGSKOLEN ·

Komponentdokument

PROSJEKT	ACT - Automated CROWS Testing		
OPPDRAGSGIVER	Kongsberg Protech Systems		
UTFØRT VED	HBV - Høgskolen i Buskerud og Vestfold		
GRUPPE	August Kind Svendsen, Eirik Lien Roa, Henrik Berge Sørum og Ståle Rudin		
VERSJON	1.0		
ANTALL SIDER	10		
DOKUMENTHISTORIE	VERSJON	UTGITT	BESKRIVELSE
	1.0	26.05.14	Første utgivelse

INNHOLDSFORTEGNELSE

LISTE OVER TABELLER

LISTE OVER FIGURER

1 Om dokumentet

1.1 Dokumenthistorie

VERSJON	DATO	ENDRING	SIGNATUR
0.1	16.05.14	• Dokumentet ble opprettet	AKS
1.0	21.05.14	• Dokument fullført	AKS

Tabell 1: Dokumenthistorie

1.2 Definisjoner og forkortelser

Liste som beskriver ord og uttrykk som kan være brukt i dette dokumentet:

UTTRYKK	FORKLARING
HBV	Høgskolen i Buskerud og Vestfold
KPS	Kongsberg Protech Systems (oppdragsgiver)
ACT	Automated CROWS Testing
Industrirobot	Universal Robots 5
CROWS	Common Remotely Operated Weapon Station
VS	Våpenstasjon
DCP	Display Control Panel
MPU	Main Processing Unit
Konfigurasjon	Forskjellige CROWS oppsett
Firing controller	Eksternt instrument som måler fyringsspenningen
CG	Control Grip
Mainframe	Hovedrammen som gjør opp våpenstasjonen

Tabell 2: Definisjoner og forkortelser

2 Innledning

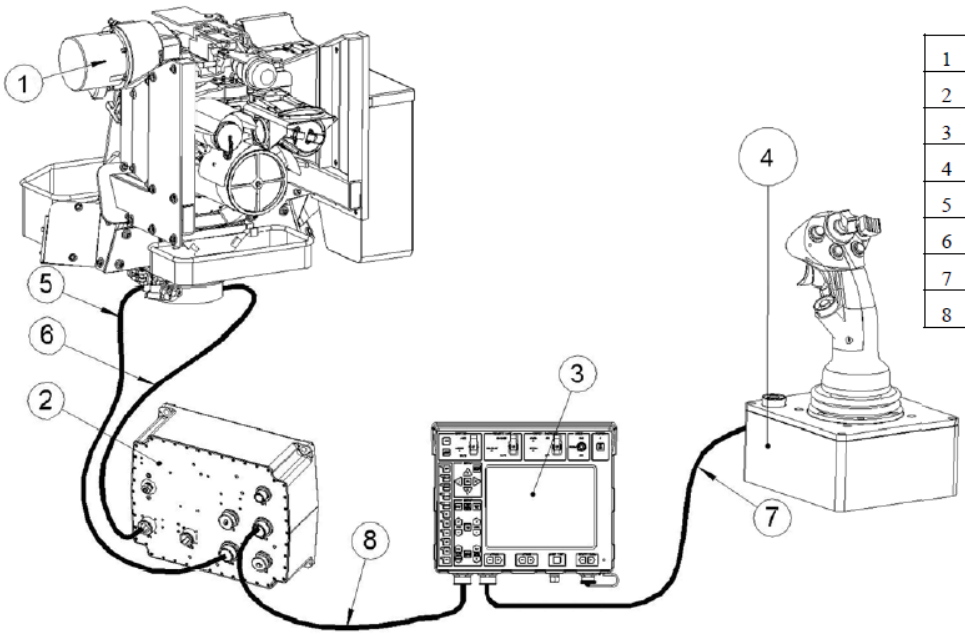
Hensikten med dette dokumentet er å beskrive de hardware-komponentene som er nødvendig for at ACT systemet skal fungere. Dette innebærer både deler som testen blir utført på og det utstyret som trengs for å gjennomføre testene. Å dokumentere dette blir viktig hvis en bruker ønsker å sette opp et testsystem, så er det kritisk at komponentene stemmer og at det er satt opp riktig i forhold til hverandre. Dokumentet dekker også litt teknisk informasjon rundt delene, dersom dette skulle bli relevant for brukeren.

3 Sammendrag

Vi har arbeidet med flere viktige komponenter under dette prosjektet. Hvilke komponenter vi har brukt, hvordan de kommuniserer og relativ plassering er det viktig at er satt riktig. CROWS konfigurasjonen består av våpenstasjonen M-153A1, en Control Grip, DCP, MPU og en Power Supply. Roboten består av robotarmen, en datamaskin som styrer denne, og et berøringsbrett for å kontrollere og kode armen. For at delene som brukes under testing står på riktig plass, bruker vi et bord der ting er montert fast. Det er viktig at alt står riktig, eller så vil ikke roboten klare å gjøre de fysiske oppgavene. Alt som utgjør systemet trenger å kommunisere sammen, og for dette har vi et lokalt TCP/IP nettverk, der hver enhet har satt sin egen statiske IP.

4 CROWS konfigurasjonen

Under prosjektet har vi utviklet produktet vårt rundt en bestemt konfigurasjon. En våpenstasjon består nemlig av flere deler, der versjon av programvare og det fysiske utseende kan variere. For at testoppsettet skal bli riktig i forhold til det produktet vi har lagd, må en gjøre en vurdering på om komponentene vil passe til systemet. Under ser vi en illustrasjon som viser de viktigste enhetene i et CROWS oppsett, og hvilke kabler som er brukt for å koble det sammen.



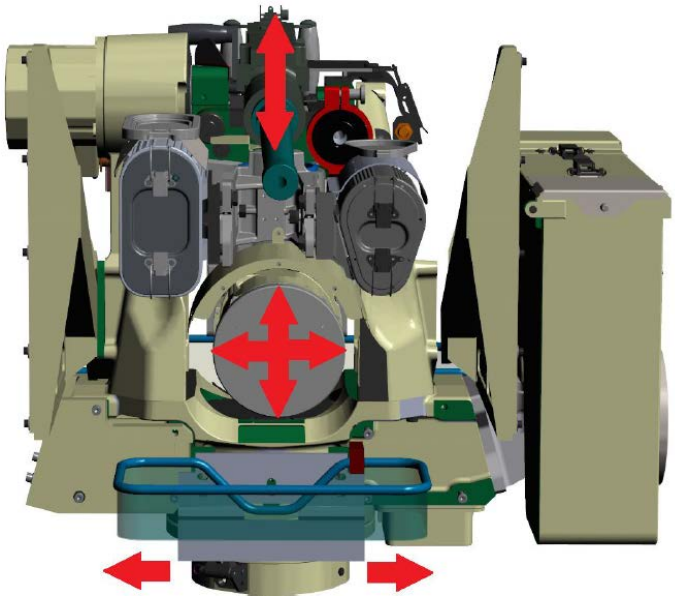
1	Weapon Station (WS)
2	Main Processing Unit (MPU)
3	Display & Control Panel (DCP)
4	Control Grip (CG)
5	W2 Cable (MPU – WS)
6	W3 Cable (MPU – WS)
7	W1 Cable (DCP – CG)
8	W504 Cable (MPU – DCP)

Figur 1: CROWS konfigurasjon

4.1 Våpenstasjonen - M153A1

Våpenstasjonen er den mest sentrale komponenten i testoppsettet. Hele testen baserer seg på å finne ut om denne komponenten fungerer som den skal. Dette er plattformen som monteres på toppen av et kjøretøy, som gjør det mulig å observere og fyre uten å måtte utsette seg for åpen ild. Våpenstasjonen vi har tatt for oss er en M153A1, ment for å kunne fjernstyre små og mellomstore kaliber, granatkastere, og ikke-dødelige verktøy som lyskastere eller lasermålere.

Denne plattformen består av flere deler som kan bevege seg uavhengig av hverandre, illustrert på bildet til høyre. Våpenet kan bevege seg uavhengig vertikalt, siktehodet både vertikalt og horisontalt, og mainframe horisontalt. Dette gjør det mulig for stasjonen å utføre manøvre som ikke ville vært mulig ellers. Et eksempel er muligheten til å holde målet midt i bildet uten at retningen våpenet blir påvirket.



Figur 2: Våpenstasjonens bevegelsesretninger

Under testing er det viktig at våpenstasjonen har et stødig underlag den er godt festet til. Det er et tungt redskap som kan akselerere veldig raskt når den roterer, og om uheldig vil den på dårlig underlag kan havne i ubalanse.

Våpenstasjonen trenger en egen programvare for å fungere. Dette kjører direkte på stasjonen, men hvilken versjon det er kan variere. Vi har utviklet ACT systemet til å fungere på versjon 3.11.0.

4.2 Display Control Panel

Display Control Panel, heretter DCP, er som navnet tilsier et kontrollpanel som hører til våpenstasjonen. Her finner vi de knappene og spakene som kontrollerer og navigerer de forskjellige innstillingene og funksjonene stasjonen har. Dette kontrollpanelet kommer i flere varianter, der posisjonen på knappene og antall knapper vil være forskjellig. Roboten klarer ikke å tilpasse seg kontrollpanelet automatisk, så alle bevegelser må kodes på forhånd.

Figur 3: DCP

Vi har valgt å ta for kontrollpanelet kalt DCP. I fremtiden vil det være ønskelig å utvide kompatibiliteten slik at andre kontrollpanel som «FCU2», «FCU3» og «WSCP» også vil være støttet. Dette er bare et spørsmål om scripting av bevegelsene til roboten, og en mulighet til å velge type kontrollpanel i ACT brukergrensesnittet.

4.3 Main Processing Unit

Denne komponenten har som funksjon å prosessere signalene mellom DCP-en og våpenstasjonen. ACT systemet er ikke avhengig av en spesiell modell eller versjon av MPU-en, så lenge resten av våpenstasjonen fungerer med den enheten som er installert så skal ikke dette føre til komplikasjoner.

4.4 Control Grip

Control Gripen er det viktigste verktøyet når en skal kontrollere våpenstasjonen, og kan minne om den klassiske joysticken en kjenner fra andre steder. Det er ved bruk av denne en blant annet kan fyre av våpenet montert, og bevege VS for å bestemme hva som vises på DCP-skjermen.

Figur 4: Control Grip

Som ved DCP-en så er det flere modeller av denne enheten som fungerer som styringsverktøy, der utformingen varierer stort fra modell til modell. I starten av prosjektet fikk vi utdelt en Control Grip, men det finnes alternativer som «Thumb Stick», «Dual Grip», og «SSAM». Disse har ikke vi lagt fokus på når vi har utviklet og er derfor ikke støttet for øyeblikket, men også her er det bare et spørsmål om å lage nye scripts som tilpasser roboten modellen som brukes.

4.5 Power supply

Power supply, eller strømforsyningen, er apparatet som tilfører strøm til CROWS systemet via en vanlig stikkontakt. ACT systemet har ingen direkte kommunikasjon med dette apparatet, men i løpet av enkelte tester kan det være at testeren må justere antall volt ved hjelp av skruknappen montert på frontpanelet. Ikke alle strømforsyningene har denne funksjonen, men den finnes på «Delta Elektronika SM 35-45» som vi har brukt under utviklingen.

Figur 5: Delta Elektronika SM 35-45

5 Robot

For å utføre de fysiske oppgavene som testene krever bruker vi en industrirobot. Mer presist er dette en UR5 fra Universal Robots. Roboten kan minne om en litt lang arm med flere ledd, og er tydelig preget av produsentens visjon om å lage en lettbrukt og elegant, men likevel robust og funksjonsrik robot. Ytterst på armen har en muligheten til å feste på et eget verktøy. I vårt tilfelle festet vi på gummibelagte metalpinner som gjør det mulig å trykke på knapper og spaker. Robotsystemet består av robotarm med verktøy, en datamaskin som robotprogramvaren ligger på, og et berøringsbrett. Under er delene nærmere beskrevet, men om en ønsker å gå i detalj så anbefales det å lese teknologidokumentet for roboten[2].

5.1 Robotarm

Armen gjør det mulig for oss å utføre de fysiske oppgavene som testen krever. Den består av hele 6 ledd, som gir muligheten til å gjøre veldig detaljerte bevegelser med en stor rekkevidde. Det som begrenser hva denne armen kan gjøre er verktøyet som er festet på enden. Verktøyet vi bruker er nokså enkelt med bare faste deler. Det består av en brikke som monteres på enden av robotarmen. På denne er det flere festehull, og her har vi montert noen metalpinner (omtrent 10 cm lange og 0.5 cm i diameter) som fungerer som kontaktpunkt mellom robot og VS. Underveis i prosjektet måtte vi bytte verktøyet på grunn av det viste seg at metallpinnene ikke var festet godt nok. Vi forbedret designet ved å endre formen på fingrene, plassere de annerledes på brikka, og skru de ekstra godt fast ved hjelp av strammeskiver.

Figur 6: Robotarm

5.2 Robotdatamaskin med berøringsbrett

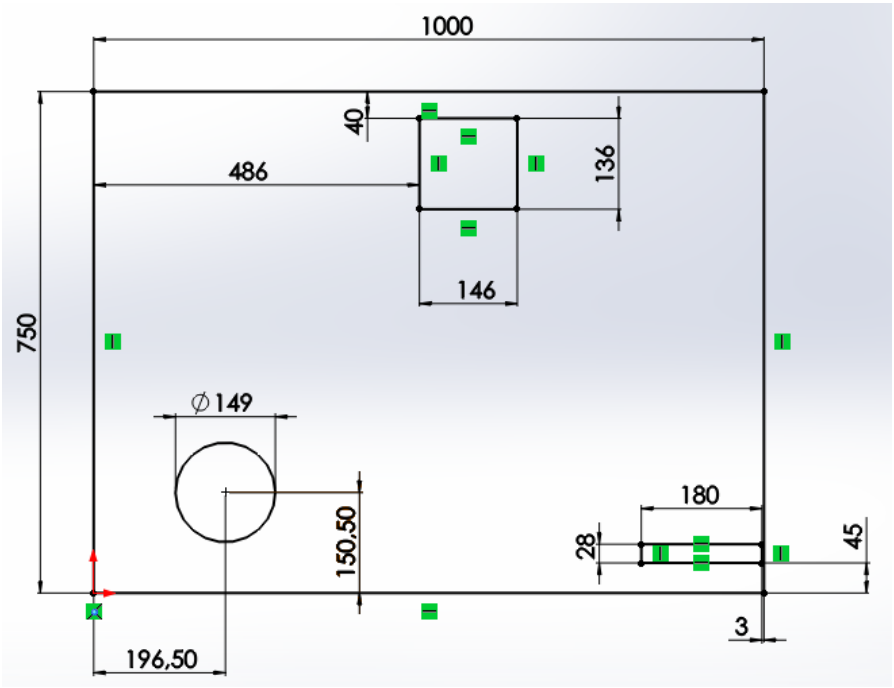
Robotarmen alene vil ikke være i stand til å utføre noe som helst uten en datamaskin som kan gi instrukser. Denne datamaskinen fulgte med robotsystemet og er spesiallagd for å kontrollere robotarmen. Den benytter seg av Linux som operativsystem og holder på egen programvare som gjør det enkelt å programmere bevegelser til roboten.

For å benytte seg av programvaren for robotbevegelser følger det med et berøringsbrett som gjør dette enda enklere. Med denne kan en kjøre, endre og lage bevegelsesmønstre, og bestemme hvilke eventuelle input som skal fungere som triggere og output om en ønsker det. Dette har vært viktige funksjoner for at ACT systemet skal kunne utføre oppgaver i riktig rekkefølge. På brettet finner vi en stor og rød knapp som vil avbryte bevegelsessekvensen dersom det skal bli nødvendig. På baksiden finner vi en sort knapp som gjør det mulig å bevege armen fritt, dersom den havner i en posisjon den ikke var ment til.

6 Mobil plattform

Når en utfører tester med ACT systemet er det helt nødvendig at utstyret er plassert riktig i forhold til hverandre. Hvis ikke vil ikke roboten treffe med bevegelsene og testen blir ugyldig. Arbeidsgiver hadde heldigvis gjort dette arbeidet på forhånd, og vi hadde allerede fra prosjektstart en mobil plattform som robotarm, CG og DCP er montert til. Plattformen er i hovedsak et bord med to etasjer satt på hjul. I den ene enden i andre etasje er det montert et feste for DCP-en, der en bare skrur til noen skruer etter plassering. I motsatt ende finner vi festet for robotarmen, og på den langsiden er CG montert. I første etasje står robotdatamaskinen, MPU og strømforsyningen fritt.

Vi sitter ikke på de nøyaktige målene på hvordan utstyret er montert på bordet, men har gjort et forsøk på å finne de. Resultatet er illustrert under. Sirkelen nederst til venstre representerer robotarmen, firkanten øverst i midten er CG, og til høyre har vi første ramme for DCP-en.



Figur 7: Robotbord

7 Nettverket

Nettverket er kommunikasjonsveien mellom komponentene i ACT systemet, og uten dette vil det ikke være noen dataflyt. Det er lokalt TCP/IP nettverk, der hver komponent som er koblet opp har sin egen statiske IP-adresse. For datamaskinene gjør en dette via kontrollpanelet i Windows, mens på våpenstasjonen gjøres det via scriptet som kjører automatisk når våpenstasjonen starter. Robotdatamaskinen har en funksjon som gjør det mulig å velge hvilken IP den skal prøve å koble seg til, noe den repeterer helt til den får tilkobling. For å koble alt sammen har vi brukt to ethernet-switcher.

8 Referanser

REFERANSE	DOKUMENTTITTEL	VERSJON
[1]	Kravspesifikasjon	2.0
[2]	Teknologidokument - Robot	1.0

Tabell 3: Referanser

ACT AUTOMATED CROWS TESTING

TEK. DOK - ROBOT

AUGUST KIND SVENDSEN
EIRIK LIEN ROA
HENRIK BERGE SØRUM
STÅLE RUDIN

KONGSBERG

HØGSKOLEN I BUSKERUD OG VESTFOLD · HBV · PROFESJONSHØGSKOLEN ·

Teknologidokument - Robot			
PROSJEKT	ACT - Automated CROWS Testing		
OPPDRAGSGIVER	Kongsberg Protech Systems		
UTFØRT VED	HBV - Høgskolen i Buskerud og Vestfold		
GRUPPE	August Kind Svendsen, Eirik Lien Roa, Henrik Berge Sørum og Ståle Rudin		
VERSJON	1.0		
ANTALL SIDER	40		
DOKUMENTHISTORIE	VERSJON	UTGITT	BESKRIVELSE
	1.0	26.05.14	Første utgivelse

INNHOLDSFORTEGNELSE

LISTE OVER TABELLER

LISTE OVER FIGURER

1 Om dokumentet

1.1 Dokumenthistorie

VERSJON	DATO	ENDRING	SIGNATUR
0.1	04.05.14	• Dokumentet ble opprettet	HBS
1.0	26.05.14	• Første utgivelse	HBS

Tabell 1: Dokumenthistorie

1.2 Definisjoner og forkortelser

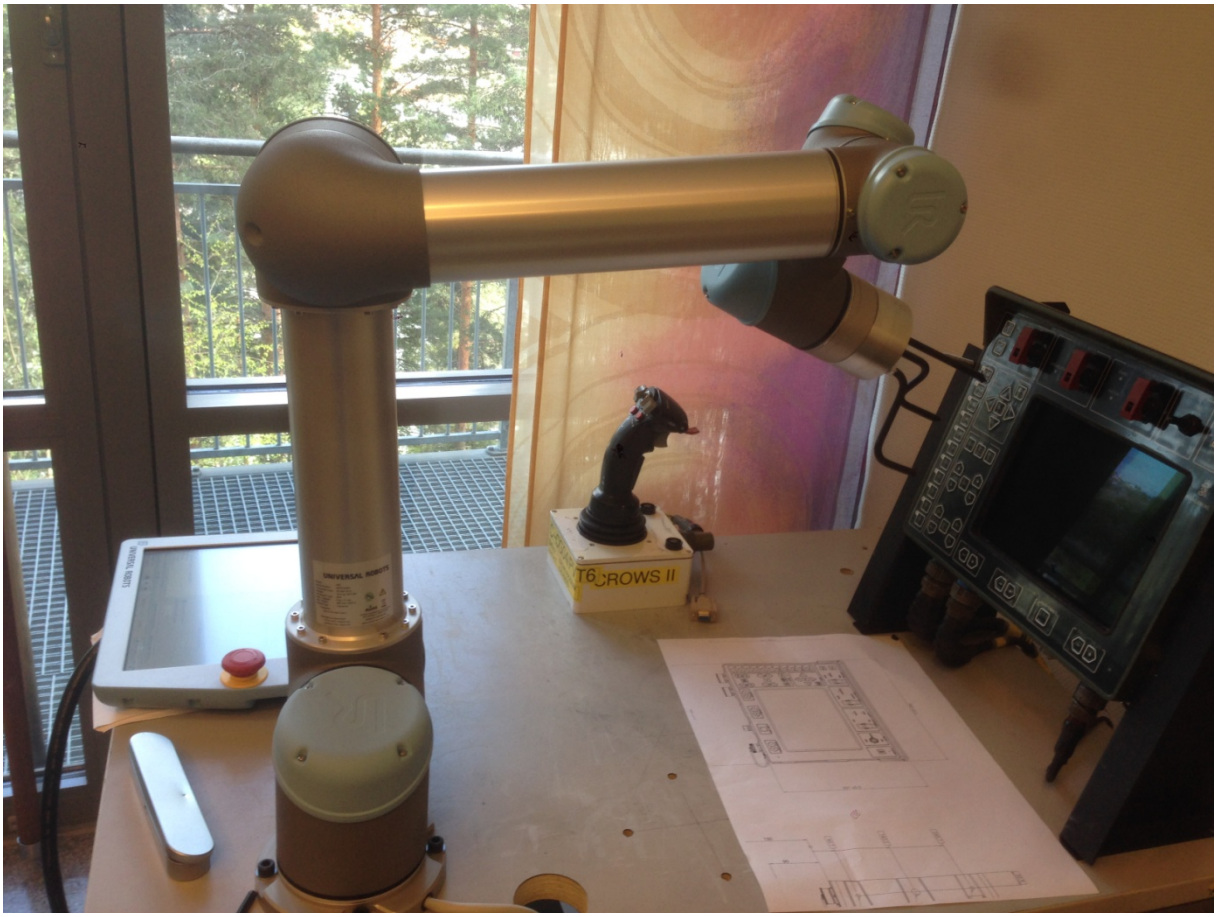
Liste som beskriver ord og uttrykk som kan være brukt i dette dokumentet:

UTRYKK	FORKLARING
HBV	Høgskolen i Buskerud og Vestfold
KPS	Kongsberg Protech Systems (oppdragsgiver)
ACT	Automated CROWS Testing
UR5	Universal Robots 5
CROWS	Common Remotely Operated Weapon Station
RWS	Remote Weapon Station
DCP	Display Control Panel
MPU	Main Processing Unit
Konfigurasjon	Forskjellige CROWS oppsett
Firing controller	Eksternt instrument som måler fyringsspenningen
CG	Control grip

Tabell 2: Definisjoner og forkortelser

2 Innledning

Bacheloroppgaven går ut på å automatisere funksjonstester som blir utført på CROWS våpenstasjoner. En sentral del av automatiseringen av disse testene er en robotarm som skal gjøre alle "manuelle" oppgaver automatisk. Dette innebærer å trykke på knapper og brytere på DCP. I tillegg skal den flytte og avfyre med Control Grippen.
KPS har kjøpt inn en robotarm fra Universal Robots(UR5). Dette er en robotarm som har 6 ledd som kan rotere 360°. I tillegg hadde KPS laget et verktøy som roboten bruker til å trykke på knapper og bevege på CG. Vi har brukt dokumentasjonen til roboten for å sette oss inn i hvordan roboten fungerer. Software manual[2], user manual[3] og scriptmanual[4].



Figur 1: Robot

3 Sammendrag

Dette dokumentet er all dokumentasjon til roboten. Dokumentet inneholder de tekniske spesifikasjonene, planene, verktøy, URScript programmering, ACT_Robotscript og konklusjon.

De tekniske spesifikasjonene er hentet fra dokumentet technical spesifications UR5 som er laget av Universal robots, denne tabellen beskriver alle de tekniske spesifikasjonene til roboten.

Planene er en beskrivelse av hva et plan er og hva dette brukes til. Planene lages slik at det er lettere å navigere roboten. DCP planet ligger på DCP slik at Z koordinatet alltid er null på dette planet.

Verktøyet var ferdig laget når vi startet på bachelorprosjektet. Verktøyet er bygget opp av to deler. En sylinder og 3 stenger som er bøyd. Sylinderen er verktøyfestet til stengene, denne festes til roboten med fire skruer og har flere gjenget hull som stengene er montert i.

URScript programmering er et kapitel som forklarer litt om hvordan man programmerer roboten og hvilke funksjoner som vi bruker i våres program. MoveJ funksjonen brukes til bevegelsene til roboten, denne funksjonen er bygget opp av vendepunkter som roboten beveger seg mellom. Force funksjonen brukes for å trykke på knapper, denne funksjonen lar roboten påføre en kraft til omgivelsene i en bestemt akse. Wait funksjonen kan bygges opp på ulike måter, når man lager en slik funksjon kan man bestemme hvordan denne funksjonen skal vente. I vårt program brukes denne til å vente en bestemt tid før programmet skal fortsette, eller så skal programmet vente på en beskjed fra PCen.

ACT_Robotscript er robotprogrammet. Dette kapitelet inneholder en beskrivelse av hvordan programmet er bygget opp og hvordan de ulike delene fungerer, i tillegg så er det et underkapitel som har bilde av alle vendepunkter og koordinatene til verktøyet i de forskjellige vendepunktene. Programmet er bygget opp av tå tråder som kjører parallelt og subprogrammer som utfører de fysiske bevegelsene. Subprogrammene blir kalt fra hovedprogrammet når en beskjed er mottat fra PCen.

Til slutt er det en konklusjon som forklarer hvorfor vi har gjort som vi har gjort når vi har programmert roboten og hvilke beslutninger vi har tatt underveis i prosjektet. Det er også beskrevet hva som ikke fungerer bra nok som burde endres i videreutvikling av systemet.

4 Tekniske spesifikasjoner

6 – Axis robot arm with a working radius of 850 mm / 33.5 in	
Weight:	18.4 kg / 40.6 lbs
Payload:	5 kg / 11 lbs
Reach:	850 mm / 33.5 in
Joint ranges:	+/- 360° on all joints
Speed:	Joint: Max 180°/sec. Tool: Approx. 1 m/sec. / Approx 39.4 in/sec.
Repeatability:	+/- 0.1 mm / +/- 0.0039 in (4mil)
Footprint:	Ø149 mm / 5.9 in
Degrees of freedom:	6 rotating joints
Control box size (WxHxD):	475 mm x 423 mm c 268 mm / 18.7 x 16.7 x 10.6 in
I/O ports:	10 digital in, 10 digital out, 4 analogue in, 2 analogue out
I/O power supply:	24 V 1200 mA in control box and 12 V/24 V 600 mA in tool
Communication:	TCP/IP 100Mbit: IEEE 802.3u, 100BASE-TX Etherned socket & Modbus TCP
Programming:	Polyscope graphical user interface on 12 inch touchscreen with mounting
Noise:	Comparatively noiseless
IP classification:	IP54
Power consumption:	Apporx: 200 watts using a typical program
Collaboration operation:	Tested in accordance with sections 5.10.1 and 5.10.5 of EN ISO 10218-1:2006
Materials:	Aluminium, ABS plastic
Temperature:	The robot can work in a temperature range of 0-50°C
Power supply:	100-240 VAC, 50-60 Hz
Calculated Operating Life:	35,000 Hours

Tabell 3: Tekniske spesifikasjoner til UR5[1]

5 Planene(Planes)

Planet er en flate som brukeren har satt. Dette planet blir satt av tre punkter. Når vi lagde planene brukte vi vektøyet til roboten og satt den på tre punkter på DCPen og CGen. Når punktene er satt lager roboten automatisk et plan rundt disse tre punktene.

Planet brukes til å navigere roboten i forhold til DCPen og CGen.
Når roboten skal trykke på en knapp eller flippe en bryter så må den vite hvor DCPen er. Når et plan er laget er det lett å navigere verktøyet til roboten i forhold til det gitte planet.

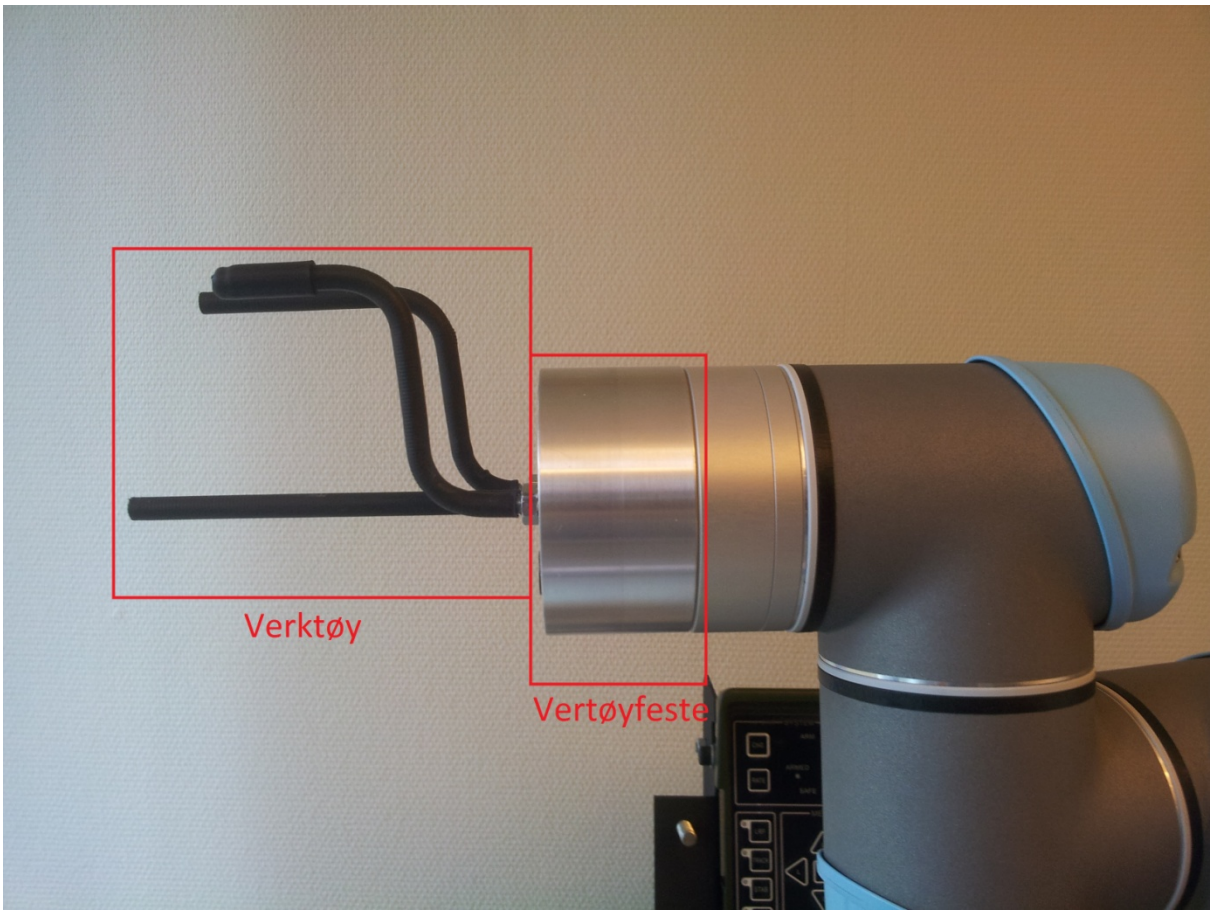
Figur 2: DCP planets akser

Figur 3: CG planets akser

Vi kom borti ett problem med å lage et plan. Dette planet er statisk og må endres manuelt. På grunn av dette er det viktig å oppdatere planet hver gang det er en ny DCP som roboten skal arbeide på eller det har blitt gjort endringer som kan ha foresaket at DCPen ikke står nøyaktig på samme sted som sist. DCPen har noe ”slingring” når den skal monteres til brakettene og derfor kan det bli noe forskjell når en ny DCP settes inn eller den gamle blir løsnet fra braketten.
Vi så noe på å lage et program som gjør at roboten lager planet selv ved hjelp av andre funksjoner, men vi valgte og ikke prøve dette på grunn av flere andre ting i systemet som trengte prioritet.
Hvis det blir gjort endringer som gjør at DCP ikke står på samme plass, må punktene i planet oppdateres. Dette gjøres ved å gå på "features" og velg planet "DCP_plan" med robotens GUI. Her må man oppdatere punktene til planet slik at de stemmer med det nye planet. Første punktet(bottom_left) settes ved at verktøyet til roboten ligger inntil DCP på den nederste knappen til venstre. Når verktøyet er i posisjon så oppdaterer man dette punktet. Det samme gjøres for de andre punktene (Top_left) og (Top_right). Verktøyet settes inntil DCP i midten av den øverste knappen til venstre og den øverste knappen til høyre. Når verktøyet er i posisjon så må man oppdatere punktet. Man oppdaterer punktene ved å trykke "change this point" og OK.

6 Verktøy

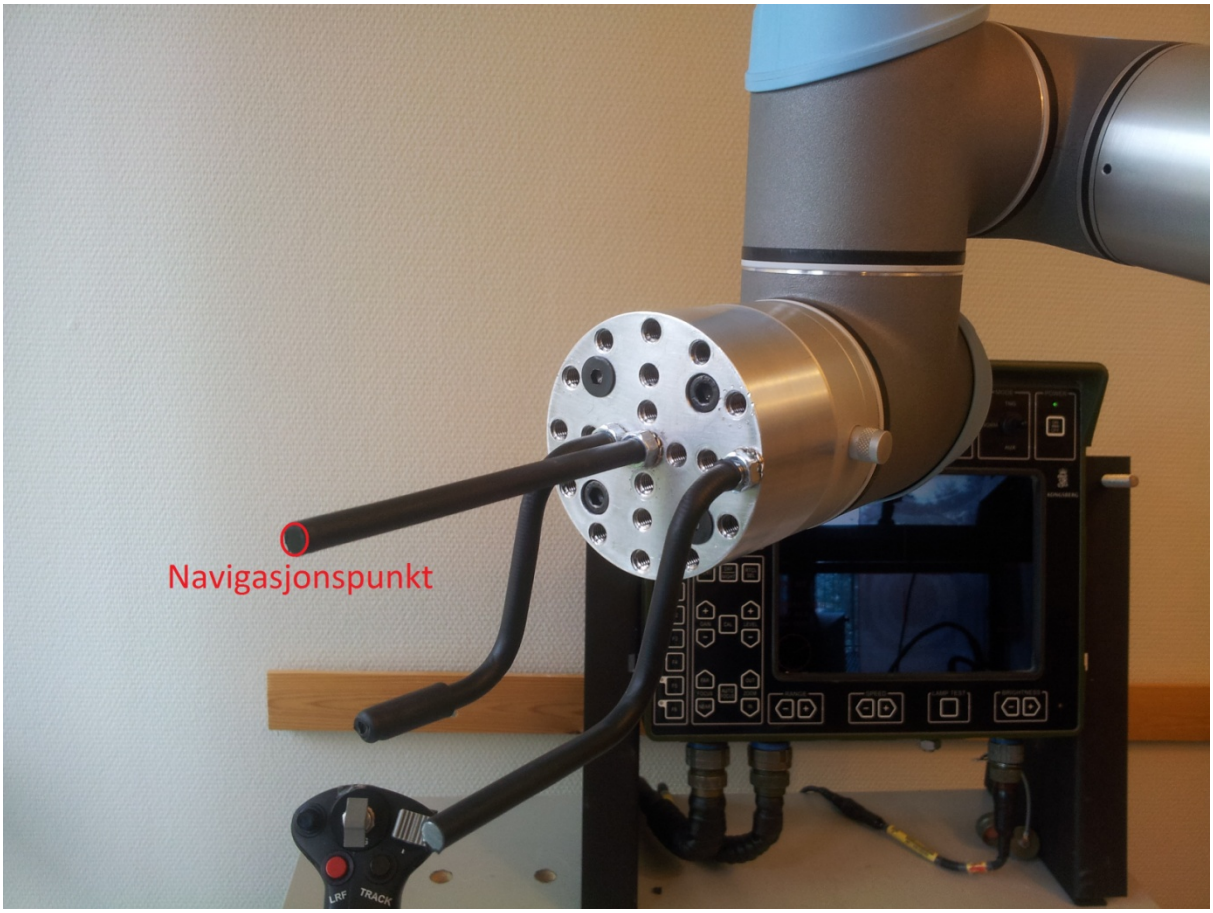
For at roboten skal kunne utføre en fysisk jobb på DCP og CG så må det monteres et verktøy på roboten. Dette verktøyet er det som trykker på knapper, flipper brytere og flytter på Control grip.



Figur 4: Verktøy og verktøyfeste

Verktøyet er bygget opp av to deler. Den første delen er en sylinder som er laget av K-Tech. Sylinderen er den som monteres på roboten og gjør det mulig å montere forskjellig verktøy til roboten. Sylinderen monteres fast til roboten med fire bolter, og har flere ferdig gjenget hull som det kan monteres verktøy i. Den andre delen er de tre stengene som brukes for å gjøre en jobb på DCP eller CG. De tre stengene er ulike, en er helt rett og brukes for å trykke på DCP og flippe brytere av eller på. De to andre har en to vinkler på ca. 90°. Dette er for at de skal kunne komme på den andre siden av CG i forhold til den rette stangen. De bøyde stengene brukes til å trykke på palm og trigger knappene på CG, og en av dem brukes også til å flippe opp sikkerhetsbeskyttelsene på DCPen.

Når det monteres på et verktøy til roboten så kan man sette inn hvor verktøyet sitter og hvor langt det stikker ut fra roboten. Dette gjøres for at man skal kunne navigere roboten med hensyn på verktøyet og ikke tuppen av roboten. I våres tilfelle stikker verktøyet ut ”11.5cm” så når man har satt et punkt som er 11.5cm utenfor roboten så navigerer man etter dette punktet. Det gjør det lettere å bevege roboten til riktig koordinater.



Figur 5: Navigasjonspunkt

7 URScript programmering

Språket som blir brukt til å programmere roboten er noe som heter URScript Programming Language. Det er tre måter å kontrollere roboten: The Graphical User-Interface Level, the Script Level and the C-APILevel. URScript er robotens programmerings språk som blir brukt **til** å styre roboten på Script Level. Vi bruker forskjellige scripts for å kontrollere bevegelsene til roboten. Når roboten skal programmeres brukes Polyscope(det grafiske brukergrensesnittet). Dette er en berøringsskjerm som er koblet til robotens datamaskin.

7.1 MoveJ

MoveJ gjør bevegelser som er kalkulert i leddet på roboten. Hvert ledd Kontrolleres for og nå ønsket endelokasjon samtidig. Det fører til en kurvet bane for verktøyet. De delte parametrene som gjelder denne bevegelsestypen, er maksimal leddhastighet og leddakselerasjon som brukes for bevegelseskalkulasjonene, spesifisert i henholdsvis deg/s og deg/s2. MoveJ er laget slik at roboten skal bevege seg raskt mellom vendepunkter. Problemet med MoveJ er at den ikke tar hensyn til

verktøyets bane mellom disse vendepunktene. På grunn av dette måtte vi lage noen ekstra vendepunkter til roboten, slik at det er sikkert at roboten ikke krasjer i DCP eller CG når den beveger seg.
En MoveJ funksjon er når roboten utfører en bevegelse som er bygget opp av et eller flere vendepunkter. Vi bruker en MoveJ funksjon for alle oppgaver roboten skal utføre. Et eksempel på en slik funksjon er når roboten beveger seg fra start posisjon til DCP, trykker på en knapp, og returnerer til startposisjon igjen. Det kan også være andre funksjoner inne i en MoveJ funksjon. I noen suprogrammer bruker vi wait eller force funksjoner inne i MoveJ funksjonen. Dette brukes for at roboten skal holde inne en knapp i en lengre tid(wait), eller at roboten skal klare å trykke på en knapp(force).

7.2 Force

Kraftmodus muliggjør samkjøring og bruk av kraft i en valgbar akse i robotens arbeidsområde. Det er Forksjellige kraftmodustyper. Vi har valgt å bruke kraftmodustypen Ramme. Rammetypen muliggjør mer avansert bruk. Her kan kompatibilitet og kraft i alle seks frihetsgrader velges uavhengig av hverandre. Vi brukte denne funksjonen når roboten har verktøyet liggende på en knapp i planet. Den vil da påføre en bestemt kraft i Z retning. Vi prøvde oss litt fram og fant ut at 25N var nok kraft til å kunne trykke på knapper med god margin. En kraft funksjon brukes for å unngå skade på DCP og passe på at roboten ikke iverksetter nødstopp funksjonen. Roboten har en innebygd funksjon som gjør at roboten stoppes hvis den bruker mer enn en gitt kraft på et objekt. Dette er for å ikke ødelegge robot og omgivelser.

7.3 Wait

Wait funksjonen er som alle andre wait funksjoner. Vi bruker to forskjellige wait funksjoner. Den ene er å vente en bestemt tid og den andre er å vente på en funksjon. I våres tilfelle er dette å vente på en string fra PC("GO").

7.4 Scripts

Det blir brukt en del scripts i programmet våres. Disse scriptene blir brukt til to forskjellige ting. Den ene er å sende en beskjed (en string) til PCen fra roboten, og den andre er å lese en beskjed fra PCen. Disse scriptsa brukes for å kommunisere med PCen mens robot programmer kjører. Hensikten med dette er at programmet på PCen som styrer hele systemet alltid vet hva roboten gjør og hvor langt den er i suprogrammene. Et eksempel der det er viktig at PCen vet hvor roboten er i suprogrammene er når den utfører et arbeid på CG. Da må PCen vite hvilke knapper som er trykket så den kan verifisere at disse knappene er aktive med våpenstasjon.

8 ACT_Robotscript

Dette er programmet som styrer roboten. Dette programmet inneholder alle funksjoner og bevegelser roboten trenger for å gjøre det den får beskjed om å utføre. Programmet er bygget opp av et hovedprogram som holder på TCP kobling mellom PC og robot, variable, en loop og if-setninger som kaller på suprogrammer. Alle variable skrives med *kursiv,* alle stringer vil skrives som "Eksempelstring" og alle suprogrammer skrives som **SubP_eksempel.** Det er viktig at IP adressen i robotprogrammet settet til den IP adressen som skal kjøre ACT programmet.

8.1 Hovedprogram

Dette er programmet som kommuniserer med PCen og kaller på suprogrammer når roboten skal utføre bevegelser.

```
Program
  Init Variables
  Robot Program
    heartcount=0
    null=""
    Socket_Open= False
    loop= False
    stopdata="stopdata"
    data="data"
```

Figur 6: Variable

Det første som blir gjort i programmet er å sette noen variable som skal brukes senere.
stopdata og *data* er variable som skal holde på stringen sendt fra PC-en og bruke denne som argument i IF-setningene som kaller på suprogrammer.
heartcount er en int verdi som er satt til 0, denne brukes til å restarte TCP kommunikasjonen mellom PC og robot hvis denne feiler.
null er bare en tom string som brukes i IF-funksjoner til å sjekke om *stopdata* er tom.
Socket_Open og *loop* er boolske variable som brukes til å sette TCP kommunikasjonen og løkken som kaller på suprogrammer.

Hvis *Socket_Open* er false skal roboten koble seg til PC med TCP. Hvis roboten ikke klarer å opprette kommunikasjon med PC innen to sekunder som er standard satt fra robotleverandør så skal programmet vente i 0.2 sekunder og prøve igjen. Dette vil programmet prøve å gjøre helt til den klarer å opprette kommunikasjon med PC. Hvis programmet lykkes i å koble til PC så sender den stringen ”Connected” til PC slik at brukeren vet at det er kommunikasjon mellom robot og PC. Deretter blir *loop* og *Socket_Open* satt til true.

```
If Socket_Open= False
  Socket_Open=socket_open("192.168.0.5",30002)
If Socket_Open= True
  socket_send_string("Connected")
  loop= True
  Socket_Open= True
```

Figur 7: TCP socket

Når det er opprettet kommunikasjon, så sjekkes det om *loop* er true og om *Socket_Open* er true. Hvis begge disse IF-setningene er oppfylt sendes det en stringen ”READY” til PC-en slik at brukeren vet at den kan starte en test. Når stringen er sendt starter det en loop som kjører igjennom IF-setninger som bruker *data* variabelen til å sjekke om den stemmer med en en av IF-argumentene.
Hvis *data* = ”ON/OFF” så kaller hovedprogrammet på suprogrammet **SubP_on_off** osv. Er det ingen av IF-setningene som oppfylles så venter programmet i 0.2 sekunder før den kjører igjennom loopen igjen. Denne loopen kjøres helt til et suprogram blir startet og starter igjen når et suprogram er ferdig. Loopen vil kjøres helt til programmet stoppes eller kommunikasjonen mellom robot og PC blir stoppet.

```
If loop= True
 If Socket_Open= True
  socket_send_string("READY")
  While loop= True
   If data="ON/OFF"
    data=""
    Call SubP_on_off
   If data="SEL"
    data=""
    Call SubP_sel
   If data="ARM_ON"
    data=""
    Call SubP_arm_on
   If data="ARM_OFF"
    data=""
    Call SubP_arm_off
   If data="SAFETY_OPEN"
    data=""
    Call SubP_open_safety_caps
   If data="SAFETY_CLOSE"
    data=""
    Call SubP_close_safety_caps
   If data="TRIGGER"
    data=""
    Call SubP_trigger
   If data="PALM"
    data=""
    Call SubP_palm
   If data="MOVE_PALM"
    data=""
    Call SubP_palm_move
   If data="WARM_RST"
    data=""
    Call SubP_warm_rst
   Else
    Wait: 0.2
Else
 Wait: 0.2
```

Figur 8: Call suprogramms loop

For å motta stringer fra PC måtte vi implementere en tråd i programmet som kjører parallelt med resten av programmet. Denne tråden blir brukt til å motta det som PC-en sender og sjekke at det fortsatt finnes kommunikasjon mellom robot og PC. Tråden startes med en gang programmet starter og går så lenge programmet er på. Det første som sjekkes er om *Socket_Open* er true. Hvis variabelen er true så leser den stringen fra PC-en og putter denne i variabelen *stopdata*. Denne vil leses kontinuerlig og trenger ikke inneholde noe. Når denne leses vil den sjekke om det finnes en string fra PC. Hvis den ikke har klart å lese en string etter to sekunder vil stringen bli satt til å være tom. Neste som skjer i tråden er at roboten sender en beskjeden ”OK” til PC-en. Dette vil bli brukt til å sjekke om det finnes kommunikasjon. Deretter vil det bli sjekket flere IF-setninger, og den første sjekker om *stopdata* ikke er tom. Hvis denne variabelen ikke er tom så settes *data* til å være det samme som *stopdata*. Hvis *stopdata* er ”STOP” vil roboten sende meldingen ”stopped” til PC-en og programmet på roboten stoppes. Hvis *stopdata* er tom så plusses det på 1 i *heartcount*. Hvis *stopdata* er ”OK” så settes *heartcount* til 0 igjen. Hvis *heartcount* er lik 2 eller større vil programmet lukke kommunikasjonen, sette *loop* til false, *Socket_Open* til false og *heartcount* til 0. Dette er IF-setningen som lar roboten prøve å koble til på nytt hvis kommunikasjonen blir borte mens programmet kjører.

```
Thread_1
  If Socket_Open= True
    stopdata=socket_read_string()
    socket_send_string("OK")
    If stopdata≠null
      data=stopdata
    If stopdata="STOP"
      socket_send_string("stopped")
      Halt
    stopdata=socket_read_string()
    If stopdata=""
      heartcount=heartcount+1
    If stopdata="OK"
      heartcount=0
    If heartcount≥2
      socket_close()
      loop= False
      Socket_Open= False
      heartcount=0
    Else
      Wait: 0.5
  Else
    Wait: 0.5
```

Figur 9: Tråd

8.2 Suprogrammer

Suprogrammene er egne scripts som blir kalt fra hovedprogrammet hvis en beskjed er sendt fra PC-en. Et subprogram gjør en rekke bestemte bevegelser og funksjoner. Robotens IF-setninger inne i loopen er de som bestemmer hvilke subprogram som skal kjøres. Robotprogrammet mottar en string fra PCen når roboten skal gjøre noe. Innholdet i denne stringen brukes i IF-setningene. Når en beskjed er mottatt og en IF-setningen oppfylles så kaller hovedprogrammet et subprogram. Et eksempel på et subprogram er at roboten skal skru av eller på Våpenstasjonen.

MoveJ er hovedfunksjonen som blir brukt i suprogrammene. Vi har laget et vendepunkt som er det samme i starten og slutten av alle suprogrammer. Dette vendepunktet kalles ”Start”. Resten av vendepunktene vil variere fra subprogram til subprogram.

Alle suprogrammene inneholder to socket_send_string funksjoner der de sender en string til PC-en når programmet starter med ”Starting” og subprogramnavnet(eks. ”Starting SubP_on_off”). Den andre funksjonen er helt like i alle suprogrammer og sender stringen (”DONE”) når programmet er ferdig å kjøre. Disse stringene brukes i programmet på PC-en til å vite når roboten ufører oppgaver og når roboten er ferdig. Det er også noen andre socket_send_string funksjoner som brukes for å sende en beskjed til PC at en gitt bevegelse er utført. Dette brukes for at PCen skal vite at for eksempel palm switchen er trykket inn.

8.2.1 SubP_on_off

SubP_on_off suprogrammet er suprogrammet som gjør at roboten fysisk trykker på av/på knappen på DCP-en(Skrur av og på Våpenstasjon og DCP). Dette programmet består av en MoveJ funksjon, en Force(frame) funksjon, en Wait funksjon og en send string skript.

Roboten vil bevege seg fra startposisjon til waypoint_2. Verktøyet til roboten vil være på samme X og Y koordinater som av/på knappen til DCP og Z koordinaten er 50mm unna. Neste vendepunkt har samme X og Y koordinater men Z er på DCP. Deretter bruker roboten force funksjonen som bruker 25N i Z aksen. Det vil si at den ”dytter” verktøyet mot DCP planet med 25N kraft og knappen trykkes inn. Nå venter roboten 1 sekund før den bruker de samme vendepunktene tilbake til startposisjonen.

```
SubP_on_off
  socket_send_string("Starting SubP_on_off")
  MoveJ
    Start
    Waypoint_2
    Waypoint_5
    Force
      Wait: 1.0
    Waypoint_5
    Waypoint_2
    Start
    socket_send_string("DONE")
```

Figur 10: SubP_on_off

8.2.2 SubP_sel

SubP_sel suprogrammet er programmet som får roboten til å trykke på ”sel” knappen på DCP-en. Dette suprogrammet er bygget opp helt likt som ON/OFF. Eneste forskjellen er vendepunktene waypoint_1 og waypoint_3. Waypoint_1 er samme X og Y koordinater som sel knappen på DCP og Z er 50mm unna DCP. Neste vendepunkt er samme X og Y koordinater med Z er på DCP. Deretter brukes roboten force funksjonen som bruker 25N i Z aksen for å trykke på knappen. Deretter venter roboten i 1 sekund før den bruker de samme vendepunktene tilbake i startposisjon.

```
SubP_sel
  socket_send_string("Starting SubP_sel")
  MoveJ
    Start
    Waypoint_1
    Waypoint_3
    Force
      Wait: 1.0
    Waypoint_3
    Waypoint_1
    Start
    socket_send_string("DONE")
```

Figur 11: SubP_sel

8.2.3 SubP_open_safety_caps

SubP_open_safety_caps suprogrammet utfører bevegelsene som trengs for å vippe opp sikkerhetsbeskyttelsen til vippebryterne SYSTEM, REMOTE SAFE og SAFETY OVERRIDE. Her må vi rotere flere ledd slik at vi kan benytte oss av et annet verktøy enn det som brukes til å trykke på knapper. Det første vendepunktet(waypoint_8) er en rotering av robotens ledd slik at verktøyene står skrått oppover. waypoint_9 har samme X koordinat som sikkerhetsbeskyttelsen men Y er litt mindre slik at verktøyet skal komme seg under beskyttelsen senere i suprogrammet og Z koordinatet er 150mm unna DCP. Deretter beveger roboten seg til waypoint_16 som er nesten inntil DCP planet. I neste vendepunkt har roboten beveget seg i positiv Y retning og litt i Z slik at verktøyet ligger under sikkerhetsbeskyttelsen. Da kan roboten bruke en lineær bevegelse i negativ Z retning slik at robotens verktøy beveger seg vekk fra DCP til neste vendepunkt og bryteren blir vippet opp. Når dette er gjort beveger roboten seg tilbake til waypoint_16. Herfra kan roboten bevege seg lineært i X retning og så verktøyet ligger rett under neste sikkerhetsbeskyttelse. Her vil den utføre samme bevegelse som på første sikkerhetsbeskyttelse. Så repeteres de samme bevegelsene en siste gang for å vippe opp den siste sikkerhetsbeskyttelsen og deretter bevege seg til startposisjon.

```
SubP_open_safety_caps
  socket_send_string("Starting SubP_open_safety_caps")
  MoveJ
    Start
    Waypoint_8
    Waypoint_9
    Waypoint_16
    Waypoint_11
    Waypoint_12
    Waypoint_16
    Waypoint_13
    Waypoint_14
    Waypoint_15
    Waypoint_13
    Waypoint_17
    Waypoint_18
    Waypoint_19
    Start
  socket_send_string("DONE")
```

Figur 12: SubP_open_safety_caps

8.2.4 SubP_close_safety_caps

SubP_close_safety_caps suprogrammet utfører bevegelsene som trengs for å vippe ned sikkerhetsbeskyttelsen til vippebryterene SYSTEM, REMOTE SAFE og SAFETY OVERRIDE. Det første vendepunktet er waypoint_25 og er en rotering av robotens ledd slik at vertøyet som er øverst til vanlig vil nå være nederst, dette gjøres for at ingen av de andre verktøyene skal kollidere med DCP når vippebryterne vippes ned. Waypoint_30 er vendepunktet som er litt utenfor DCP planet med X koordinaten til sikkerhetsbeskyttelsen og Y er litt over slik at verktøyet skal kunne komme seg ovenfor og flippe sikkerhetsbeskyttelsen ned. Waypoint_10 er samme X og Y men Z er nesten på DCP. Når roboten beveger seg til neste vendepunkt(waypoint_20) så flippes bryteren ned. Bevegelsen er en lineær bevegelse i negativ Y retning. Deretter beveger roboten seg opp til waypoint_10 igjen og videre til neste bryter. Herfra blir det utført de samme bevegelsene for å vippe ned sikkerhetsbeskyttelsen. Waypoint_21 er rett over

```
SubP_close_safety_caps
  socket_send_string("Starting SubP_close_safety_caps")
  MoveJ
    Start
    Waypoint_25
    Waypoint_30
    Waypoint_10
    Waypoint_20
    Waypoint_10
    Waypoint_21
    Waypoint_22
    Waypoint_21
    Waypoint_23
    Waypoint_24
    Waypoint_25
    Start
  socket_send_string("DONE")
```

Figur 13: SubP_close_safety_caps

sikkerhetsbeskyttelsen, waypoint_22 er nedenfor(bare bevegelse i negativ Y retning) og så opp igjen til waypoint_21. Samme skjer for neste sikkerhetsbeskyttelse. Når den siste sikkerhetsbeskyttelsen er vippen ned så beveger roboten seg tilbake til start.

8.2.5 SubP_arm_on

SubP_arm_on subprogrammet uforer bevegelsene som vipper SYSTEM bryteren opp i ARMED. Programmet starter i Start posisjon. Det første vendepunktet er samme X Koordinat som ARM bryteren, Y er litt mindre slik at verktøyet skal komme under bryteren og Z er 50mm unna DCP. Neste vendepunkt er samme X og Y koordinater men Z er så nærme DCP at verktøyet ligger under vippebryteren. Deretter utfører roboten en bevegelse lineært i Y retning slik at bryteren vippes opp. Roboten bruker samme vendepunkter tilbake til startposisjon.

```
SubP_arm_on
  socket_send_string("Starting SubP_arm_on")
  MoveJ
    Start
    Waypoint_6
    Waypoint_4
    Waypoint_7
    Waypoint_4
    Waypoint_6
    Start
    socket_send_string("DONE")
```

Figur 14: SubP_arm_on

8.2.6 SubP_arm_off

SubP_arm_off suprogrammet er nesten likt som **SubP_arm_on**, men koordinatene i vendepunktene vil være noe forandres siden bryteren skal vippes ned og ikke opp. Det er også et ekstra vendepunkt som trengs for å forsikre oss at roboten ikke krasjer i noe mellom vendepunkter. Det første vendepunktet etter start(waypoint_26) vil være samme som ARM brytere, men Z er 40mm unna DCP planet. waypoint_27 har samme X og Z men Y er høyere slik at verktøyer kan komme seg over vippebryteren. waypoint_28 har samme X og Y, men Z er nærmere slik at verktøyet ligger over bryteren. Deretter beveger roboten seg lineært i negativ Y retning til neste vendepunkt. Når dette er gjort vil roboten bevege seg tilbake til waypoint_26 og deretter tilbake til startposisjon.

```
SubP_arm_off
  socket_send_string("Starting SubP_arm_off")
  MoveJ
    Start
    Waypoint_26
    Waypoint_27
    Waypoint_28
    Waypoint_29
    Waypoint_26
    Start
  socket_send_string("DONE")
```

Figur 15: SubP_arm_off

8.2.7 SubP_palm

SubP_palm er suprogrammet som aktiverer PALM switchen på control grip. Dette programmet består av en MoveJ funksjon, en Wait funksjon og noen scripts som sender beskjeder til PCen. Roboten starter i startposisjon og roterer mens den beveger seg nærmere CG slik at den får posisjonert seg litt utenfor CG med verktøyene slik at de kan komme på hver sin side av CG. Dette punktet heter cg_start og er likt for alle suprogrammene som skal utføre noe på CG. waypoint_33 ligger nærmere CG og men verktøyene vil fortsatt være rett utenfor CG. Dette vendepunktet brukes for å passe på at verktøyene ikke kommer borti CG når roboten skal bevege seg til neste vendepunkt.

```
SubP_palm
  socket_send_string("Starting SubP_palm")
  MoveJ
    Start
    cg_start
    Waypoint_33
    Waypoint_45
    Waypoint_34
    socket_send_string("DONE")
    Wait data="GO"
    socket_send_string("CONTINUES")
    Waypoint_45
    Waypoint_33
    cg_start
  socket_send_string("DONE")
```

Figur 16: SubP_palm

Til waypoint_45 vil roboten bevege seg i X retning slik at verktøyene er plassert på hver sin side av CG. Det neste ventepunktet har samme koordinater til roboten, men ikke verktøyet. Her vil roboten bare rotere sitt ytterste ledd så den kan aktivere PALM switchen. Når PALM switchen er aktivert i waypoint_34 vil roboten sende ”DONE” til PC og vente på å få beskjeden ”GO” tilbake. Når beskjeden er mottatt sender roboten beskjeden ”CONTINUES” til PCen og bruker de samme vendepunktene tilbake til startposisjon. Scripts brukes i SubP_palm, SubP_palm_move og SubP_trigger for at programmet på PCen alltid skal vite hva som teoretisk skal være aktivert så programmet kan sjekke dette opp mot våpenstasjon.

8.2.8 SubP_palm_move

SubP_palm_move har helt like vendepunkter som SubP_palm. Forskjellen er et ekstra vendepunkt og noen flere send og les funksjoner. Når roboten er i waypoint 34 vil den sende stringen ”DONE” til PCen og vente på stringen ”GO”. Når denne er mottatt vil den sende stringen ”CONTINUES” til PCen og beveger seg til waypoint_55. Dette vendepunktet har lik Z og Y som waypoint_34, men den beveger seg i X retning slik at CG beveger seg mot venstre. Når roboten er i dette vendepunktet sender den ”DONE” igjen og venter på beskjeden ”GO2” fra PCen. Når dette er mottatt sender den ”CONTINUES” igjen og bruker de samme vendepunktene tilbake til start.

```
SubP_palm_move
  socket_send_string("Starting SubP_palm_move")
  MoveJ
    Start
    cg_start
    Waypoint_33
    Waypoint_45
    Waypoint_34
    socket_send_string("DONE")
    Wait data="GO"
    socket_send_string("CONTINUES")
    Waypoint_55
    socket_send_string("DONE")
    Wait data="GO2"
    socket_send_string("CONTINUES")
    Waypoint_34
    Waypoint_45
    Waypoint_33
    cg_start
    Start
  socket_send_string("DONE")
```

Figur 17: SubP_palm_move

8.2.9 SubP_trigger

SubP_trigger er i likhet med **SubP_palm**. De to første vendepunktene er helt like. Når roboten flytter seg til waypoint_46 vil den utføre samme bevegelse som til waypoint_45 men den vil bevege seg litt lengre i X retning. Dette gjør roboten for å ta i bruk det tredje verktøyet til å trykke på trigger knappen. Når roboten er i waypoint_46 vil den rotere det ytterste leddet slik at palm switchen aktiveres, da vil roboten sende stringen "DONE" til PCen og starte med en wait funksjon som venter på en melding fra PCen. Når meldingen er mottatt sender roboten stringen "CONTINUES" til PCen og roterer det ytterste leddet litt til slik at trigger også aktiveres. Her vil roboten repetere samme funksjoner som den gjorde når palm switchen ble aktivert. Når stringen "GO2" er mottatt fra PCen så bruker den de samme vendepunktene tilbake til startposisjon.

```
SubP_trigger
  socket_send_string("Starting SubP_trigger")
  MoveJ
    Start
    cg_start
    Waypoint_33
    Waypoint_46
    Waypoint_31
    socket_send_string("DONE")
    Wait data="GO"
    socket_send_string("CONTINUES")
    Waypoint_35
    socket_send_string("DONE")
    Wait data="GO2"
    socket_send_string("CONTINUES")
    Waypoint_46
    Waypoint_33
    cg_start
    Start
  socket_send_string("DONE")
```

Figur 18: SubP_trigger

8.2.10 SubP_warm_rst

SubP_warm_rst er suprogrammet som resetter en spesifikk feilmelding på dcp. Dette skjer hvis SAFETY bryteren blir flippet opp og ned to ganger innen 1.5 sekunder. Suprogrammet beveger seg først til Waypoint_36 som er X og Y koordinater rett under vippe bryteren og Z er 50mm unna DCP. Neste vendepunkt er helt inntil DCP slik at verktøyet er rett under bryteren. På grunn av at bevegelsene må gjøres innen et gitt tidsrom så måtte vi gjøre bevegelsene som vipper bryteren til skrå bevegelser. Fra Waypoint_37 til Waypoint_38 beveger roboten seg skrått opp og ut ifra DCP slik at bryteren vippes opp og verktøyet blir plassert slik at den bare trenger å bevege seg i Z retning for å plassere verktøyet over bryteren. Deretter blir samme bevegelse utført men da nedover og utover. Da plasseres vertøyet i Waypoint_40 og er litt utenfor Waypoint_37 i Z retning. Så repeteres de samme bevegelsene en gang til for å aktivere warm reset. Når warm reset er aktivert beveger roboten seg tilbake til startposisjon.

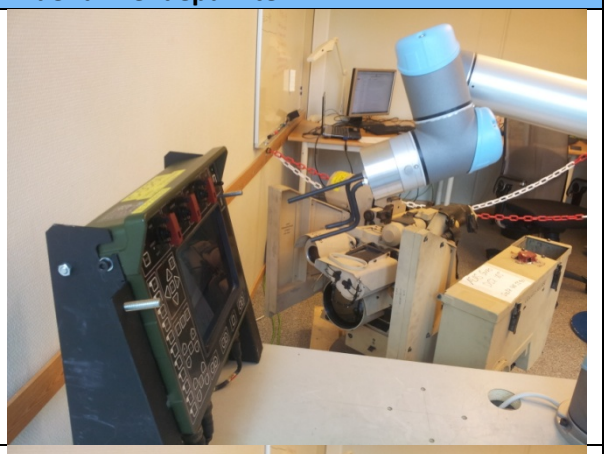
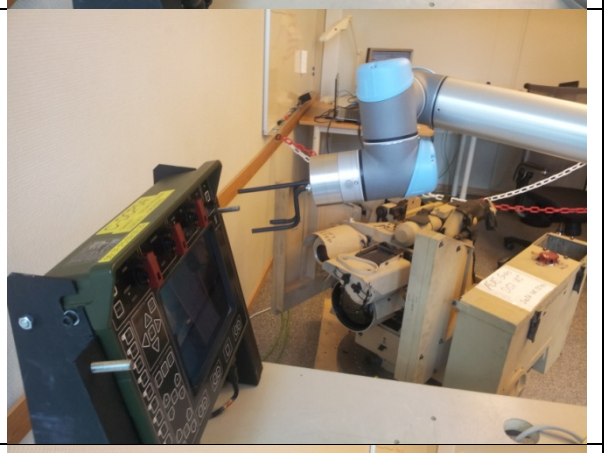
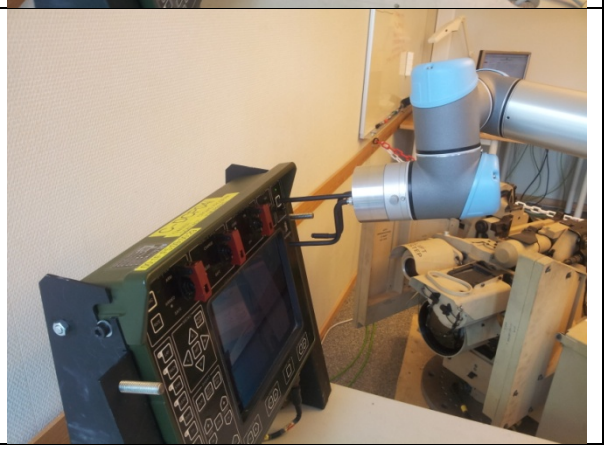
```
SubP_warm_rst
  socket_send_string("Starting SubP_warm_rst")
  MoveJ
    Start
    Waypoint_36
    Waypoint_37
    Waypoint_38
    Waypoint_39
    Waypoint_40
    Waypoint_37
    Waypoint_38
    Waypoint_39
    Waypoint_40
    Waypoint_36
    Start
  socket_send_string("DONE")
```

Figur 19: SubP_warm_rst

8.3 Waypoints bilder

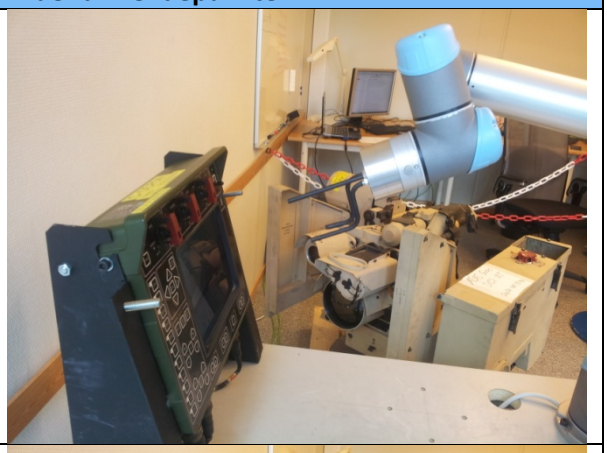
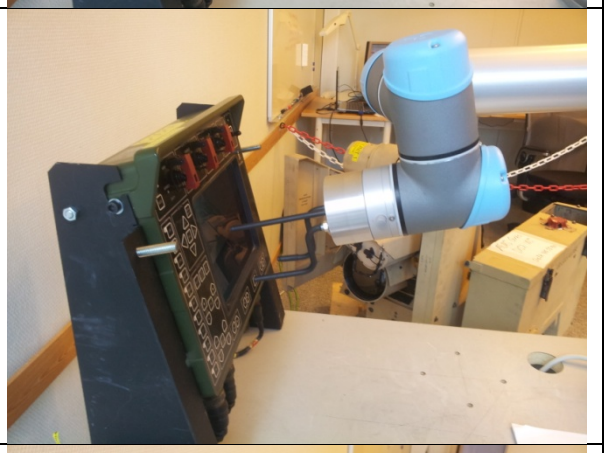
Her vil det være bilder som viser posisjonen til robot og robotens verktøy i alle Waypoints. Alle koordinater der roboten utfører en jobb på DCP er i forhold til DCP planet og der roboten utfører en jobb på CG er i forhold til CG planet. Koordinatene blir regnet fra tuppen av det rette verktøyet til roboten. Alle koordinater er i millimeter.

8.3.1 SubP_on_off

Vendepunkter:	Bilder av vendepunkter:
Start X: -170 Y: 200 Z: -140	
waypoint_2 X: -300 Y: 220 Z: -50	
waypoint_5 X: -298.5 Y: 223 Z: 0	

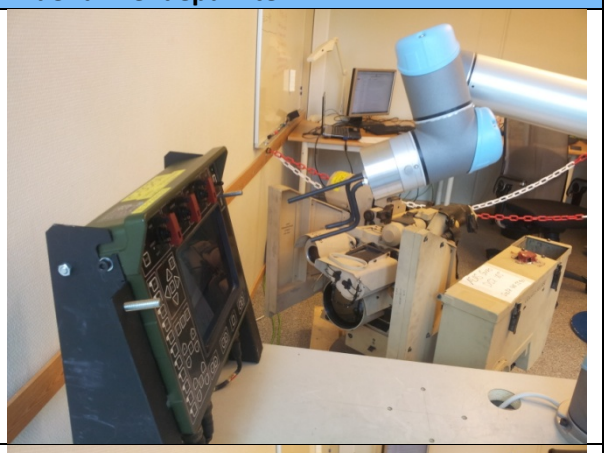
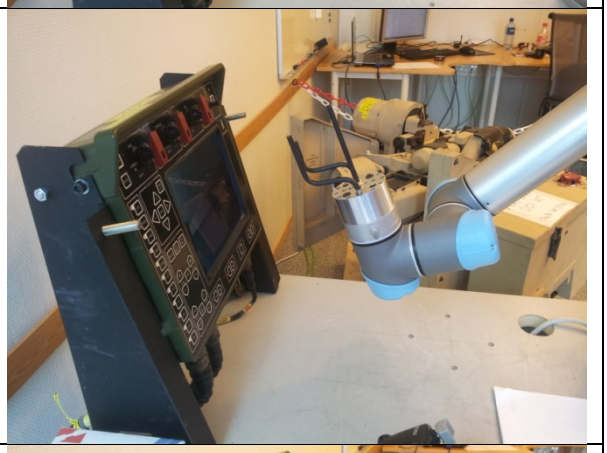
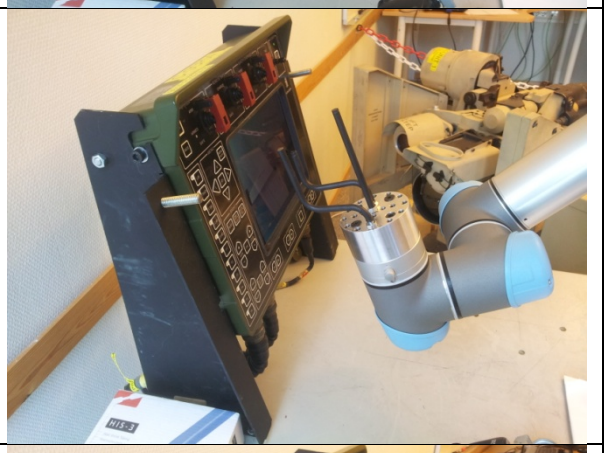
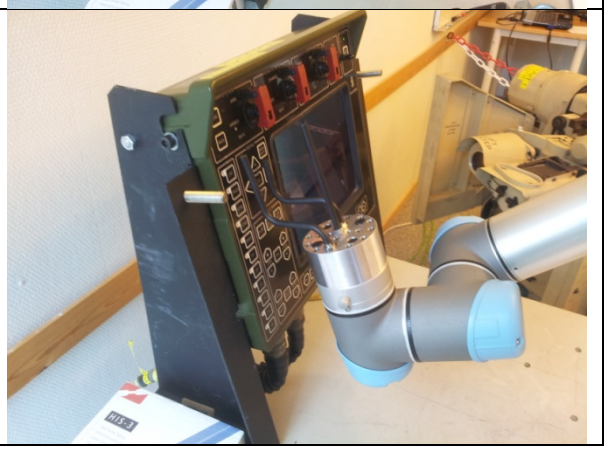
Tabell 4: SubP_on_off

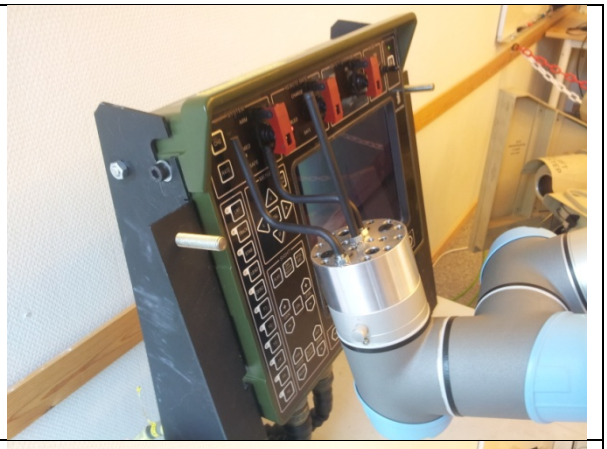
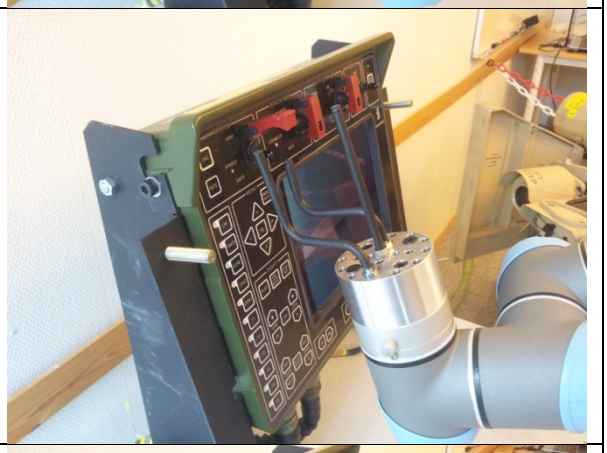
8.3.2 SubP_sel

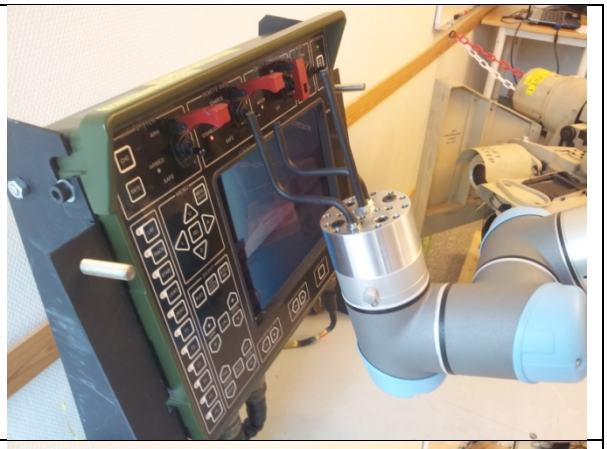
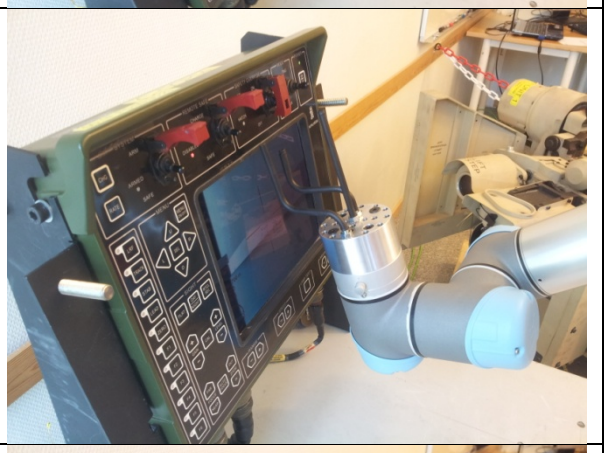
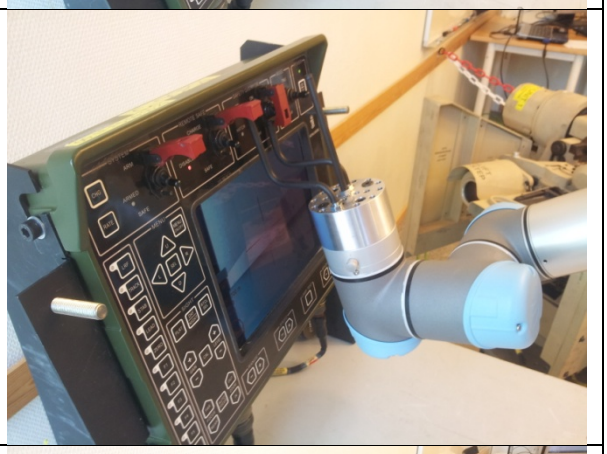
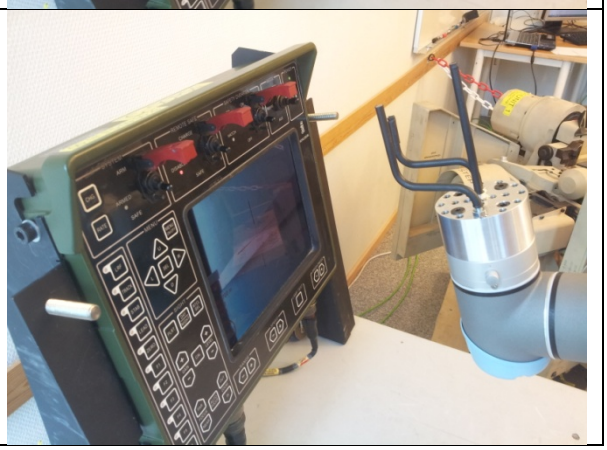
Vendepunkt:	Bilder av vendepunkter:
Start X: -170 Y: 200 Z: -140	
waypoint_1 X: -41 Y: 153 Z: -50	
waypoint_3 X: -41 Y: 153 Z: 0	

Tabell 5: SubP_sel

8.3.3 SubP_open_safety_caps

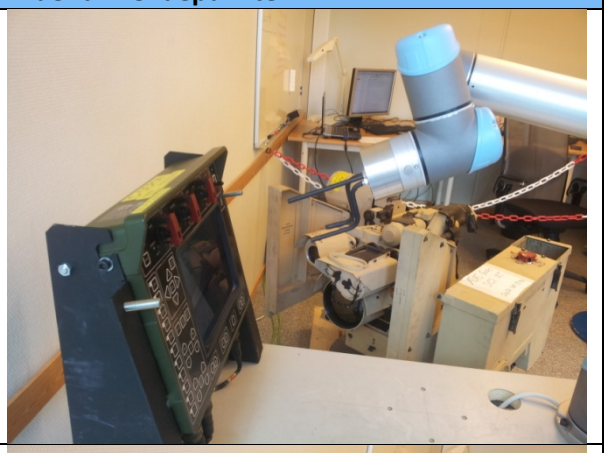
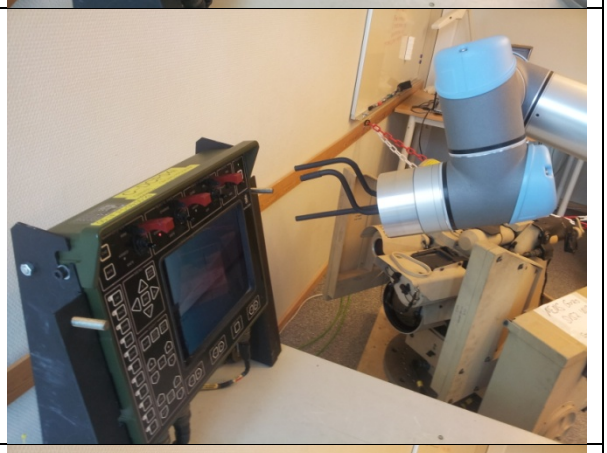
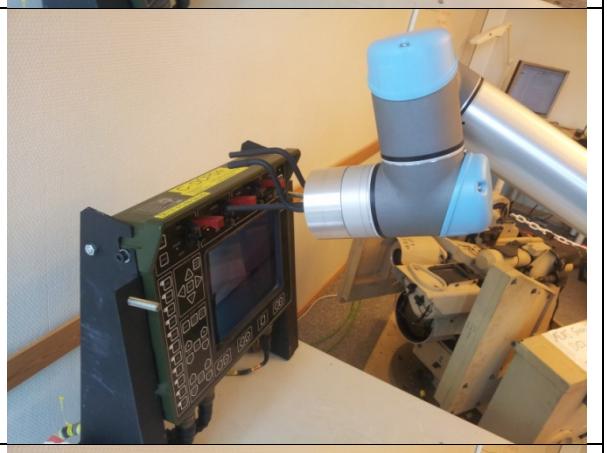
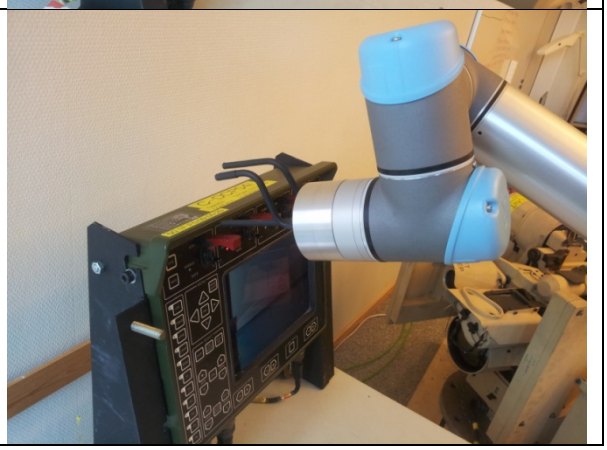
Vendepunkter:	Bilder av vendepunkter:
Start X: -170 Y: 200 Z: -140	
waypoint_8 X: -170 Y: 200 Z: -200	
waypoint_9 X: -31 Y: 210 Z: -150	
waypoint_16 X: -31 Y: 210 Z: -70	

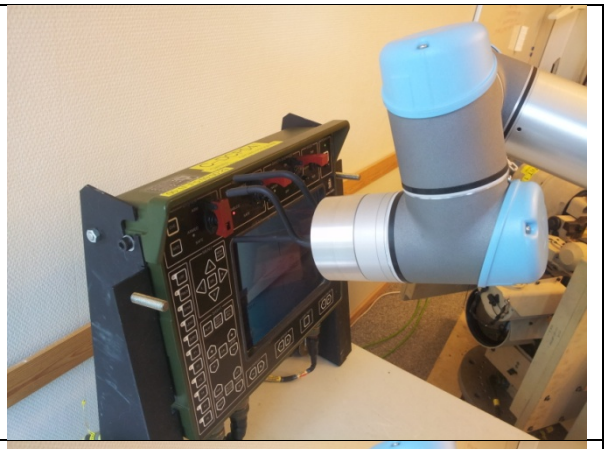
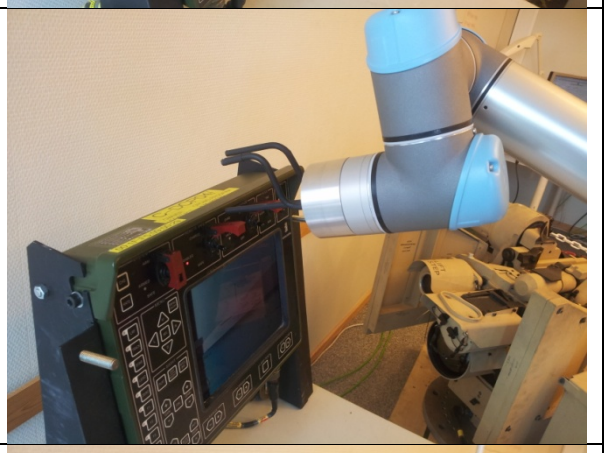
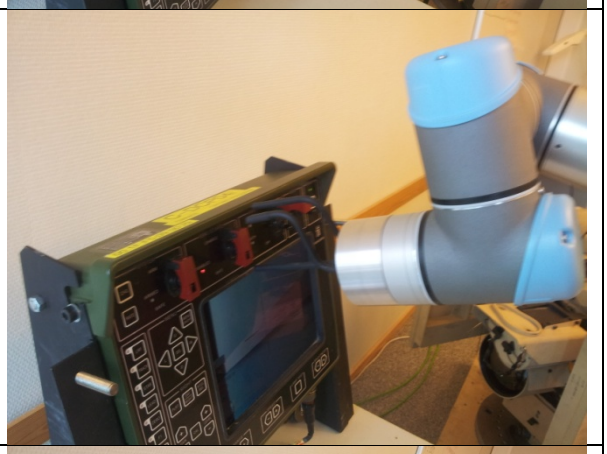
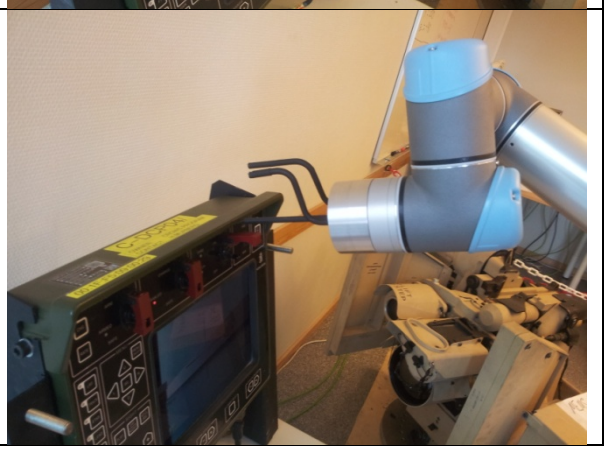
waypoint_11 X: -31 Y: 250 Z: -66	
waypoint_12 X: -31 Y: 250 Z: -100	
waypoint_13 X: -111 Y: 210 Z: -70	
waypoint_14 X: -111 Y: 250 Z: -64	

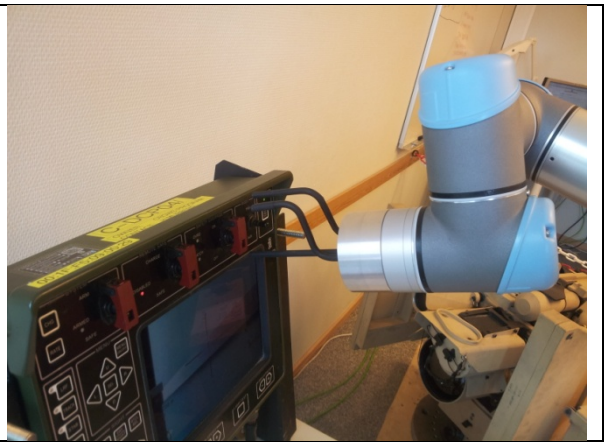
waypoint_15 X: -111 Y: 250 Z: -100	
waypoint_17 X: -194 Y: 210 Z: -70	
waypoint_18 X: -194 Y: 250 Z: -66	
waypoint_19 X: -194 Y: 250 Z: -200	

Tabell 6: SubP_open_safety_caps

8.3.4 SubP_close_safety_caps

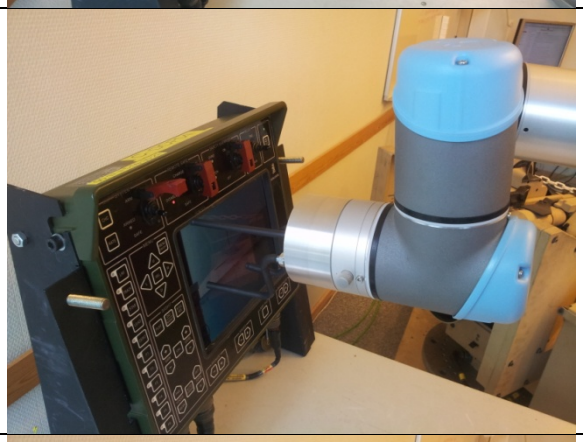
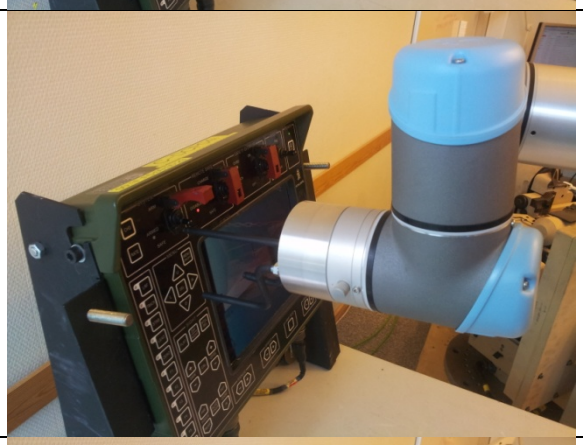
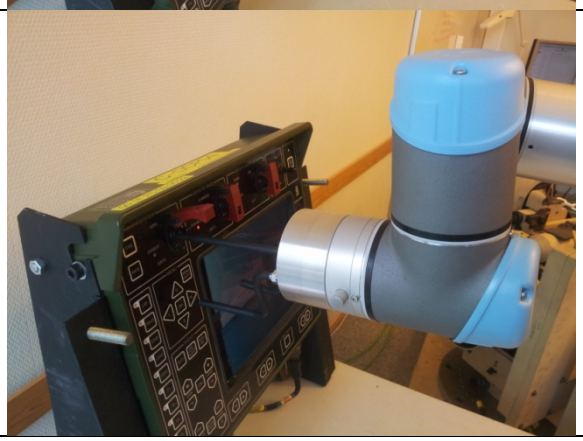
Vendepunkter:	Bilder av vendepunkter:
Start X: -170 Y: 200 Z: -140	
waypoint_25 X: -170 Y: 200 Z: -140	
waypoint_30 X: -45 Y: 260 Z: -60	
waypoint_10 X: -45 Y: 260 Z: -30	

waypoint_20 X: -45 Y: 200 Z: -30	
waypoint_21 X: -125 Y: 260 Z: -30	
waypoint_22 X: -125 Y: 200 Z: -30	
waypoint_23 X: -210 Y: 260 Z: -30	

waypoint_24 X: -210 Y: 200 Z: -30	

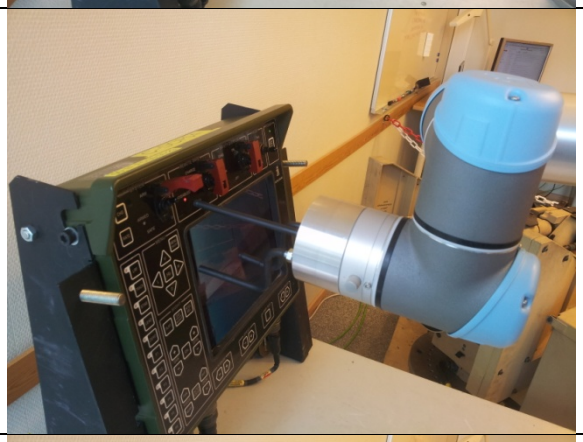
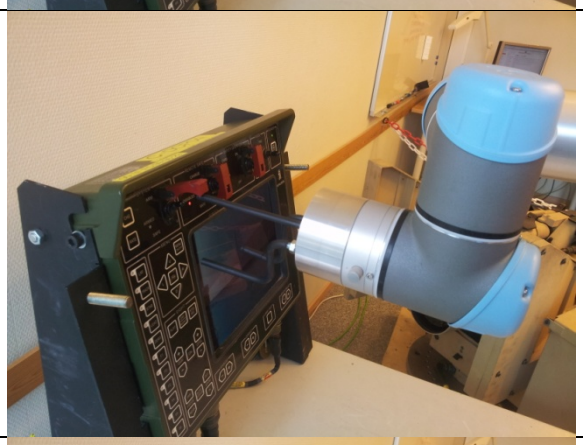
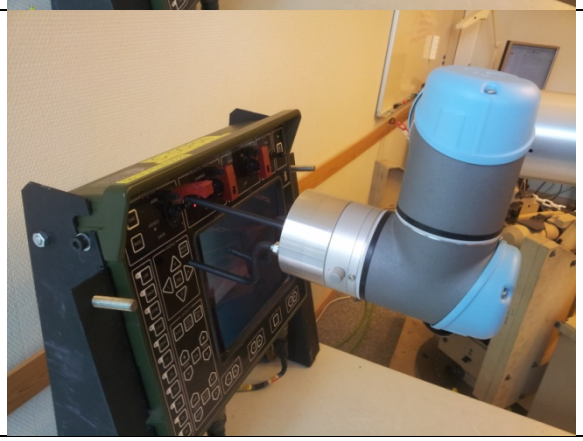
Tabell 7: SubP_close_safety_caps

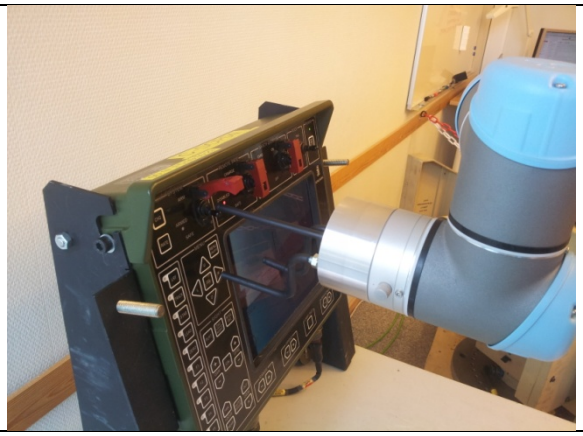
8.3.5 SubP_arm_on

Vendepunkter:	Bilder av vendepunkter:
Start X: -170 Y: 200 Z: -140	
waypoint_6 X: -52 Y: 208 Z: -50	
waypoint_4 X: -52 Y: 208 Z: -15	
waypoint_7 X: -52 Y: 219 Z: -15	

Tabell 8: SubP_arm_on

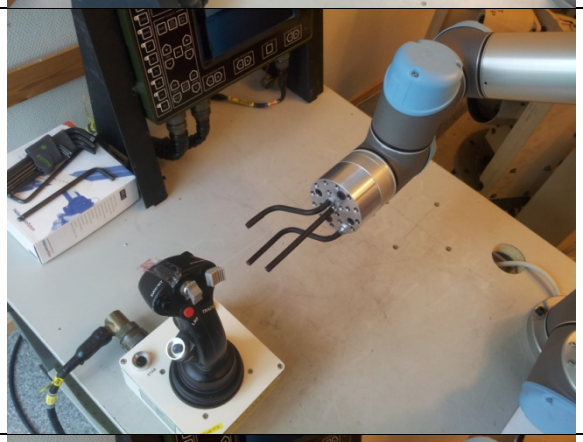
8.3.6 SubP_arm_off

Vendepunkter:	Bilder av vendepunkter:
Start X: -170 Y: 200 Z: -140	
waypoint_26 X: -53 Y: 222 Z: -40	
waypoint_27 X: -53 Y: 233 Z: -40	
waypoint_28 X: -53 Y: 233 Z: -20	

waypoint_29 X: -53 Y: 222 Z: -20	

Tabell 9: SubP_arm_off

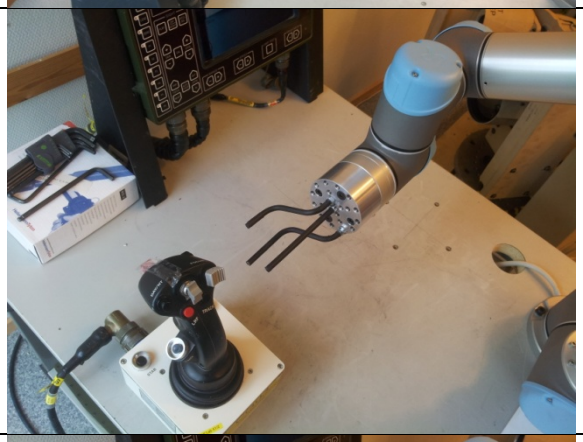
8.3.7 SubP_palm

Vendepunkter:	Bilder av vendepunkter:
Start X: -170 Y: 200 Z: -140	
cg_start X: -150 Y: 168 Z: 64	
waypoint_33 X: -96 Y: 168 Z: 64	
waypoint_45 X: -44 Y: 167 Z: 64 Wrist 3: 36.5degrees	

waypoint_34 X: -44 Y: 167 Z: 64 Wrist 3: 42 degrees	

Tabell 10: SubP_palm

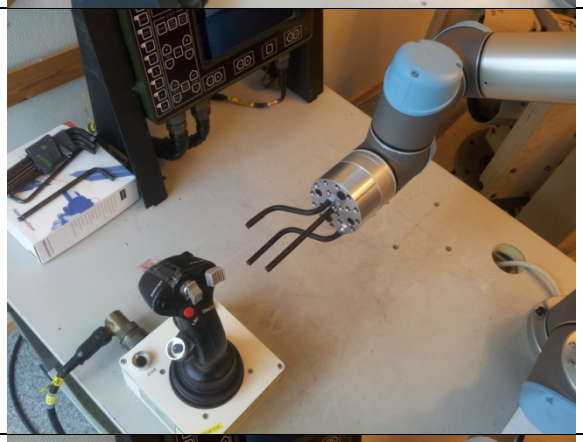
8.3.8 SubP_palm_move

Vendepunkter:	Bilder av vendepunkter:
Start X: -170 Y: 200 Z: -140	
cg_start X: -150 Y: 168 Z: 64	
waypoint_33 X: -96 Y: 168 Z: 64	
waypoint_45 X: -44 Y: 167 Z: 64 Wrist 3: 36.5degrees	

waypoint_34 X: -44 Y: 167 Z: 64 Wrist 3: 42degrees	
waypoint_55 X: -29 Y: 167 Z: 56	

Tabell 11: SubP_palm_move

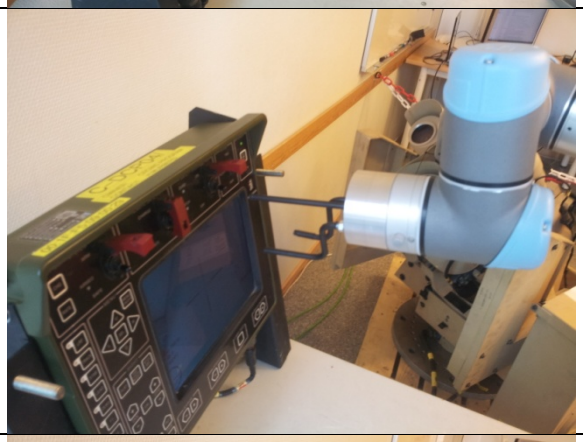
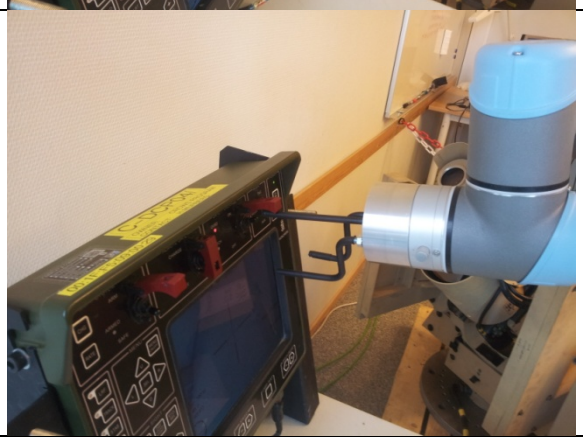
8.3.9 SubP_trigger

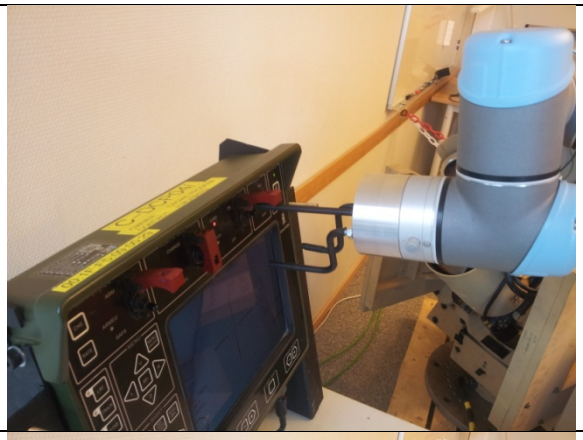
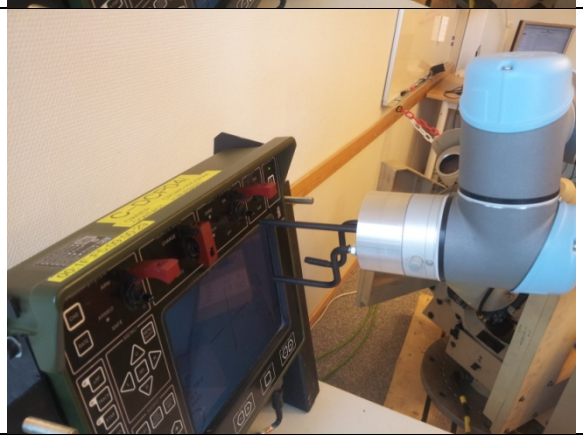
Vendepunkter:	Bilder av vendepunkter:
Start X: -170 Y: 200 Z: -140	
cg_start X: -150 Y: 168 Z: 64	
waypoint_33 X: -96 Y: 168 Z: 64	
waypoint_46 X: -22 Y: 170.5 Z: 65.5 Wrist 3: 39 degrees	

waypoint_31 X: -22 Y: 170.5 Z: 65.5 Wrist 3: 43.5 degrees	
waypoint_35 X: -22 Y: 170.5 Z: 65.5 Wrist 3: 45.5 degrees	

Tabell 12: SubP_trigger

8.3.10 SubP_warm_rst

Vendepunkter:	Bilder av vendepunkter:
Start X: -170 Y: 200 Z: -140	
waypoint_36 X: -216 Y: 208 Z: -50	
waypoint_37 X: -216 Y: 208 Z: -12	
waypoint_38 X: -216 Y: 233 Z: -30	

waypoint_39 X: -216 Y: 233 Z: -19	
waypoint_40 X: -216 Y: 208 Z: -25	

Tabell 13: SubP_warm_rst

9 Konklusjon

Roboten har alle funksjoner som trengs for å utføre testene vi trenger roboten til å gjøre.

Slik roboten er i dag kan de fleste testene til KPS kjøres. Alt som skulle være programmert til bachelorprosjektet er gjort og det gjenstår bare fremtidige utviklinger hvis KPS trenger at roboten skal utføre andre, mer spesielle oppgaver. Hvis det skal legges inn flere subprogrammer må det lages et nytt subprogram med de bevegelsene og scriptene som trengs for at en oppgave skal utføres. Dette er relativt lett å gjøre, men vil være noe tidskrevende. Det må også lages en ny IF-setning i loopen som kaller på det nye subprogrammet.

I fremtiden burde verktøyet til roboten revurderes slik at det kan lages en bedre løsning så styring av CG blir lettere og mer sikkert. Slik verktøyet er i dag vil det være en veldig liten sannsynlighet for at det bøyes slik at verktøyet ikke vil aktivere PALM og TRIGGER bryterne. Per dags dato fungerer alt som det skal, men med x antall gjentatte tester er det mulighet for at det kan oppstå feil.

Den andre tingen som ikke fungerer optimalt er planene som blir laget. Så lenge DCP og CG står på nøyaktig samme plass vil det aldri bli noe problem med at roboten ikke treffer knapper eller lignende, men med en gang det blir en endring i posisjon så må planene oppdateres som beskrevet i kapittelet Planene. Her vurderte vi å se om det var mulig å programmere roboten slik at den klarte å lage disse planene selv som en del av kalibreringen, men måtte nedprioriteres på grunn av høyere prioriteringer til andre deler av systemet. Planene fungerer optimalt slik det er nå, men det er viktig at personer som gjør endringer på posisjonen til DCP eller CG oppdaterer dette planet.

Mot slutten av prosjektet måtte vi programmere roboten på nytt og oppdatere hele dokumentasjonen da det første verktøyet ikke fungerte sikkert nok. Det ble da byttet verktøy og det ble montert på en annen måte slik at det er mer sikkert.

Fremtidige videreutviklinger som burde gjøres er forklart mer detaljert i Fremtidsdokumentet[5] .

10 Referanser

REFERANSE	DOKUMENTTITTEL	VERSJON
[1]	Technical spesifications UR5	
[2]	software_manual_no_Global	
[3]	user_manual_no_UR5_Global	
[4]	scriptmanual	
[5]	Fremtidsdokument	1.0

Tabell 14: Referanser

ACT AUTOMATED CROWS TESTING

TEK. DOK - XML

AUGUST KIND SVENDSEN
EIRIK LIEN ROA
HENRIK BERGE SØRUM
STÅLE RUDIN

KONGSBERG

HØGSKOLEN I BUSKERUD OG VESTFOLD · PROFESJONSHØGSKOLEN · HBV

Teknologidokument - XML

PROSJEKT	ACT - Automated CROWS Testing		
OPPDRAGSGIVER	Kongsberg Protech Systems		
UTFØRT VED	HBV - Høgskolen i Buskerud og Vestfold		
GRUPPE	August Kind Svendsen, Eirik Lien Roa, Henrik Berge Sørum og Ståle Rudin		
VERSJON	1.0		
ANTALL SIDER	12		
DOKUMENTHISTORIE	VERSJON	UTGITT	BESKRIVELSE
	1.0	26.05.2014	Første utgivelse

INNHOLDSFORTEGNELSE

LISTE OVER TABELLER

LISTE OVER FIGURER

1 Om dokumentet

1.1 Dokumenthistorie

VERSJON	DATO	ENDRING	SIGNATUR
0.1	01.05.14	• Dokumentet ble opprettet	AKS
1.0	26.05.14	• Første utgivelse	AKS

Tabell 1: Dokumenthistorie

1.2 Definisjoner og forkortelser

Liste som beskriver ord og uttrykk som kan være brukt i dette dokumentet:

UTTRYKK	FORKLARING
HBV	Høgskolen i Buskerud og Vestfold
KPS	Kongsberg Protech Systems (oppdragsgiver)
ACT	Automated CROWS Testing
Industrirobot	Universal Robots 5
CROWS	Common Remotely Operated Weapon Station
RWS	Remote Weapon Station
VS	Våpenstasjon
DCP	Display Control Panel
MPU	Main Processing Unit
Konfigurasjon	Forskjellige CROWS oppsett
Firing Controller	Eksternt instrument som måler fyringsspenningen
CG	Control Grip

Tabell 2: Definisjoner og forkortelser

2 Innledning

I kravspesifikasjonen[1] finner vi rammekrav 2 som er et krav om at det skal være mulig å legge til eller endre tester uten å måtte endre på kompilert kode. For å gjøre dette mulig trenger systemet et eksternt dokument som holder på informasjon om testen. Systemet skal kunne lese og deretter eksekvere innholdet i dokumentet. Ønskelig var det at dette dokumentet var enkelt å lese og navigere i både for mennesker og programmet.

For å løse dette har vi valgt å bruke XML formatet. XML er kort for «Extensible Markup Language» og er egnet til å designe og transportere data på en enkel måte. Det fine med XML er at programkoden en skriver også er enkelt å lese for mennesker. Derfor kan en person med lite XML-erfaring enkelt sette seg inn i dokumentets innhold og til og med redigere det etter kort tid. Det gjør det også egnet for prosjektgruppa som har lite erfaring med datalagring fra tidligere, og som har en stram tidsplan å tilegne seg kunnskap på området.

Mot slutten av dette dokumentet vil det vises et eksempel på et enkelt eksempel av det som vil bli gjennomgått.

3 Sammendrag

Vi har brukt XML filformatet for å løse problemstillingen i rammekrav 2. Vi har definert tre tags som blir brukt for å strukturere dokumentet, <test>, <sequence> og <cmd>. Disse tilsvarer de byggeklossene som KPS-testene består av. Test er root-noden som er den første tagen som åpnes og siste som lukkes, og brukes bare en gang per testfil. «sequence» brukes til å definere en sekvens. Hver sekvens holder på flere <cmd> tags, og det er her kommandoene i en sekvens ligger.

Disse kommandoene er navnet på metoder som ligger lagret i programmet. For å få programmet til å kjøre en metode, så skriver en ned navnet på metoden inne i <cmd> taggen, så vil det eksekveres sekvensielt.

For å opprette et XML dokument som ACT systemet kan lese holder det med standardprogrammet Notepad som følger med Windows. Etter testsekvensen er skrevet lagrer en filen som «*.xml» istedenfor <*.txt>. Eksempel på for syntaksen på en typisk testsekvens er gitt senere i dokumentet.

Ved hjelp av XML har vi nå muligheten til å opprette og redigere kjørbare testsekvenser på få minutter, uten å måtte endre kompilert kode eller ved hjelp av et spesielt program.

4 Testdokumentets oppbygning

Testene KPS utfører på våpenstasjonene de produserer har noe som kan minne om en hierarkisk struktur. I XML er dette veldig enkelt å gjenskape, fordi de som regel skrevet på den måten uansett. Ytterst i strukturen finner vi noden som holder på det hele, det vil si selve testen. Denne noden blir som regel kalt «root»-noden. I testen vi har brukt som utgangspunkt i dette prosjektet finner vi litt under 100 sekvenser. Disse utgjør det neste laget med noder. I hver sekvens finner vi igjen noen kommandoer, som regel mellom 2-4 stykker. En kommando er et steg i sekvensen, og når alle stegene er tatt så må en vurdere om sekvensen gikk som forventet eller ikke.

Med XML har man muligheten til å lage sine egne tags, som gjør det mulig å søke opp innholdet senere. For å skrive en test som ACT systemet kan kjøre trenger en bare å bruke tre enkle tags. Disse er enkle fordi de representerer de tre byggeklossene nevnt tidligere som utgjør den hierarkiske teststrukturen. Under vil vi ta for oss disse tre tagsene.

Taggen <test>

Dette er det den ytterste noden som holder på hele testen. Denne taggen er den første som åpnes og siste som lukkes, og brukes kun én gang per XML ark. I denne taggen finner vi attributtet «testname». Testname er som navnet tilsier stedet der brukeren kan sette navnet til testen, uten krav for hva som kan stå der. For eksempel <test testname = «navnpåtest»>.

Taggen <sequence>

Sequence taggen markerer når det kommer en ny sekvens. I motsetning til test taggen så brukes denne flere ganger i løpet av et testdokument. Det ene attributtet vi kan bruke her er «id», som inneholder et unikt tall som stiger for hver sekvens. Denne bruker vi for å gjøre det mulig å søke opp spesifikke sekvenser, slik at vi for eksempel får muligheten til å bestemme hvilken sekvens testen skal starte fra.

Taggen <cmd>

Det er i denne taggen vi finner det viktigste innholdet i dokumentet. Her ligger selve kommandoene som ACT systemet kommer til å lese av og eksekvere. Attributtet vi bruker her heter «part», og brukes på samme måte som «id» i sequence taggen. I hver sekvens vil «part» starte på 1, og stige med én verdi for hver nye kommando. Når en ny sekvens da starter vil denne altså starte på 1, selv om forrige sekvens også gjorde det.

Selve innholdet vi finner i denne taggen vil være navn på metoder som ACT systemet vil lese av og kjøre. Derfor er det viktig å skrive navnet helt riktig, eller så vil ikke programmet klare å skjønne hva som står der. Nedenfor har vi en oversikt over hvilke metoder som er mulig å putte inn, og en liten beskrivelse som forklarer hva metoden gjør.

4.1 Oversikt metoder

Navn	Beskrivelse
dcp_press_on	Robot trykker PÅ-bryteren på DCP
dcp_press_off	Robot trykker AV-bryteren på DCP
dcp_press_sel	Robot trykker SELECT på DCP
dcp_press_safety_open	Robot åpner safety caps
dcp_press_safety_close	Robot lukker safety caps
dcp_set_arm_on	Robot setter RWS i ARM
dcp_set_arm_off	Robot skrur ARM av
dcp_warm_reset	Robot tvinger frem en «Warm reset»
cg_press_palm	Robot trykker PALM på CG
cg_press_palm_move	Robot trykker PALM og beveger CG (med lov)
cg_press_palm_not_move	Robot trykker PALM, beveger CG (uten lov)
cg_press_palm_fire	Robot trykker PALM og FIRE på CG (med lov)
cg_press_palm_no_fire	Robot trykker PALM og FIRE på CG (uten lov)
connect_WS	Kobler til våpenstasjonen
verify_armed	Verifiserer at ARM er satt til ON
verify_safe	Verifiserer at ARM er satt til
verify_firing_enabled	Verifiserer ARM on og palm aktiv
verify_firing_not_enabled	Verifiserer ARM off og palm inaktiv
verify_Fire	Verifiser at VS fyrer
verify_NoFire	Verifiser at VS ikke fyrer
verify_Palm	Verifiser at PALM
verify_MovementAfterAllowed	Verifiserer om VS beveger seg
verify_MovementAfterNotAllowed	Verifiserer VS bevegelse når ikke tillatt
verifyMessage	Verifiserer feilmelding og skjermbildet
verifySequence	Verifiserer alle kommandoene i sekvensen

Tabell 3: Oversikt metoder

5 Hvordan opprette et testdokument

En fin egenskap med XML formatet er at det å opprette et dokument er veldig enkelt. I ACT programmet er det integrert en liten guide på dette som en finner ved å trykke på «Help» tabben og deretter «XML guide».

For å opprette et XML dokument følger en disse fire stegene:

1. Lag en ny notepad-fil.
2. Fyll inn XML kode (eksempelkode er vist senere i dokumentet).
3. Trykk «Fil» og «Lagre som».
4. Bytt ut «*.txt» med «*.xml» og trykk «Lagre».

Hvis disse stegene ble fulgt og gjort riktig, så skal du nå sitte med en test om ACT systemet kan kjøre.

Vær obs på:

- Siste kommando i hver sekvens må være «verifySequence» for at sekvensen skal gå gjennom en verifikasjon på om den sekvensen som ble kjørt skal bli godkjent eller ikke.
- Hvis en sekvens i testen er slik at ACT systemet ikke kan kjøre den, må det likevel deklareres i XML dokumentet som en tom sekvens. Dette er for at sekvensnummeret i rapporten skal stemme overens med resultatet til den riktige sekvensen.

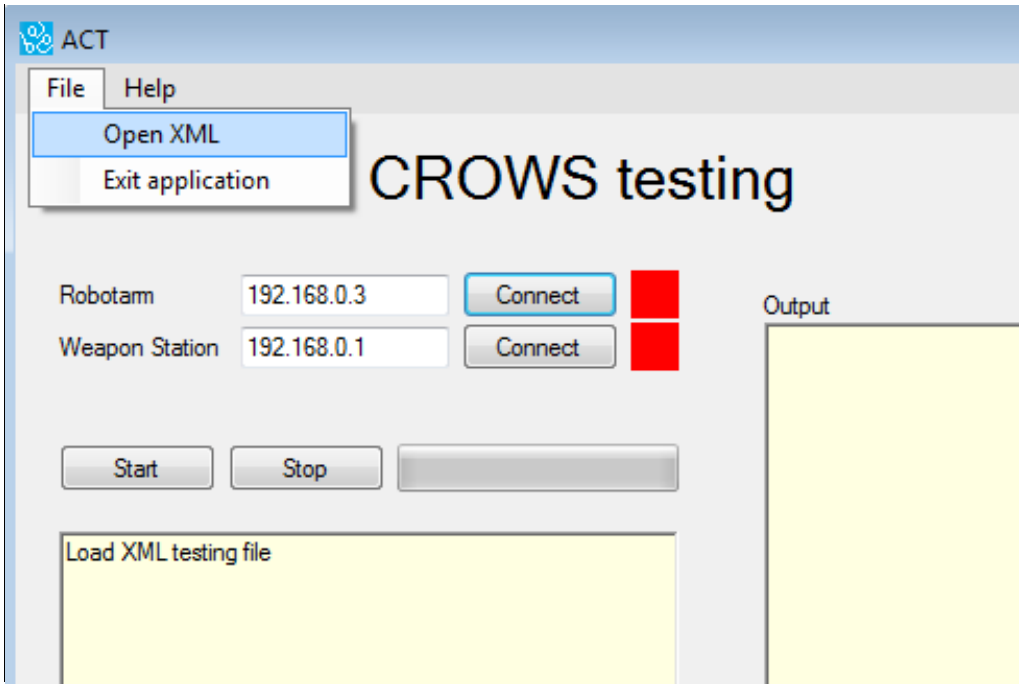
5.1 Testeksempel

Under er et eksempel på en vilkårlig testsekvens på to sekvenser som inneholder tre kommandoer hver (testsekvensen er ment som et eksempel på syntaks, og vil ikke nødvendigvis være en test som burde kjøres):

```
<test testname = «random_testname»>
  <sequence id = «1»>
    <cmd part = «1»>dcp_press_on</cmd>
    <cmd part = «2»>dcp_press_sel</cmd>
    <cmd part = «3»>dcp_set_arm_on</cmd>
    <cmd part = «4»>verifySequence</cmd>
  </sequence>
  <sequence id = «2»>
    <cmd part = «1»>cg_press_palm_fire</cmd>
    <cmd part = «2»>dcp_press_sel</cmd>
    <cmd part = «3»>dcp_warm_reset</cmd>
    <cmd part = «4»>verifySequence</cmd>
  </sequence>
</test>
```

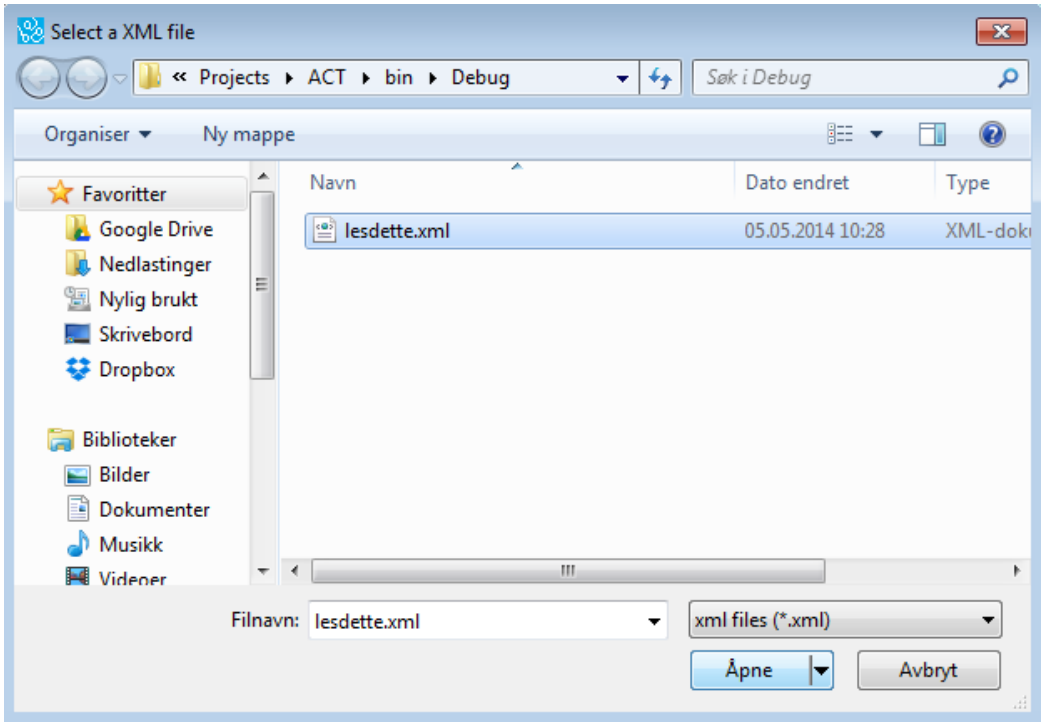
6 Laste inn test

Det er enkelt å laste inn testfilen i ACT programmet. Når programmet er åpnet starter en først ved å trykke «File»-tabben øverst i vinduet som vist nedenunder. (GUI er fortsatt under utvikling på bildet).



Figur 1: Laste inn test

Da vil en få opp et nytt vindu som lar deg bla gjennom datamaskinens filer. Klikk deg frem til XML-fila og velg «Åpne». Nå skal testen være lastet inn. (GUI er fortsatt under utvikling på bildet (an XML)).



Figur 2: Åpne XML fil

7 Avlesning og behandling

Når testdokumentet er lastet inn i programmet er neste steg å prosessere det. I dette avsnittet vil vi gå gjennom koden vi har skrevet for at dette har blitt mulig. For spørringene har vi brukt LINQ.

Først må filen lastes inn og gjøres om til noe programmet kan behandle. I dette tilfellet laster vi inn filen og lagrer det som et element ved hjelp av klassen «XElement». Under ser vi koden, der vi kaller det innlastede elementet for «test». Det inne i parentesen er for lokasjonen av XML dokumentet.

```
// loads the XML file
XElement test = XElement.Load(MyGlobals.strFilePath + "/" + MyGlobals.strFileName);
```

Nå har vi et element kan vi gjøre spørringer på for å hente ut det vi ønsker. Det første vi henter ut er navnet på testen, som ligger lagret i attributtet til test noden. Vi deklarerer hva resultatet skal hete, hvor spørringen skal skje, for så å filtrere ut det som ligger lagret i attributtet. Deretter bruker vi en «foreach» løkke som printer ut resultatet, i dette tilfellet bare én gang.

```
// checks testname and how many sequences there is
var selectTestInfo = from i in test.DescendantsAndSelf("test")
                     select new
                     {

                         testName = (string)i.Attribute("testname")
                     };

foreach (var item in selectTestInfo)
{
    Console.WriteLine("The current test running is named: ");
    Console.WriteLine(item.testName);
    Console.WriteLine("\r\n");
    MyGlobals.testName = item.testName;
}
```

I neste spørring skal vi få et resultat som holder på alle sekvensene. Dette gjør vi på samme måte som i forrige spørring. Denne gangen henter vi ut alle sekvenselementer som ikke er tomme. Vi kan nå bruke resultatet av spørringen til å f.eks. telle antall sekvenser ved å bruke .Count() metoden som vist nedenfor. Dette tallet vil bestemme størrelsen på arrayet som vil holde på sekvensene.

```
// selects sequences with data
var selectedSeqs = from i in test.DescendantsAndSelf("test")
                   where i.Element("sequence") != null
                   select i;

// counts sequences
var seq_list = selectedSeqs.DescendantsAndSelf("sequence");
Console.WriteLine("Got {0} sequences in this test.", seq_list.Count());
Console.WriteLine("\r\n");


MyGlobals.sequenceCount = seq_list.Count();
MyGlobals.sequence = new string[seq_list.Count()][];
```

Nå vil vi bla gjennom sekvensene ved hjelp av en for-løkke, for så å hente ut de kommandoene de holder på. Vi bruker attributtet «id» som ligger i «sequence»-taggen for å skille på sekvensene. Tallet for-løkka er på vil være lik id verdien til sekvensen.

```
// goes through the seqs and prints their data
for (int counter = 1; counter < seq_list.Count() + 1; counter++)
{
    // converts the counter int to string
    string counterString = counter.ToString();
    Console.WriteLine("Counterstring value is: " + counterString);

    var selected = from cli in test.DescendantsAndSelf("sequence")
                   where (string)cli.Attribute("id") == counterString
                   select cli;
```

I dette resultatet skal det ligge alle kommandoene til en bestemt sekvens. Vi trenger nå å telle antall kommandoer for å lage det arrayet som de skal lagres i. Dette gjør vi på samme måte som da vi telte antall sekvenser. I arrayet putter vi selve innholdet i kommandonoden ved å skrive «cmd.Value» for å hente det ut, der innholdet vil være en string som representerer navnet til en metode.

```
foreach (var g in selected)
{
    idStringHolder = g.Attribute("id").Value;
    Console.WriteLine("Current sequence running: " + idStringHolder);
}

// counts commands
var cmd_list = selected.DescendantsAndSelf("cmd");
Console.WriteLine("Got {0} cmds in this sequence.", cmd_list.Count());

MyGlobals.sequence[sequenceNr] = new string[cmd_list.Count()];
int cmdNr = 0;

foreach (var cmd in cmd_list)
{
    Console.WriteLine("cmd part={0} holds: " + cmd.Value, cmd.Attribute("part").Value);
    MyGlobals.sequence[sequenceNr][cmdNr] = cmd.Value;
    cmdNr++;
}
```

8 Konklusjon

Etter å ha tatt i bruk teknologien forklart ovenfor kan vi nå lage og endre tester uten å måtte endre på kompilert kode. I fremtiden kan det tenkes at testene kan holde på mer informasjon ved at flere attributter legges til, som for eksempel estimert testtid. Dette vil ikke kreve mye endring i den kompilerte koden, og skal være mulig å hente ved enkle spørringer med bruk av LINQ. Andre mulig utvidelser relatert til XML-delen av systemet er drøftet i Fremtidsdokumentet[2].

9 Referanser

REFERANSE	DOKUMENTTITTEL	VERSJON
[1]	Kravspesifikasjon	2.0
[2]	Fremtidsdokument	1.0

Tabell 4: Referanser

ACT AUTOMATED CROWS TESTING

TEK. DOK - SOFTWARE

AUGUST KIND SVENDSEN
EIRIK LIEN ROA
HENRIK BERGE SØRUM
STÅLE RUDIN

KONGSBERG

HBV
HØGSKOLEN I BUSKERUD OG VESTFOLD · PROFESJONSHØGSKOLEN ·

Teknologidokument - Software

PROSJEKT	ACT - Automated CROWS Testing		
OPPDRAGSGIVER	Kongsberg Protech Systems		
UTFØRT VED	HBV - Høgskolen i Buskerud og Vestfold		
GRUPPE	August Kind Svendsen, Eirik Lien Roa, Henrik Berge Sørum og Ståle Rudin		
VERSJON	1.0		
ANTALL SIDER	18		
DOKUMENTHISTORIE	VERSJON	UTGITT	BESKRIVELSE
	1.0	26.05.14	Første utgivelse

INNHOLDSFORTEGNELSE

LISTE OVER TABELLER

LISTE OVER FIGURER

1 Om dokumentet

1.1 Dokumenthistorie

VERSJON	DATO	ENDRING	SIGNATUR
0.1	18.05.14	• Dokumentet ble opprettet	SR
1.0	26.05.14	• Første utgivelse	SR

Tabell 1: Dokumenthistorie

1.2 Definisjoner og forkortelser

Liste som beskriver ord og uttrykk som kan være brukt i dette dokumentet:

UTTRYKK	FORKLARING
HBV	Høgskolen i Buskerud og Vestfold
KPS	Kongsberg Protech Systems (oppdragsgiver)
ACT	Automated CROWS Testing
UR5	Universal Robots 5
CROWS	Common Remotely Operated Weapon Station
RWS	Remote Weapon Station
DCP	Display Control Panel
MPU	Main Processing Unit
Konfigurasjon	Forskjellige CROWS oppsett
Firing controller	Eksternt instrument som måler fyringsspenningen
CG	Control grip

Tabell 2: Definisjoner og forkortelser

2 Innledning

ACT programvaren skal kunne utføre automatiske tester på CROWS konfigurasjoner. Vi har delt opp programmet i flere deler og i dette dokumentet vil selve programmet bli forklart med fokus på GUI, systemkontroller, kommunikasjon og testrapport.

Programmet har en del forskjellige egenskaper som må være på plass for at den skal kunne utføre tester på CROWS konfigurasjoner. Den må kunne kommunisere med både robot og våpenstasjon, og må ha en metode for å kunne kjøre testene sekvensielt med tanke på hvilke metoder den skal kalle på.

Når KPS kjører disse testene manuelt må de krysse av om de forskjellige sekvensene blir godkjent eller ikke. I ACT programvaren må vi ha en lignende funksjon som må skje automatisk. Derfor valgte vi å lage en lignende tabell som skulle inneholde riktig sekvensnummer og om sekvensen ble godkjent eller ikke.

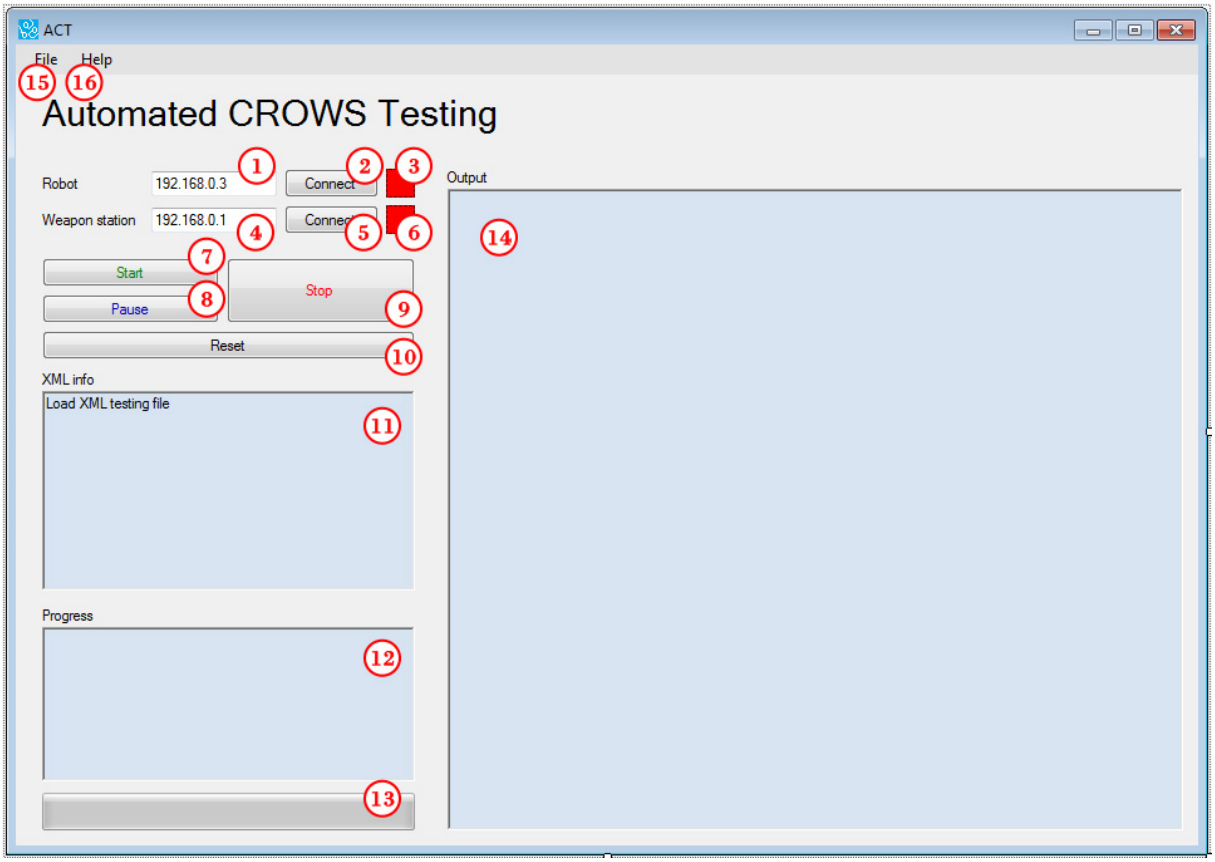
3 Sammnedrag

Vi lagde en GUI så enkel som mulig slik at det skulle være lett for brukeren å navigere seg frem. Med passende navn til de forskjellige knappene brukeren kan trykke på, og korte guider på hvordan man bruker de forskjellige egenskapene til programmet.

I dette programmet er det mye kommunikasjon som skjer over et lokalt nettverk, derfor er det viktig å bruke en pålitelig kommunikasjonsmetode. Dette blir nærmere forklart senere i dokumentet.

For at programmet skal kunne gjennomføre en test må den ha en kontroller som kan holde styr på hvilke oppgaver som skal bli utført til enhver tid. I har vi brukt en database for å lagre resultatene for hver sekvens som er kompatibel med ACT programvaren.

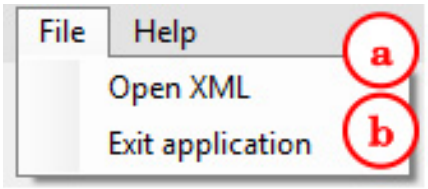
4 Overordnet - GUI



Figur 1: Illustrasjon av GUI

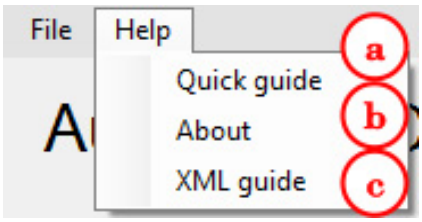
1. Her tenkte vi at man skal kunne skrive ip adressen til roboten, men den blir ikke brukt i programmet slik som det er nå.
2. Når du trykker på denne knappen vil programmet sette opp en TCP server som lytter om det er noe aktivitet på en hvilken som helst ip adresse. Hvis den finner aktivitet på en ip adresse vil den starte en egen tråd som holder denne tilkoblingen. Denne tilkoblingen kan både sende og motta stringer.
3. Denne røde firkanten blir grønn når kommunikasjonen med roboten er oppe.
4. Her skal du skrive inn ip adressen til våpenstasjonen (DCP).
5. Når du trykker på denne knappen sjekker systemet om du kan få kommunikasjon med våpenstasjonen. Dette er kun en sikkerhetsfunksjon slik at programmet vet om den kan få kommunikasjon når du starter en test.
6. Denne røde firkanten blir grønn om programmet får satt opp kommunikasjon med våpenstasjonen (DCP).
7. Du kan ikke trykke på denne knappen før 3 og 6 er grønne. Dvs. at du må først sette opp en kommunikasjon med roboten, og se at du kan få kommunikasjon med våpenstasjonen. I tillegg må du også velge et XML dokument som inneholder en test før du kan trykke på denne knappen. Når denne knappen blir trykket så starter programmet testen som ligger i XML dokumentet brukeren har lastet opp.

8. Når du trykker på pause knappen vil programmet gjøre ferdig den kommandoen den er på deretter pause programmet. Når du trykker på den samme knappen en gang til vil programmet fortsette der den var.
9. Stopp knappen er en nødstopp. Når du trykker på denne knappen vil hele programmet stoppe alle tråder i programmet, og sender en kommando til roboten som gjør at den stopper uansett om den er midt i en bevegelse og stopper programmet som kjører på roboten. Noe som vil si at du må starte robotprogrammet på nytt før du kan sette opp kommunikasjon igjen.
10. Denne knappen er for å nullstille programmet etter du er ferdig med en test slik at du kan laste inn et nytt XML dokument slik at du kan starte en ny test uten at du må restarte programmet.
11. I denne tekstboksen vises info om XML dokumentet du har lastet inn i programmet. Slik som hva testen heter, hvor mange sekvenser det er og totalt antall kommandoer.
12. Når du starter testen kan du se hvilke sekvens og kommando programmet utfører til enhver tid.
13. Dette er en illustrasjon på hvor langt du har kommet i testen. Dette er en «progress bar» som har så mange nivåer som det er kommandoer. For hver kommando som blir ferdig vil den gå opp ett nivå.
14. I denne tekstboksen vil du kunne se hva som blir utført av programmet under en test.
15. Her har du to alternativer:
    a. Open XML:
       Hvis du trykker på denne kan du velge en XML fil som inneholder en av testene.
    b. Exit application:
       Trykker man her vil man avslutte hele programmet.



Figur 2: Illustrasjon av File tab

16. Her har to tre alternativer
    a. Quick guide:
       Her står det kort introduksjon på hvordan man bruker programmet.
    b. About:
       Her står det kort om hva hensikten med programmet er og hvem som har lagd den.
    c. XML guide:
       Her har du en guide på hvordan man setter opp xml dokumentet med alle de forskjellige metodene man kan bruke og hva de brukes til.



Figur 3: Illustrasjon av Help tab

5 Software

5.1 Kommunikasjon

TCP/IP

TCP/IP står for ”transmission control protocol / internet protocol” og er den mest populære og brukte protokollen for nettverkskommunikasjon. Vi har brukt TCP/IP-kommunikasjon over en socket på et lokalt nettverk. For å få til dette må vi ha en server og en klient. Vi valgte å sette ACT programvaren som server og robot som klient.

Server: Setter opp en socket på en bestemt port og lytter om det er noe aktivitet på denne porten.

Klient: En klient trenger IP-adressen til serveren og hvilke portnummer den lytter på.

5.1.1 Robot

Definerer først hvilke variable vi trenger for å sette opp TCP/IP serveren.

```
private TcpListener RC_Server;
private Thread RC_Listner_Thread;
private TcpClient RC_Client;
```

Figur 4: Definerer variable

Så starter vi en ny tråd som vil starte serveren.

```
public void Server()
{
    this.RC_Server = new TcpListener(IPAddress.Any, 30002);
    this.RC_Listner_Thread = new Thread(new ThreadStart(ListenForRCClient));
    this.RC_Listner_Thread.Start();
}
```

Figur 5: Server matode

Denne tråden vil kjøre i en evig loop og lytte om det er noe aktivitet på portnummer 30002. Når den oppdager aktivitet vil den opprette en ny tråd som vil sette opp kommunikasjon med klienten som prøver å koble seg til.

```
private void ListenForRCClient()
{
    this.RC_Server.Start();

    while (true)
    {
        //Block before client is connected
        RC_Client = this.RC_Server.AcceptTcpClient();


        //create thread to handle communication with connected client
        Thread RC_Client_Thread = new Thread(new ParameterizedThreadStart(HandleRCClientComm));
        RC_Client_Thread.Start(RC_Client);
    }
}
```

Figur 6:listenForRCClient metode

Det neste vi gjorde var å sette opp en metode som leser det roboten skriver over socket-en.

```
public void HandleRCClientComm(object RCclient)
{
    TcpClient tcpRCclient = (TcpClient)RCclient;
    NetworkStream RCclientStream = tcpRCclient.GetStream();
    byte[] message = new byte[4096];
    int bytesRead;

    byte[] write = new byte[4096];

    CommunicationRC_ReceiveData.CommunicationRC_Receive("Connected");
    form.showRobotConnectionStatus(form.picRobotStatus);
    form.btnEnableWSConnect(form.btnWSConnect);
    while (true)
    {
        bytesRead = 0;

        try
        {
            bytesRead = RCclientStream.Read(message, 0, 4096);
        }
        catch
        {
            break;
        }
        if (bytesRead == 0)
        {
            break;
        }
        ASCIIEncoding encoder = new ASCIIEncoding();
        System.Diagnostics.Debug.WriteLine(encoder.GetString(message, 0, bytesRead));
        string strMessage = encoder.GetString(message, 0, bytesRead);
        CommunicationRC_ReceiveData.CommunicationRC_Receive(strMessage);

        string response = default(string);
        if (strMessage == "OK")
        {
            CommunicationRC_SendData.CommunicationRC_Send("OK");
        }
        else if (strMessage == null)
        {
            CommunicationRC_ReceiveData.CommunicationRC_Receive("Stringen er null");
        }
        if (response != default(string))
        {
            SendToRC(response);
            CommunicationRC_ReceiveData.CommunicationRC_Receive(response);
        }
    }
    tcpRCclient.Close();
}
```

Figur 7: HandleRCClientComm metode

Den neste metoden vi lagde var for å kunne sende meldinger over socket-en og til roboten.

```
public void SendToRC(string msg)
{
    byte[] DataToRC;
    DataToRC = Encoding.Default.GetBytes(msg);

    try
    {
        if (this.RC_Client.Connected == true)
        {
            this.RC_Client.GetStream().Write(DataToRC, 0, DataToRC.Length);
        }
    }
    catch (Exception e) { }
}
```

Figur 8: SendToRC metode

5.1.2 Våpenstasjon

Når vi skal kommunisere med våpenstasjonen gjør vi også dette over TCP/IP, men denne gangen er det våpenstasjonen som er serveren og ACT programvaren klient. Denne kommunikasjonen skjer over telnet porten som er 23. Telnet blir brukt til å utveksle 8 bits informasjon begge veier mellom server og klient.

Når vi skal koble til våpenstasjonen trenger vi IP-adressen og hvilke portnummer vi skal koble oss opp til. Siden vi bruker telnet bruker vi port nummer 23. Se i installasjonsguiden[1] for å finne IP-adressen til våpenstasjonen.

```
public void WS_Connect(string WS_IpAdress, int WS_Port)
{
    try
    {
        WS_RemoteEndPoint = new IPEndPoint(IPAddress.Any, WsPort);
        WS_tcpclient = new TcpClient();
        WS_tcpclient.Connect(IPAddress.Parse(WS_IpAdress), WsPort);
        CommunicationWS_SendData.CommunicationWS_Send("Dette er data fra WS" + Environment.NewLine,false);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

Figur 9: WS_Connect metode

Her starter vi en ny tråd av metoden “tfunc”.

```
public void WS_Receive()
{
    MyGlobals.tTFunc = new Thread(tfunc);
    MyGlobals.tTFunc.Start();
}
```

Figur 10: WS_Recieve metode

I denne metoden lytter vi om våpenstasjonen prøver å sende en string og leser det den sender.

```
private void tfunc()
{
    int bytes;
    string msgWS;
    while (WS_tcpclient.Connected == true)
    {
        if (WS_tcpclient.Connected == true)
        {
            try
            {
                while (true)
                {
                    bytes = WS_tcpclient.GetStream().Read(WS_datalenght, 0, 1024);
                    msgWS = Encoding.ASCII.GetString(WS_datalenght, 0, bytes);
                    CommunicationWS_ReceiveData.CommunicationWS_Receive(msgWS);
                    Console.WriteLine(MyGlobals.verificationCheck);
                    if (msgWS.IndexOf("->") > 0)
                    { break; }
                }
            }
            catch (IOException e)
            {
                // kobling antakeligvis brutt fra andre siden...
            }
            catch (Exception e)
            {
                // Alle andre feil
            }
        }
    }
}
```

Figur 11: tfunc metode

I denne metoden prøver vi å skrive til våpenstasjonen hvis kommunikasjonen er oppe.

```
public void SendToWS(string msg, bool veri)
{
    if (veri == true)
    {
        MyGlobals.verificationCheck = true;
    }
    byte[] DataToWS;
    form.writeToLog("Out: " + msg);
    DataToWS = Encoding.Default.GetBytes(msg);

    try
    {
        if (WS_tcpclient.Connected == true)
        {
            WS_tcpclient.GetStream().Write(DataToWS, 0, DataToWS.Length);
        }
    }
    catch (Exception e) { }
}
```

Figur 12: SendToWS metode

5.2 Systemkontroller

Mutual exclusion

Mutual exclusion er ett program objekt som forhindrer at to tråder prøver å bruke samme ressurs. Bare en tråd eier mutex-en om gangen. Når en tråd bruker en ressurs må den sperre mutex-en slik at ingen andre tråder kan bruke den samme ressursen. Når tråden er ferdig å bruke ressursen åpner den mutex-en igjen for at den neste tråden kan bruke den ønskede ressursen.

Delegate

Delegate bruker man for å referere til metoder, og bruke dem slik at man kan kalle på dem mellom tråder med hjelp av "invoke" funksjonen.

5.2.1 Kontroller

I ACT programvaren bruker vi mutex for å pause en tråd til en annen tråd er ferdig å kjøre. Metoden som er illustrert på figur 13 kjøres i en egen tråd som eier mutex-en helt til "WaitOne()" funksjonen kjører og tråden venter til den blir satt i gang igjen.

Dette er hovedtråden når man starter en test og kjører gjennom testen sekvensielt. For at programmet skal vite hvilke kommando den skal kjøre henter den ut informasjon fra test-tekvens arrayet (se Teknologidokument – XML for mer informasjon[2]) og lagrer denne stringen.

```
for (int i = 0; i < MyGlobals.sequenceCount; i++)
{
    int helpSecondDimension = MyGlobals.sequence[i].Length;
    for (int j = 0; j < helpSecondDimension; j++)
    {
        help = MyGlobals.sequence[i][j];
        MyGlobals.tRobot = new Thread(betweenTest);
        MyGlobals.tRobot.Start();
        autoEvent.WaitOne();
    }
    MyGlobals.writeToDatabaseThread = new Thread(writeToDatabase);
    MyGlobals.writeToDatabaseThread.Start();
    autoEvent.WaitOne();
}
```

Figur 13: Illustrasjon av kontroller

Når programmet skal kjøre en kommando trenger den en string som inneholder hvilke kommando den skal kjøre. Når denne kommandoen er ferdig vil mutex-en åpne seg igjen og hovedtråden vil kunne forsette med å sette i gang neste kommando.

```
public void betweenTest()
{
    Type type = typeof(CommandsFunctions);
    MethodInfo method = type.GetMethod(help);
    string result = (string)method.Invoke(methods, null);
    Console.WriteLine(result);

    autoEvent.Set();
}
```

Figur 14: Metoden som setter i gang en kommando

Her er et eksempel på en kommando som programmet kan kjøre. I denne metoden skal programmet sende en string til roboten som gjør at den skrur på våpenstasjonen.

Kan finne en fullstendig liste over hvilke metoder man kan bruke i Teknologidokument – XML[2].

Alt som skjer i hver kommando vil loggføres i en egen fil. For eksempel hvis roboten har flere bevegelser i en kommando med verifikasjon mellom hver bevegelse vil det først loggføres at roboten har beveget seg og hva den har gjort, så vil verifikasjonen loggføres, så vil robotbevegelsen loggføres igjen osv.

```
//Robot turns the WeaponStation ON
public void dcp_press_on()
{
    string Starting = "Starting SubP_on_off";
    string robotCommand = "ON/OFF";
    string loggRobotDone = "Robot has pressed the on/off button to turn the system on";
    string loggRobotNotDone = "Robot didn't press the on/off button to turn the system on";
    int sleep = 120000;

    rMethods.startRobotMovement(Starting, robotCommand, loggRobotDone, loggRobotNotDone, sleep);
}
```

Figur 15: Eksempel på en kommando-metode

Når en sekvens er ferdig og har verifisert om sekvensen ble godkjent eller ikke godkjent vil dette settes inn i databasen.

```
//Writes a row in the table in our database
public void writeToDatabase()
{
    sqlMethods.insertContent(MyGlobals.onSequence, MyGlobals.sequenceResult);
    autoEvent.Set();
}
```

Figur 16: Metoden som skriver til databasen

Når hele testen er ferdig vil det vises et nytt vindu med informasjon om hvordan testen har gått. Siden denne metoden bruker informasjon fra en annen tråd måtte vi bruke delegate.

```
//Opens a new window with the test results
public void showResultForm()
{
    form.Invoke((MethodInvoker)delegate()
    {
        testFinished.Show();
        testFinished.saveTestResultReport();
    });
    autoEvent.Set();
}
```

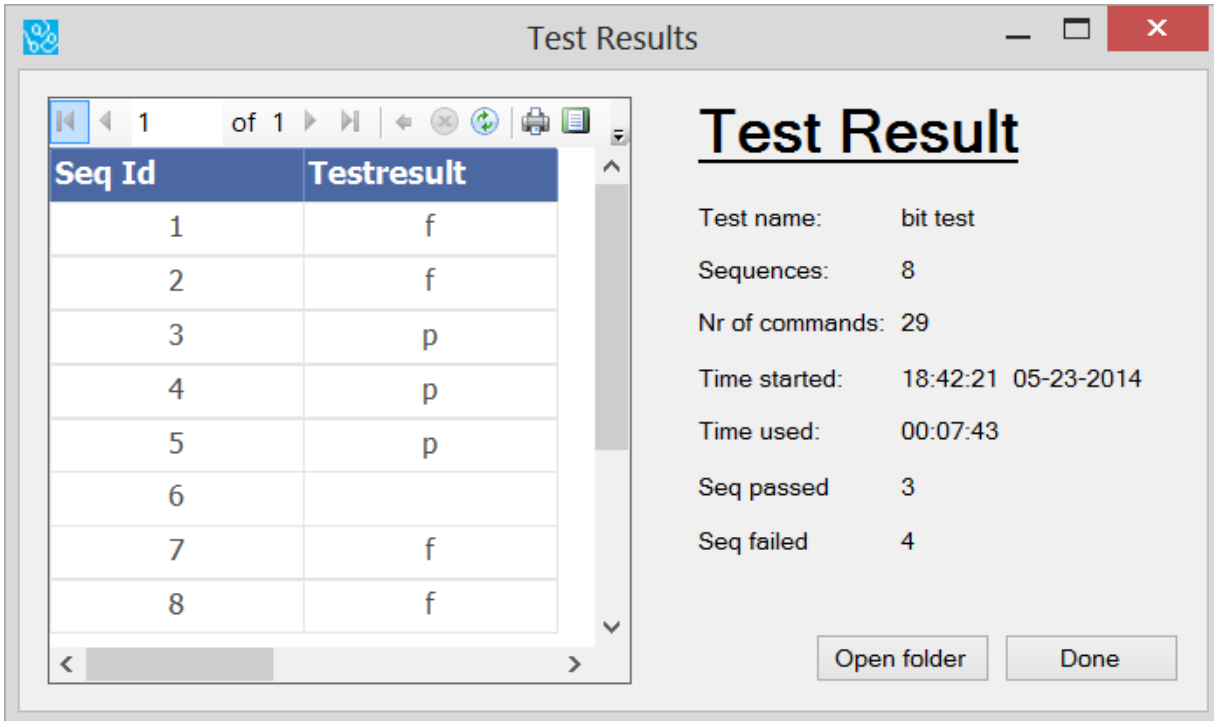
Figur 17: Metoden som viser test resultatene

Under ser du et eksempel på hvordan "Test Result" vindu ser ut. Til høyre på bilde ser man resultatet av hver sekvens i testen. Hvis det står en "f" i kolonne "Testresult" vil det si at sekvensen feilet og står det en "p" vil det si at sekvensen er godkjent. Hvis det er en tom rute i tabellen vil det si at ACT programvaren ikke er kompatibel med denne sekvensen og da må gjøres manuelt.

Til høyre på bildet ser man diverse informasjon på hvordan testen har gått og hva den inneholder. Slik som navnet på testen, hvor mange sekvenser det var, totalt antall kommandoer, når testen startet, hvor lang tid den har brukt, hvor mange sekvenser som ble godkjent og hvor mange som feilet.

Trykker man på "Open folder" åpnes mappen med alle filene som tilhører den testen seg i Windows Explorer. Denne mappen inneholder ****

Test Result: Som inneholder den samme informasjonen som er vist til høyre på bilde nedenfor.
Test Report: Som inneholder tabellen med oversikt over hvilke sekvenser som er godkjent eller ikke som er illustrert til venstre på bilde nedenfor.
Logg: Som inneholder all loggen som er gjennomført i testen, slik som robotbevegelser som er gjennomført og all verifisering som er gjort.



Figur 18: Eksempel på hvordan "Test Result" vindu kan se ut

5.3 Automatisk testrapport

Som nevnt tidligere bruker vi en Microsoft SQL server som er nødvendig for at ACT programvaren skal kunne kjøre. Det finnes en guide på hvordan denne blir installert og hvordan man skal legge inn databasen man trenger i Installeringsguiden[1]. Etter man har installert serveren og lagt inn databasen trenger man ikke tenke på den lenger. Den vil kjøre i bakgrunnen uten at du trenger å gjøre noe som helst.

Hva brukes databasen til i ACT programvaren?

Som beskrevet i Teknologidokumentet – XML[2] inneholder en test flere sekvenser som igjen inneholder flere kommandoer. Når man starter en test vil innholdet i tabellen (i databasen) bli slettet.

```
//Deletes the content of the table inside our database
public void deleteTableContent()
{
    string connectionString = "Data Source=." + @"\" + "SQLExpress;Initial Catalog=ACT_db;Integrated Security=True";
    SqlConnection connection = new SqlConnection(connectionString);
    connection.Open();
    string slett = @"DELETE FROM Testresults";
    SqlCommand command = new SqlCommand(slett, connection);
    command.ExecuteNonQuery();
    connection.Close();
}
```

Figur 19:Metoden som sletter alt innhold i tabellen i databasen

Når programmet er på en bestemt sekvens har denne sekvensen diverse verifikasjonskriterier som må være oppfylt for at sekvensen enten blir oppfylt eller ikke oppfylt. For hver kommando som skal verifisere noe, enten det er å sjekke at palm er aktivert eller våpenstasjonen fyrer, er dette en slags mellomverifisering. Når en kommando er ferdig vil “OK” eller “Not OK” bli fylt inn i et array. Når hele sekvensen er ferdig sjekkes det om alle mellomverifiseringene er “OK”. Hvis alle er “OK” vil det si at sekvensen er godkjent og en “p” vil bli skrivd til databasen. Hvis arrayet inneholder en eller flere “Not OK” vil ikke sekvensen bli godkjent og det blir sendt en “f” til databasen.

For at rapporten skal stemme overrens med testrapporten som KPS bruker har vi valgt å sette inn en funksjon som vil sende inn en rad til databasen med hvilken sekvens nummer som ikke ACT programvaren kan gjennomføre og derfor sette “Testresultat” blank.

```
//Inserts a row in the table in our database
public void insertContent(int x, bool result)
{
    string connectionString = "Data Source=." + @"\" + "SQLExpress;Initial Catalog=ACT_db;Integrated Security=True";

    SqlConnection connection = new SqlConnection(connectionString);
    connection.Open();

    if (result == true)
    {
        string insert = @"INSERT INTO Testresults (id, Testresult) VALUES (" + x + ", 'p')";
        SqlCommand command = new SqlCommand(insert, connection);
        command.ExecuteNonQuery();
        MyGlobals.seqPassed++;
    }
    else if(MyGlobals.sequenceIsEmpty == true)
    {
        string insert = @"INSERT INTO Testresults (id) VALUES (" + x + ")";
        SqlCommand command = new SqlCommand(insert, connection);
        command.ExecuteNonQuery();
    }
    else
    {
        string insert = @"INSERT INTO Testresults (id, Testresult) VALUES (" + x + ", 'f')";
        SqlCommand command = new SqlCommand(insert, connection);
        command.ExecuteNonQuery();
        MyGlobals.seqFailed++;
    }
    MyGlobals.sequenceIsEmpty = false;
    MyGlobals.sequenceResult = false;
    connection.Close();
}
```

Figur 20:Metoden som setter inn resultatet av en sekvens i tabellen i databasen

6 Konklusjon

Meningen med dette prosjektet var å lage et godt rammeverk som skulle gi et inntrykk av hva det potensielt kan bli. Vi føler sluttproduktet vårt er akkurat dette.

Vi føler programmet dekker de kravene KPS har satt med visse unntak (se testrapport[3]) for vår bacheloroppgave. Vi fikk også satt inn noen ekstrafunksjoner som vi syntes var passende å ha i programmet. Slik som en pause, stopp og reset knapp. I tillegg har vi også satt inn diverse hjelp funksjoner hvor du kan finne nyttig informasjon om de forskjellige egenskapene i programmet.

Vi har lært mye under utviklingen av dette programmet og har laget en rekke planer for videre utvikling for å gjøre programmet mer troverdig.

7 Referanser

REFERANSE	DOKUMENTTITTEL	VERSJON
[1]	Installeringsguide	1.0
[2]	Teknologidokument - XML	1.0
[3]	Testrapport	1.0

Tabell 3: Referanser

ACT AUTOMATED CROWS TESTING

TEK. DOK - VERIFIKASJON

AUGUST KIND SVENDSEN
EIRIK LIEN ROA
HENRIK BERGE SØRUM
STÅLE RUDIN

KONGSBERG

HBV
HØGSKOLEN I BUSKERUD OG VESTFOLD · PROFESJONSHØGSKOLEN ·

Teknologidokument - Verifikasjon

PROSJEKT	ACT - Automated CROWS Testing		
OPPDRAGSGIVER	Kongsberg Protech Systems		
UTFØRT VED	HBV - Høgskolen i Buskerud og Vestfold		
GRUPPE	August Kind Svendsen, Eirik Lien Roa, Henrik Berge Sørum og Ståle Rudin		
VERSJON	1.0		
ANTALL SIDER	28		
DOKUMENTHISTORIE	VERSJON	UTGITT	BESKRIVELSE
	1.0	26.05.2014	Første utgivelse

INNHOLDSFORTEGNELSE

LISTE OVER TABELLER

LISTE OVER FIGURER

1 Om dokumentet

1.1 Dokumenthistorie

VERSJON	DATO	ENDRING	SIGNATUR
0.1	18.05.14	• Dokumentet ble opprettet	ELR
1.0	26.05.14	• Første utgivelse	ELR

Tabell 1: Dokumenthistorie

1.2 Definisjoner og forkortelser

Liste som beskriver ord og uttrykk som kan være brukt i dette dokumentet:

UTTRYKK	FORKLARING
HBV	Høgskolen i Buskerud og Vestfold
KPS	Kongsberg Protech Systems (oppdragsgiver)
ACT	Automated CROWS Testing
Industrirobot	Universal Robots 5
CROWS	Common Remotely Operated Weapon Station
RWS	Remote Weapon Station
VS	Våpenstasjon
DCP	Display Control Panel
MPU	Main Processing Unit
Konfigurasjon	Forskjellige CROWS oppsett
Firing Controller	Eksternt instrument som måler fyringsspenningen
CG	Control Grip
Struct	En struct er en verditype som er typisk brukt til å kapsulere små grupper med relaterte data
Croppe	Beskjære bilder

Tabell 2: Definisjoner og forkortelser

2 Innledning

I kravspesifikasjonen[1] finner vi rammekravet RK.4 som sier at vi skal loggføre alle resultater og sekvenser, og funksjonskrav FK.2 som skal kunne verifisere CROWS software krav. Dette inneholder under krav om at software skal kunne verifisere resultater ved hjelp av tilstander på våpenstasjonen og bildeverifisering av bilde som kommer på DCP-en. Dette er kanskje den viktigste delen for KPS, for det som er hele poenget med en test av software er nettopp det å verifisere at softwaren fungerer som den skal. Det er derfor viktig at vi klarer å verifisere alle software-kravene til KPS på en god måte, og loggfører resultatene.

For å løse dette har vi tatt i bruk bildeverifisering i form av å sjekke om pikslene i bildene hentet ut fra DCP og bildene som skal bli verifisert mot er identiske. Vi har også brukt tilstandene fra våpenstasjonen til å verifisere. Dette gjør vi ved å sjekke tilstandene vi henter fra våpenstasjonen opp mot software kravene.

Vi finner også et rammekrav RK.2, og et sikkerhetskrav SK.1 som sier at softwaren vår skal kunne håndtere fatale feil på en forutsigbar måte og avbryte ved fatale feil. Med det menes at vi må kunne detektere om CROWS systemet til KPS gjør feil i forhold til hva som er ønsket. For eksempel kan dette være at den beveger eller skyter når den ikke skal.

3 Sammendrag

Vi har brukt bildeverifisering, tilstander og signaler fra våpenstasjonen til å verifisere resultater av sekvenser og tester utført av roboten og programmet vårt. Verifikasjonen er utviklet med tanke på å verifisere KPS sine software-krav. Kravene deres blir testet ved hjelp av en testprosedyre som vi har valgt å lage programmet vårt etter. Her skal verifikasjons delen verifisere resultatene i henhold til denne testprosedyren.

Siden så å si alle kravene går ut på å lese tilstander fra våpenstasjonen så er dette den viktigste formen for verifikasjon. Denne er basert på en parser som henter ut informasjon om tilstander fra våpenstasjonen og setter informasjonen klart for verifisering. Deretter er det diverse metoder som verifiserer denne dataen og godkjenner eller ikke godkjenner verifikasjonen. Alle resultater blir loggført til en egen fil.

I testene av kravene til KPS så er det veldig mye som går ut på å visuelt se etter noe på skjermen. Dette har vi da valgt å løse ved hjelp av bildegjenkjenning. Her tar vi da ut et bilde fra våpenstasjonen under kjøring, og verifiserer dette opp mot et bilde med riktig resultat. Deretter sammenligner vi pikslene i bildene og hvis disse er identiske godkjennes verifikasjonen. Vi har valgt og bare ta ut overlayet av bilde så vi slipper å tenke på hva som er på videostrømmen fra kameraet på våpenstasjonen. Overlay vil si det som blir printet på videostrømmen.

Når bilde og informasjon fra våpenstasjonen er verifisert og sammenlignet så sjekkes resultatene ved hjelp av et array av resultater. Hvis alle er godkjent så godkjennes sekvensen og det blir sendt til rapporten. Dersom noe feiler blir sekvensen ikke godkjent og rapporten rapporterer feil. Feilhåndteringen gikk ikke helt som planlagt. Vi har implementert for sjekk av uønsket bevegelse og sjekk av fatale feil, men det er ikke i bruk per dags dato.

4 Verifisering av KPS software-krav

Når KPS utfører funksjonstester er det for å verifisere at software-kravene deres blir verifisert. Dette gjøres manuelt av en person som sitter og følger en testprosedyre med forskjellige tester som verifiserer kravene. Hvis KPS skal ta i bruk automatiske tester er det derfor viktig å ha en god verifiseringsmetode, som kan gjøre en like god eller bedre jobb enn en person som sitter og gjøre dette.

Verifiseringen av KPS sine software-krav fungerer ved at en tester følger en bestemt testsekvens i form av forskjellige kommandoer. Dette vil vi automatisere ved at vi først verifiserer kommandoene og deretter verifiserer selve testsekvensen. Hvis en testsekvens viser seg å bli verifisert vil vi sende dette til en testrapport som verifiserer at testsekvensen er blitt godkjent og utført uten feil.

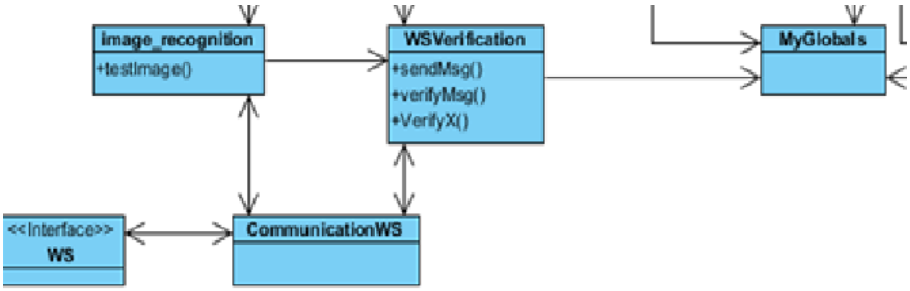
1.	System operational. Verify "SYSTEM"-switch in "ARM" Press PALM switch and FIRE trigger.	"ARMED" displayed in the Status field. "Firing Enabled" displayed in the Status field, the"ARMED" LED is lit and the solenoid clicks when palm and fire switch is activated.	
2.	Type the following message in the test terminal: `->ts FireCircuit` This command will stop the Fire Diagnostic Checking	"System Failure" is displayed in the Status field and the following FATAL message is displayed: 0706: FIRING DISABLED: System error SCS-3 – (Inadvertent Weapon Fire, SRS13) SCS-3 – (Loss of operator info, SRS10, SRS137)	
3.	Set System switch in "Safe" position. Press PALM Switch Toggle the "Safety Override" switch several times to activate a Warm Reset.	A Warm Reset is NOT initiated. The original Error message is still displayed on the FCU. A warning is displayed: 0751: WARM RESET UNAVAILABLE: Recycle power	

Figur 1: Testprosedyre

Her ser vi et eksempel på et par testsekvenser fra KPS sin testprosedyre. Helt til venstre ser man sekvensnummeret, til høyre for det igjen er kommandoene man skal gjøre på systemet, til høyre for det igjen er resultatet og det som skal verifiseres at skal skje og tilslutt skal man markere godkjent/ikke godkjent. Kommandoene her er det typisk roboten eller PC-en som gjør også er det resultat delen som er verifiseringen av test sekvensen.

Ta for eksempel sekvens 1. Der skal man skru på systemet, sette systemet i arm og prøve å skyte. Her skal da verifiseringen verifisere at det står «armed» på skjermen ved hjelp av bildeverifisering og tilstand fra våpenstasjonen, deretter skal man sjekke om man får fyrt og at «firing enabled» står på skjermen, dette må da bli gjort via tilstander og bildeverifisering av skjermen. På sekvens 2 så skal man sende inn en feilmelding og verifiseringen skal verifisere både at feilmeldingen står på skjermen, at den er riktig og sjekke at systemet har den registrert ved hjelp av å lese ut feilmeldingsregisteret fra våpenstasjonen. Når man har verifisert hver kommando så skal man sjekke om sekvensen er godkjent og skrive det til testrapporten.

4.1 Design av verifiseringsdelen



Figur 2: Utrdag av klassediagram

Her ser man et utdrag av klassediagrammet vårt som viser hvordan verifikasjonen er implementert i programmet vårt. Vi har to klasser som tar for seg verifikasjonen av resultatene fra testene. Den ene er Image_recognition som tar for seg bildeverifiseringen. Den andre klassen er WSVerification som sjekker systemtilstander, feilmeldingsregistre og bevegelser fra våpenstasjonen. Den inneholder også metoder for å verifisere en hel sekvens og metode for å verifisere om både bilde og tilstanden fra våpenstasjonen stemte.

Vi har en communication klasse som er veldig vesentlig for verifikasjonen for her går all kommunikasjonen mellom programmet vårt og våpenstasjon systemet. Den blir brukt til å hente ut tilstander og bilde fra våpenstasjonen.

En annen viktig klasse for verifikasjonen er «My globals». Her er alle de globale variablene deklarert og her har vi blant annet verifikasjon variabler som blir satt til OK eller NOT OK og blir sendt videre til for å bli verifisert. Vi har også en viktig variabel her som godkjenner eller ikke godkjenner en sekvens og sendes videre til testrapporten for å fylle ut om den er godkjent eller ikke.

Til slutt er det en annen viktig klasse som ikke er vist på bilde som er «command functions». Den hører ikke direkte til verifikasjon delen men det er her verifikasjon metodene blir kalt.

5 Verifisering med tilstander og signaler hentet fra WS

Dette er hovedverifiseringen i programmet. Denne verifiserer alt som våpenstasjonen kan returnere, dette innebærer tilstander, signaler og feilmeldingsregistre. En tilstand kan for eksempel være om systemet står i «armed» eller «safe», eller om systemet skyter eller ikke. Dette bruker vi da i form av at vi følger testsekvensene til KPS og verifiserer de forskjellige tilstandene etter hva som forventes av resultat i hver sekvens. En sekvens inneholder ofte flere ting som skal verifiseres så da skjer dette sekvensielt, altså at man gjør det stegvis ved å verifisere en og en kommando/resultat.

5.1 Hvordan fungerer det?

Dette fungerer ved at man har en metode som kjøres når programmet mottar data fra våpenstasjonen. Denne metoden kjører dataen som blir mottatt gjennom en parser som består forskjellige «regular expressions». Denne parseren sjekker etter data man er ute etter, og får man treff så setter man det resultatet man har truffet inn i enten en variabel eller et array avhengig av hvordan resultat fra våpenstasjonen man er ute etter.

```
//Parser for verification of data recieved from the weaponstation.
public void Testing(string DataRecieved)
{

    bool MatchHit;
    string RecieveInput = DataRecieved;
    //checks for error id's and stores them in a list array.
    if (RecieveInput != null)
    {
        foreach (Match match in Regex.Matches(RecieveInput, "\\b(errorId = [0-9]*)(, )([0-9]*)",
                    RegexOptions.IgnoreCase))
        {

            string testMatches = match.ToString();
            MessageList.Add(testMatches);
        }
```

Her ser man litt av koden til denne parseren som sjekker tilstander fra våpenstasjonen. Her ser man uttrykket for å finne error ID-er fra feilmeldings registeret. I dette registeret kan det ligge flere error ID-er så her blir de lagret i et liste array.

Det som skjer etter dataen fra våpenstasjonen er parset så blir resultatet fra parsingen lagret og verifikasjonsmetodene kan da bruke denne dataen til å verifisere og sjekke opp mot det forventede resultatet.

```
//Verifies error id's
if (MessageList.Contains(verificationStruct[MyGlobals.onMessage].errorID01))
{

    Console.WriteLine("OKEY");
    MyGlobals.WsMessageOk = "Ok";
    MessageList.Clear();


}
else
{
    MyGlobals.WsMessageOk = "Not Ok";
    Console.WriteLine("Message Not Ok");

    MessageList.Clear();
}
```

Her ser man et eksempel på en verifikasjon der man bruker liste-arrayet fra sist bilde til å verifisere opp mot et struct med riktige error ID-er. Det som skjer her er at hvis liste-arrayet inneholder den riktige error ID-en, så settes en global verifikasjonsvariabel (Myglobals.WsMessageOk) til «Ok» og

error ID-en er verifisert. Hvis ikke arrayet inneholder riktig error id så feiler verifikasjonen og variabelen settes til «Not Ok». Dette er et eksempel på en verifikasjon og hvordan det fungerer.

5.1.1 Parsing

Parsing vil si at man tar ut det man er ute etter fra en datastrøm. Vi bruker TCP/IP kommunikasjon som ligger lavt i nettverksmodellen, så her blir det sendt mye rå data mellom våpenstasjonen og programmet vårt. Vi må dermed vite hva vi skal hente ut av datastrømmen og deretter ta dette ut ved hjelp av parsing.

Grunnen til at vi må parse meldinger fra våpenstasjonen er at når den returnerer tilstander så kommer det ikke tilbake kun det man er ute etter, men vi får en hel datastrøm. Vi får dermed mye data vi ikke er ute etter. For at man skal være sikker på at man får tak i det man er ute etter velger vi da å bruke parsing til dette. For eksempel kan det hende vi er ute etter tilstanden til «armed» ut av masse unyttig data.

Metoden vi bruker til og parse kalles «regular expressions». Dette fungerer ved at man skriver et uttrykk for den dataen man er ute etter. Måten vi gjør dette på er at vi bruker en «foreach løkke» som løper gjennom datastrømmen fra våpenstasjonen og sjekker opp mot utrykket med data vi har sagt vi er ute etter. Under kan man se et «regular expression» hentet fra koden vår.

```
"\\b(errorId = [0-9]*)(, )([0-9]*)",
```

5.2 Hvilke funksjoner blir verifisert?

Vi har forskjellige funksjoner som skal bli verifisert ved hjelp av denne metoden. Felles for disse er at alle resultater vil bli skrevet til en logg fil som lagres til disken. Disse verifiseringene vil skrive til loggen om de mislykkes eller godkjennes.

Funksjonene har to forskjellige måter å bli godkjent på de med bildeverifisering i tillegg blir godkjent på denne måten:

Dersom verifiseringen blir godkjent så settes verifiseringsvariabelen (Myglobals.WsMessageOk) til «OK» hvis verifiseringen ikke blir godkjent settes variabelen til «Not Ok». Dette gjelder firing enabled, armed, warm reset, og feilmeldinger.

De resterende funksjonene vil bli verifisert kun av tilstander fra våpenstasjonen, så når disse blir godkjent setter man inn en OK i sekvensresultat-arrayet som sier at kommandoen er OK.

5.2.1 Feilmeldinger

Denne verifikasjonen brukes til å verifisere at riktig feilmelding registreres av våpenstasjonen. Dette sjekkes ut ifra et feilmeldingsregister der man sjekker etter riktig error id og sammenligner det man får fra våpenstasjonen og det kravet sier er riktig.

5.2.1.1 Parser

Parseren for denne verifikasjonen ser slik ut:

```
//Parser for verification of data recieved from the weaponstation.
public void Testing(string DataRecieved)
{

    bool MatchHit;
    string RecieveInput = DataRecieved;
    //checks for error id's and stores them in a list array.
    if (RecieveInput != null)
    {
        foreach (Match match in Regex.Matches(RecieveInput, "\\b(errorId = [0-9]*)(, )([0-9]*)",
                    RegexOptions.IgnoreCase))
        {

            string testMatches = match.ToString();
            MessageList.Add(testMatches);
        }
```

Den er vist som eksempel tidligere, men den fungerer ved at den sjekker etter error ID-er og setter disse inn i et liste-array.

5.2.1.2 Verifisering

Verifiseringen av denne fungerer ved at man loader et struct med riktige error ID-er som man kan verifisere opp i mot. Metoden henter inn riktig feilmelding fra structet basert på hvordan sekvens testen er på. Under ser man verifiseringen:

```
//Verifies error id's
if (MessageList.Contains(verificationStruct[MyGlobals.onMessage].errorID01))
{

    Console.WriteLine("OKEY");
    MyGlobals.WsMessageOk = "Ok";
    MessageList.Clear();


}
else
{
    MyGlobals.WsMessageOk = "Not Ok";
    Console.WriteLine("Message Not Ok");

    MessageList.Clear();
}
```

Deretter tar den liste-arrayet og sjekker om det inneholder den riktige error ID-en som er hentet fra structet.

5.2.2 Armed

Denne verifikasjonen sjekker om systemet står i safe eller i armed. Dette er simpelt en bryter som kan vippes til armed eller safe. For å sjekke og verifisere dette leser vi av tilstanden på denne som kan være 0 = safe eller 1= armed. Deretter verifiserer vi dette.

5.2.2.1 Parser

Denne parseren leser ut tilstanden til system switchen som kan enten være 0 eller 1

```
//Checks if the system is armed(1) or in safe (0).
foreach (Match match in Regex.Matches(RecieveInput, "\\b(Safe_Switch_Enabled : [0-1])",
                RegexOptions.IgnoreCase))
{

    string testMatches = match.ToString();

    MatchHit = match.Success;

    if (testMatches.Equals("Safe_Switch_Enabled : 1"))
    {

        ArmedSwitch = true;
    }
    if (testMatches.Equals("Safe_Switch_Enabled : 0"))
    {

        ArmedSwitch = false;
    }
}
```

.

Her ser man at hvis system switchen er 0 så setter vi ArmedSwitch variabelen til false, og hvis den er 1 så setter vi den til true.

5.2.2.2 Verifisering

Denne verifiseringen sjekker tilstanden på system switchen. Og vil stå i 1 dersom den er armed.

```
if (ArmedSwitch == true)
{
    MyGlobals.SystemArmed = "OK";
    Console.WriteLine("System Armed, test ok");
    string ArmswitchSafe = "System is in armed, Test OK";
    DateTime time = DateTime.Now;
    using (System.IO.StreamWriter file = new System.IO.StreamWriter(MyGlobals.appPath + @"\" + fileName, true))
    {
        file.WriteLine(time.ToString(format));
        file.WriteLine(ArmswitchSafe);
    }
    MyGlobals.WsMessageOk= "Ok";

    txtDataBox.DisplaytxtData("ArmedWS OK" + "\n");
}
else
{
    MyGlobals.WsMessageOk = "Not Ok";

    Console.WriteLine("System in safe test not ok");
    string WSArmedSwtich = "System in safe, test not ok.";
    DateTime time = DateTime.Now;
    using (System.IO.StreamWriter file = new System.IO.StreamWriter(MyGlobals.appPath + @"\" + fileName, true))
    {
        file.WriteLine(time.ToString(format));
        file.WriteLine(WSArmedSwtich);
    }
    txtDataBox.DisplaytxtData("System in safe not ok" + "\n");
}
```

Denne sjekker om ArmedSwitch variabelen er true. Hvis den er det så verifiserer den at den er i armed. Hvis den er true så vil den bli satt til godkjent hvis den er false så vil den bli satt til ikke godkjent.

5.2.3 Firing enabled

Denne verifiseringen sjekker om systemet er klar for å fyre. Det vil si at systemet står i armed og at palm switchen er trykt inn. Denne tilstanden kan leses ut i fra våpenstasjonen og kan enten være 1 som er at den er klar for fyring eller 0 som er at den ikke er klar for fyring. Deretter verifiserer vi dette.

5.2.3.1 Parser

Denne parseren leser ut tilstanden til om systemet er i firing enabled eller ikke. som kan enten være 0 eller 1.

```
//Checks if firing is enabled (1), or not (0).
foreach (Match match in Regex.Matches(RecieveInput, "\\b(Firing_Enabled: [0-1])",
                  RegexOptions.IgnoreCase))
{
    string testMatches = match.ToString();

    MatchHit = match.Success;

    if (testMatches.Equals("Firing_Enabled: 1"))
    {
        FiringEnabled = true;

    }
    if (testMatches.Equals("Firing_Enabled: 0"))
    {
        FiringEnabled = false;
    }

}
```

Her ser vi at dersom firing er enabled så setter vi FiringEnabled variabelen til true og false om den ikke er i firing enabled.

5.2.3.2 Verifisering

Her verifiserer vi Firing Enabled.

```
public void VerifyFiringEnabled()
{
    txtDataBox.DisplaytxtData("Verifies firing enabled" + "\n");
    Thread.Sleep(1000);
    CommunicationWS_SendData.CommunicationWS_Send("firestateShow" + "\n", true);
    Thread.Sleep(2000);

    if (FiringEnabled == true)
    {

        MyGlobals.FiringEnble = "OK";
        Console.WriteLine("firing is enabled, test ok");
        string FireEnable = "Firing is enabled, Test OK";
        DateTime time = DateTime.Now;
        using (System.IO.StreamWriter file = new System.IO.StreamWriter(MyGlobals.appPath + @"\" + fileName, true))
        {
            file.WriteLine(time.ToString(format));
            file.WriteLine(FireEnable);
        }
        MyGlobals.WsMessageOk = "Ok";
        txtDataBox.DisplaytxtData("Firing is enabled, OK" + "\n");
    }
    else
    {
        MyGlobals.WsMessageOk="Not Ok";
        Console.WriteLine("firing is not enabled, test not ok.");
        string FiringEnable = "Firing not enabled, Test not OK";
        DateTime time = DateTime.Now;
        using (System.IO.StreamWriter file = new System.IO.StreamWriter(MyGlobals.appPath + @"\" + fileName, true))
        {
            file.WriteLine(time.ToString(format));
            file.WriteLine(FiringEnable);
        }
```

Her sjekker vi FiringEnabled variabelen og ser om den er true eller false. True er godkjent, false er ikke godkjent.

5.2.4 «Palm switch»

Denne verifiseringen verifiserer om palm switchen er aktivert eller ikke. Det vil si om roboten holder inne palm switchen eller ikke. For å verifisere dette sjekker vi tilstanden som kan være 0 eller 1.

5.2.4.1 Parser

Denne parseren leser ut tilstanden til palm switchen, om den er aktivert eller ikke, som kan enten være 0 eller 1.

```
//Checks if the palmswitch is enabled(1) or not(0).

foreach (Match match in Regex.Matches(RecieveInput, "\\b(Palm_Switch_Enabled : [0-1])",
                RegexOptions.IgnoreCase))
{

    string testMatches = match.ToString();

    MatchHit = match.Success;

    if (testMatches.Equals("Palm_Switch_Enabled : 1"))
    {

        palmEnabled = true;
    }
    if (testMatches.Equals("Palm_Switch_Enabled : 0"))
    {

        palmEnabled = false;
    }
}
```

Her ser vi at dersom palm switchen er aktivert, så setter vi PalmSwitch variabelen til true og false om den ikke er i aktivert.

5.2.4.2 Verifisering

Her verifiserer vi palm switch.

```
if (palmEnabled == true)
{

    MyGlobals.PalmEnable = "OK";
    Console.WriteLine("Palm enabled, test ok");
    string PalmEnable = "Palm enabled, Test OK";
    DateTime time = DateTime.Now;
    using (System.IO.StreamWriter file = new System.IO.StreamWriter(MyGlobals.appPath + @"\" + fileName, true))
    {
        file.WriteLine(time.ToString(format));
        file.WriteLine(PalmEnable);
    }
    MyGlobals.CommandOK = "OK";
    SequenceResaultList.Add(MyGlobals.CommandOK);
    txtDataBox.DisplaytxtData("Palm enabled, OK" + "\n");
}
else
{
    MyGlobals.CommandOK = "Not OK";
    SequenceResaultList.Add(MyGlobals.CommandOK);
    Console.WriteLine("Palm not enabled, test not ok");
    string PalmEnable = "Palm not enabled, Test not OK";
    DateTime time = DateTime.Now;
    using (System.IO.StreamWriter file = new System.IO.StreamWriter(MyGlobals.appPath + @"\" + fileName, true))
    {
        file.WriteLine(time.ToString(format));
        file.WriteLine(PalmEnable);
    }
    txtDataBox.DisplaytxtData("Palm not enabled, not OK" + "\n");
}
```

Her sjekker vi PalmSwitch variabelen og ser om den er true eller false. True er godkjent, false er ikke godkjent.

5.2.5 Firing

Denne verifiseringen verifiserer om systemet skyter eller ikke. Det vil si at roboten holder inne palm switch og trigger samtidig som systemet står i armed. Her må vi sjekke tilstanden til triggeren som kan være aktivert (1) eller ikke aktivert (0). Deretter må man også sjekke firing enabled variabelen fra siste verifiseringsmetode.

5.2.5.1 Parser

Denne parseren leser ut tilstanden til gun trigger, om den er aktivert eller ikke, som kan enten være 0 eller 1.

```
//Checks if the system is firing (1), or not(0).
foreach (Match match in Regex.Matches(RecieveInput, "\\b(Gun_Trigger_On : [0-1])",
                RegexOptions.IgnoreCase))
{

    string testMatches = match.ToString();

    MatchHit = match.Success;

    if (testMatches.Equals("Gun_Trigger_On : 1"))
    {

        Gunfire = true;

    }
    if (testMatches.Equals("Gun_Trigger_On : 0"))
    {

        Gunfire = false;
    }
}


struct of Bit mesessages
```

Her ser vi at dersom gun trigger er aktivert, så setter vi GunTrigger variabelen til true og false om den ikke er i aktivert.

5.2.5.2 Verifisering

Her verifiserer vi firing.

```
if (Gunfire == true && FiringEnabled== true)
{
    MyGlobals.GunFireOK = "OK";
    Console.WriteLine("System is firing, test ok");
    string GunfireAllowed = "System is firing, Test OK";
    DateTime time = DateTime.Now;
    using (System.IO.StreamWriter file = new System.IO.StreamWriter(MyGlobals.appPath + @"\" + fileName, true))
    {
        file.WriteLine(time.ToString(format));
        file.WriteLine(GunfireAllowed);
    }
    MyGlobals.CommandOK = "OK";
    SequenceResaultList.Add(MyGlobals.CommandOK);
    txtDataBox.DisplaytxtData("System is firing, OK" + "\n");
}
else
{

    MyGlobals.CommandOK = "Not OK";
    SequenceResaultList.Add(MyGlobals.CommandOK);
    Console.WriteLine("System is not firing");
    string Firing = "System is not firing, Test not OK";
    DateTime time = DateTime.Now;
    using (System.IO.StreamWriter file = new System.IO.StreamWriter(MyGlobals.appPath + @"\" + fileName, true))
    {
        file.WriteLine(time.ToString(format));
        file.WriteLine(Firing);
    }
    txtDataBox.DisplaytxtData("System is not firing, not OK" + "\n");
}
```

Her sjekker vi gun trigger variabelen og ser om den er true eller false i tillegg må vi se at firing enabled variabelen er satt til true. Hvis begge er true så er verifikasjonen godkjent, dersom det er false på en av de er den ikke godkjent.

5.2.6 Bevegelse

Denne verifiseringen verifiserer bevegelse av våpenstasjonen. Den fungerer ved at man sjekker posisjon før og etter bevegelse og ser om den har forandret seg.

5.2.6.1 Parser

Denne fungerer ved at man leser ut elevasjon som a og azimuth som b og får dermed posisjonen til våpenstasjonen.

```
//Reads the two main axis; elevation and azimuth to check the position of the weaponsation.

int res = 0;
if (RecieveInput.Contains("Main Axis"))
{
    foreach (Match match in Regex.Matches(RecieveInput, "[+-]?(?=[0-6][.][0-9])[0-9]*(?:[.][0-9]*)?",
                RegexOptions.IgnoreCase))

    {

        string testMatches = match.ToString();

        if (res == 1)
        {
            b = Convert.ToDouble(testMatches, System.Globalization.CultureInfo.InvariantCulture);


        }
        if (res == 0)
        {
            a = Convert.ToDouble(testMatches, System.Globalization.CultureInfo.InvariantCulture);
            MatchHit = match.Success;
            res = 1;
        }
```

5.2.6.2 Verifisering

Her verifiserer vi at den ikke skal ha lov til å flytte på seg. Her sjekker vi om posisjonen til våpenstasjonen har forandret seg før og etter kjøring. Den vil bli verifisert godkjent dersom posisjonen er den samme.

```
if ((elevation < 0.000004) && (azimuth < 0.000004) && (elevation > -0.000004) && (azimuth > -0.000004))
{
    MyGlobals.MovementNotAllowed = "OK";
    Console.WriteLine("WS did not move, test OK");
    DateTime time = DateTime.Now;
    using (System.IO.StreamWriter file = new System.IO.StreamWriter(MyGlobals.appPath + @"\" + fileName, true))
    {
        file.WriteLine(time.ToString(format));
        file.WriteLine(MovementNotAllowed);
    }
    MyGlobals.CommandOK = "OK";
    SequenceResaultList.Add(MyGlobals.CommandOK);
    txtDataBox.DisplaytxtData("System is not moving, test OK" + "\n");
}
else
{

    MyGlobals.CommandOK = "Not OK";
    SequenceResaultList.Add(MyGlobals.CommandOK);
    Console.WriteLine("ws moved");
    string WsMovement = "WS moved, Test not OK";
    DateTime time = DateTime.Now;
    using (System.IO.StreamWriter file = new System.IO.StreamWriter(MyGlobals.appPath + @"\" + fileName, true))
    {
        file.WriteLine(time.ToString(format));
        file.WriteLine(WsMovement);
    }
    txtDataBox.DisplaytxtData("System is moving, test not OK" + "\n");
}
```

5.2.7 Warm reset

Denne metoden verifiseres på samme måte som feilmeldinger. Den eneste forskjellen er at man kun leter etter warm reset feilmeldingen, også verifiserer opp mot error ID-en dens.

Verifisere en kommando med bildegjenkjenning og tilstander fra våpenstasjonen

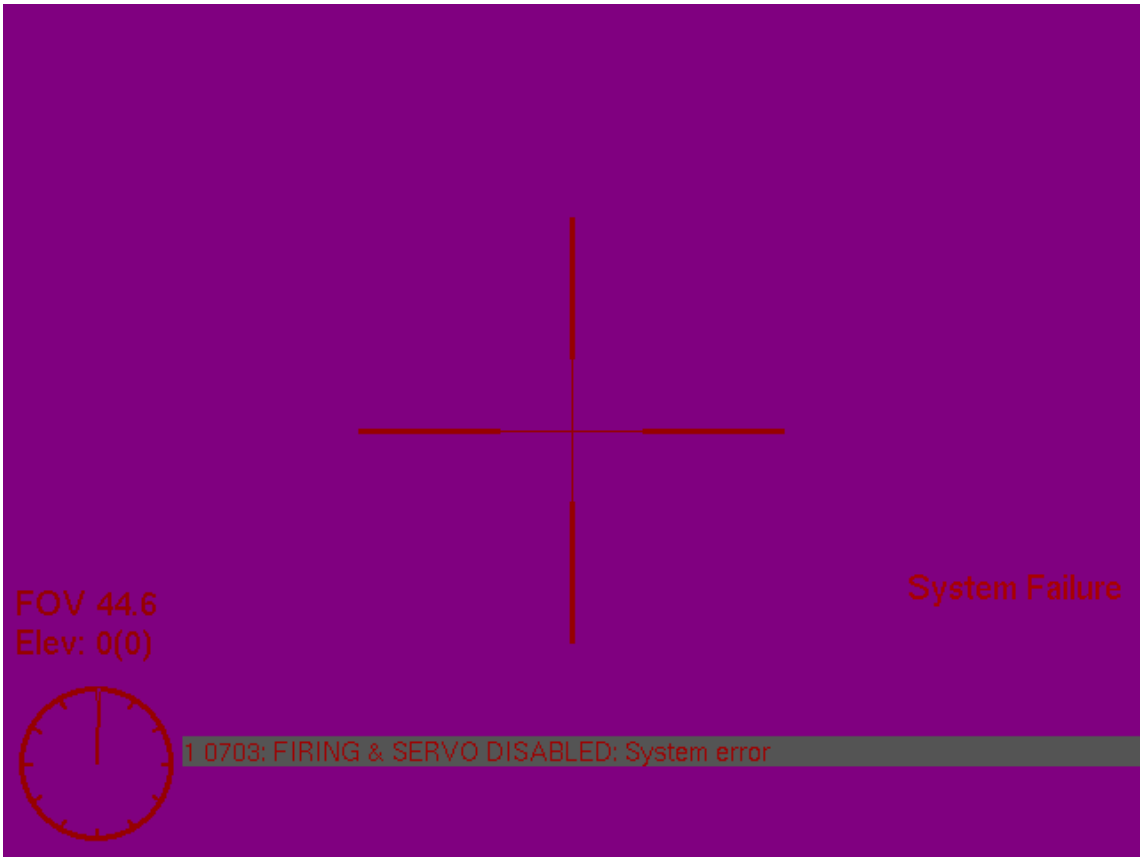
6 Bildeverifisering

Bildeverifisering er den andre måten vi verifiserer resultater på. Dette gjør vi fordi mange av testene til KPS går ut på å verifisere visuelt på skjermen. Vi må dermed hente ut et bilde fra våpenstasjonen under test og verifisere dette opp mot et riktig resultat hentet fra en fildestinasjon med verifikasjons bildene med forventet resultat. Fil destinasjonen skal være under Mydocuments og lagres i mappe «ACT_Bilder. Metoden vi bruker for å verifisere bilder er å sammenligne pikslene og sjekke at disse stemmer overens og er identiske. Deretter verifiserer vi resultatet.

Vi verifiserer 5 forskjellige funksjoner så langt. Det er at firing enabled står på skjermen, armed er vist på skjermen, at system failure kommer opp om det skal. Vi verifiserer at riktige feilmeldinger kommer og at warm reset meldingen kommer.

6.1 Overlay

Vi har valgt å ta ut overlayet av bilde. Det vil si at bilde vi tar ut fra våpenstasjonen bare inneholder det som blir printet av systemet på videostrømmen fra kameraet. Vi fant ut at dette er den beste og enkleste metoden fordi da slipper vi å tenke på bakgrunnsstøy og ta hensyn til videostrømmen. Et eksempel på overlay ser man under:



Figur 3: Bilde overlay fra DCP

6.2 Bilde hentet fra WS under test

For å verifisere bilde under test blir vi nødt til å hente dette ut fra våpenstasjonen under selve testen. Dette gjør vi ved å sende in en funksjon til våpenstasjonen som gjør at vi kan ta ut overlayet av displayet. Dette gjør vi ved å sende inn en fil som inneholder funksjoner for å ta ut overlay. Denne funksjonen blir sendt ved hver oppstart.

Når dette er gjort kan vi ta ut bilde og kalle det hva vi vil. Bilde blir da lagret i mappen vi har delt med FileZilla serveren, som skal ligge under MyDocuments og under mappe ACT_Bilder. Vi kan selv kalle bilde som blir tatt ut for det vi selv ønsker.

Eksempelkode vises under:

```
string ImageCaptured = MyGlobals.onMessage.ToString();
string ImageVerify = MyGlobals.onMessage + ".bmp";

LoadImages();

txtDataBox.DisplaytxtData("Verifies Image" + "\n");

string send = "capture \"" + ImageCaptured + "\"" + "\n";
txtDataBox.DisplaytxtData("sent in" +send);


CommunicationWS_SendData.CommunicationWS_Send(send,false);
```

Her sender man capture «pluss navn på bilde du ønsker å lagre det som» til våpenstasjonen som er metoden for å hente ut overlayet. Bilde vil dermed bli lagret i den delte FileZilla mappen, som skal ligge under my dokuments og hete ACT_Bilder.

6.3 Bilde som blir verifisert opp mot

Når vi har tatt ut bilde fra våpenstasjonen må vi sammenligne dette opp mot et bilde som har riktig resultat. Dette gjør vi ved at vi har disse filene lagret i en mappe på disken og setter disse bildene inn i et struct array som vi verifiserer opp mot. Funksjonen finner selv ut hvilket bilde som skal være til hver enkelt test ved hjelp av en variabel som holder styr på hvor du er.

Kode vises under:

```
public void LoadImages()
{
    string path = Environment.GetFolderPath(System.Environment.SpecialFolder.MyDocuments) + @"\ACT_bilder\Bildertilbittest";
    ImageVerification[0].Images = path + @"\0706.bmp";
    ImageVerification[1].Images = path + @"\0705.bmp";
    ImageVerification[2].Images = path + @"\0703.bmp";


}
```

Her ser man starten på et struct med bilder lagret til en destinasjon på PC. Så verifiserer man opp mot disse.

6.4 «Croppe» bildet

Når vi har begge bildene vi skal verifisere «cropper» vi bildene slik at vi bare sjekker den delen vi er ute etter. For eksempel om vi skal verifisere at det står «armed» på skjermen så har det en fast posisjon hele tida, så da kan vi «croppe» bildet til og kun verifisere den biten vi er ute etter.

Dette gjør vi med alle verifikasjonene.

Her henter vi bildet som skal verifiseres opp mot som riktig resultat(img1), deretter blir bildet fra WS hentet ut(img2).

Kode vises under:

```
// gets the verify image
Bitmap x = (Bitmap)Image.FromFile(ImageVerification[MyGlobals.onMessage].Images);
Bitmap x2 = x.Clone(new Rectangle(100, 400, 540, 80), x.PixelFormat);


img1 = x2;

// get image from ws
Bitmap x3 = (Bitmap)Image.FromFile(ImgPath + ImageVerify);
Bitmap x4 = x3.Clone(new Rectangle(100, 400, 540, 80), x.PixelFormat);


img2 = x4;
```

Her ser vi måten vi «cropper» bildene på. Vi har et «bitmap» som vi kloner og lager en ny rektangel av det bilde hvor vi styrer hvor denne rektangelet skal være ved hjelp av koordinater og størrelse. De to første er posisjon x og y og de to siste i parentesen er størrelse x og y.

6.5 Hvordan fungerer algoritmen?

Når vi så har begge bildene ferdig «croppet» og klare for verifisering trenger vi en algoritme som sammenligner alle pikslene i begge bildene. Dette kjører den med at den løper gjennom alle pikslene ved hjelp av «for løkker».

Kode vises under:

```
//Algorithm for pixel comparrison
if (img1.Width == img2.Width && img1.Height == img2.Height)
{

    for (int i = 0; i < img1.Width; i++)
    {
        for (int j = 0; j < img1.Height; j++)
        {

            img1_ref = img1.GetPixel(i, j).ToString();

            img2_ref = img2.GetPixel(i, j).ToString();

            if (img1_ref != img2_ref)
            {
                count2++;
                flag = false;
                break;

            }
            count1++;

        }
```

Her ser man at hvis "img 1" er ulik "img 2", så blir et flagg satt til false som blir sendt videre til verifiseringen og testen av bildene feiler. Hvis de løper gjennom uten å være ulike godkjennes testen.

6.6 Verifiseringen

Når algoritmen for å sjekke om pikslene er identiske er ferdig så sender den en beskjed til verifiseringen om de var like eller ikke. Dette gjøres ved hjelp av et flagg. Dersom de var like så settes en bilde verifiserings variabel til «Ok», dersom de var ulike så settes den til «Not Ok».

```
        if (flag == false)
        {
            Console.WriteLine("Sorry, Images are not same , " + count2 + " wrong pixels found");
            Console.WriteLine("NOT OK");

            MyGlobals.ImageOk = "Not Ok";
        }
        else
        {
            Console.WriteLine(" Images are same , " + count1 + " same pixels found  and " + count2 + " wrong pixels found");
            Console.WriteLine("OK IMG");
            MyGlobals.ImageOk = "Ok";
        }
    }
    else
        MessageBox.Show("can not compare this images");
```

6.6.1 En siste verifisering

For noen funksjoner så kreves det både bilde verifisering og tilstands verifisering fra våpenstasjonen. Hvis dette kreves(noe det gjør for alle som blir bilde verifisert), så sendes resultatet for bilde og resultatet fra våpenstasjonen inn til en siste verifisering av kommandoen.

Her sjekkes bilde resultatet og tilstands resultatet opp mot hverandre og hvis begge er OK så godkjennes resultatet som vist under:

```
// verifies message and image
public void VerifyMessageWSImg()
{
    txtDataBox.DisplaytxtData("Verifies Message and Image" + "\n");
    if (MyGlobals.WsMessageOk == "Ok" && MyGlobals.ImageOk == "Ok")
    {


        Console.WriteLine("Message OK!!");
        string WsMessage = "message and image OK, test past";
        DateTime time = DateTime.Now;
        using (System.IO.StreamWriter file = new System.IO.StreamWriter(MyGlobals.appPath + @"\" + fileName, true))
        {
            file.WriteLine(time.ToString(format));
            file.WriteLine(WsMessage);
        }

        MyGlobals.CommandOK = "OK";
        SequenceResaultList.Add(MyGlobals.CommandOK);
        txtDataBox.DisplaytxtData("Message and image okey! test ok!" + "\n");
    }
```

Dersom en av dem eller begge feiler så blir ikke kommandoen godkjent. Resultatene blir uansett loggført.

7 Verifisering av sekvens

Når en kommando blir verifisert så settes de til «OK» eller «Not OK» og settes inn i et sekvensresultat array list. For å verifisere sekvensen sjekker man om arrayet inneholder «Not OK», hvis den gjør dette feiler sekvensen og mans ender videre til rapporten via en global variabel at sekvensen feilet.

Dersom den ikke finner «Not Ok», men bare «OK» så verifiseres sekvensen som godkjent og rapporten får beskjed om at sekvensen ble kjørt riktig. Rapporten godkjenner sekvensen.

Sekvensen vil loggføre resultatet til en loggfil og i output.

Etter kjørt sekvens nuller man ut arrayet så det er klart for neste sekvens.

Under ligger kode av metoden.

```
// verify sequence
 public void VerifySequence()
 {
     txtDataBox.DisplaytxtData("Verifies Sequence" + "\n");
     DateTime time = DateTime.Now;
     if (SequenceResaultList.Contains("Not OK"))
     {
         Console.WriteLine("Test sequence" + MyGlobals.onSequence + " FAILED");

         using (System.IO.StreamWriter file = new System.IO.StreamWriter(MyGlobals.appPath + @"\" + fileName, true))
         {
             file.WriteLine(time.ToString(format));
             file.WriteLine("Test sequence" + MyGlobals.onSequence + " FAILED");
         }
         SequenceResaultList.Clear();
         txtDataBox.DisplaytxtData("Sequence failed"+"\n");
     }
     else
     {
         MyGlobals.sequenceResult = true;
         Console.WriteLine("TEST SEQUENCE OK" );

         using (System.IO.StreamWriter file = new System.IO.StreamWriter(MyGlobals.appPath + @"\" + fileName, true))
         {
             file.WriteLine(time.ToString(format));
             file.WriteLine("Test sequence" + MyGlobals.onSequence + " OK");
         }
         SequenceResaultList.Clear();
         txtDataBox.DisplaytxtData("Sequence Success" + "\n");

     }
```

8 Feilhåndtering

En funksjon som er viktig i dette systemet er å kunne håndtere feilsituasjoner. Det vil si å detektere feil på våpenstasjonen under kjøring av testen. Her så vi på ulike scenarioer og kom opp med at vi skulle prøve å se om vi fikk til å detektere uønsket bevegelse, uønsket fyring og sjekke opp feilmeldings registre over fatale feil.

Målet var å måle fyringstilstanden ved hjelp av et eksternt måleutstyr og la dette detektere spenning på fyringssolenoiden og si fra til programmet vårt. Vi skulle i programmet ha en sjekk på når det skulle være spenning og ikke.

Når det gjelder bevegelse og sjekk av feilmeldings registre skal dette sjekkes gjennom hele kjøringen med unntak av når vi verifiserer feilmeldinger eller bevegelse. Dette skulle vi gjøre ved å sende inn kommandoer så vi får ut riktig tilstand, for så å se at den ikke bevegde seg uønsket eller hadde fatale feil. Her skulle vi ha en sjekk på om feilhåndteringen skulle kjøres eller ikke. For noen ganger skal man kunne bevege våpenstasjonen osv.

Vi lykkes ikke helt med å håndtere feilsituasjoner. Vi lagde en klasse som tar for seg dette, men som ikke er ordentlig implementert i systemet vårt. Den fungerte når vi hadde den som en egen del, men den skapte litt kluss i programmet når vi testet den sammen med systemet.

Den klassen tar for seg sjekk av fatale feil og uønsket bevegelse. Det den gjør er å sende inn kommandoer som gir tilbake tilstander, deretter kjøres disse inn i en «regular expression» som henter ut tilstander om feilmeldinger og bevegelse og sender dette videre til en metode som sjekker av disse tilstandene. Hvis disse metodene enten detekterer bevegelse eller fatale feil så vil de rapportere systemfeil og kutte programmet. Her har vi laget en sjekk i programmet for når denne feilhåndteringsklassen skal kjøre og ikke. Slik at den skal kjøres når vi ikke ønsker at det skal være bevegelse eller fatale feil i systemet. Denne metoden for sjekk av bevegelse og fatale feil kjøres i en egen tråd.

Når det gjelder fyring som kanskje er det viktigste å sjekke så lykkes vi ikke med å få ferdig det eksterne måleutstyret, så dette er ikke implementert enda, men vil være med i fremtiden. Dette utstyret vil da måle spenningen på fyringssolenoiden og rapportere til systemet vårt når det er spenning der. Systemet vårt vil dermed bestemme om det er riktig at det skal være spenning der eller ikke. Hvis det ikke skal være spenning og utstyret rapporterer spenning vil det komme fatal systemfeil og programmet avsluttes. Dette er som sagt ikke implementert enda, men denne ville da bli kjørt i en egen tråd som sjekker fyring gjennom hele testen.

8.1 Feilhåndteringsklassen

Denne klassen fungerer ved at det er en parser som får inn tilstander fra våpenstasjonen hele tiden med mindre det er annen verifikasjon kjørende. Den kjøres i en egen tråd så denne vil stå og spinne under hele testen. Måten ven vet om den skal kjøre eller ikke blir satt ved et flagg som er true når verifikasjon kjøres og false når feilhåndtering kjøres. Dette gjør man fordi vi sjekker blant annet bevegelse og feilmeldinger som verifikasjon. Vi er kun ute etter uønskede tilfeller i feilhåndteringen.

Under vises kode av parseren som detekterer bevegelse:

```
public void ErrorMessageCheck(string test){
    string ErrorHandling = test;
    if (ErrorHandling.Contains("Main Axis"))
    {
        foreach (Match match in Regex.Matches(ErrorHandling, "[+-]?(?=[0-6][.][0-9])[0-9]*(?:[.][0-9]*)?",
                    RegexOptions.IgnoreCase))

        {

            MovementMatch = match.ToString();

            if (res == 1)
            {
                b = Convert.ToDouble(MovementMatch, System.Globalization.CultureInfo.InvariantCulture);
                Console.WriteLine(MovementMatch);

            }
            if (res == 0)
            {
                a = Convert.ToDouble(MovementMatch, System.Globalization.CultureInfo.InvariantCulture);
                Console.WriteLine(MovementMatch);
                treff = match.Success;
                res = 1;
            }

        }}
```

Den sjekker da posisjonen til våpenstasjonen ved å sette a verdi til elevasjon og b verdi til azimuth. Feilmeldingsregister parseren ser slik ut:

```
foreach (Match match in Regex.Matches(ErrorHandling, "\\b(errorId = [0-9]*)(, )([0-9]*)",
            RegexOptions.IgnoreCase))
{

    WarningMessageHit = match.ToString();

}
foreach (Match match in Regex.Matches(ErrorHandling, "\\b(Fatal)",
            RegexOptions.IgnoreCase))
{

    FatalErrorHit = match.ToString();

}
```

Denne sjekker feilmeldingsregisteret etter fatale feil.

8.1.1 Bevegelse

Metoden for bevegelse går ut på å sjekke posisjonen hele tiden når feilhåndteringen er i gang, og se om det er endringer i posisjonen. Dersom det er endring vil det si at våpenstasjonen har beveget seg og vil den gi utslag og stoppe systemet.

Koden ser slik ut:

```
public void CheckMovement()
{

      Thread.Sleep(100);
      CommunicationWS_SendData.CommunicationWS_Send(" MainAxisPrint"+"\n",false);
      Thread.Sleep(1000);
      azBefore=a;
      elBefore=b;
      Thread.Sleep(1500);
    CommunicationWS_SendData.CommunicationWS_Send(" MainAxisPrint"+"\n",false);
      Thread.Sleep(1000);
      azAfter=a;
      elAfter=b;
      Thread.Sleep(1000);
      double elevation = elAfter - elBefore;
      double azimuth = azAfter - azBefore;


      if ((elevation > 0.000004) || (azimuth > 0.000004) || (elevation < -0.000004) || (azimuth < -0.000004))
             {

                    string ErrorMessage = "WS MOVED, system failure. test ended";
                    DateTime time = DateTime.Now;
                    using (System.IO.StreamWriter file = new System.IO.StreamWriter(MyGlobals.appPath + @"\" + fileName, true))
                        {
                            file.WriteLine(time.ToString(format));
                            file.WriteLine(ErrorMessage);
                        }
                    MyGlobals.mainThread.Abort();
                    MyGlobals.tRobot.Abort();
                    CommunicationRC_SendData.CommunicationRC_Send("STOP");
                    MessageBox.Show("SYSTEM HAS MOVED FATAL ERROR!!");
                    Console.WriteLine("FatalError! WS has MOVED!!");


              }
```

8.1.2 Feilmeldingsregisteret

Denne metoden sjekker feilmeldingsregisteret kontinuerlig mens feilhåndteringen er aktivert. Ved fatale feil vil denne detektere det og gi melding om fatal error og avbryte testen.

Koden for dette vises under:

```
public void CheckMessagePool()
{
    while(true){
        while(MyGlobals.verificationCheck==false)
        {
            Console.WriteLine("Er i errorcheck");
            Thread.Sleep(100);
            CommunicationWS_SendData.CommunicationWS_Send(" printWarmResetPool"+"\n",false);
            Thread.Sleep(100);


             DateTime time = DateTime.Now;


           if ((WarningMessageHit != null) && (!WarningMessageHit.Equals("errorId = 43, 301")))
            {
                Console.WriteLine(" Messages in errorpool");
                        string ErrorMessage1 = "messages in errorpool, test still running";

                        using (System.IO.StreamWriter file = new System.IO.StreamWriter(MyGlobals.appPath + @"\" + fileName, true))
                            {
                                file.WriteLine(time.ToString(format));
                                file.WriteLine(ErrorMessage1);
                            }
                    if(FatalErrorHit!=null)
                    {
                        string ErrorMessage2 = "Fatal Error, system failure. test ended";

                        using (System.IO.StreamWriter file = new System.IO.StreamWriter(MyGlobals.appPath + @"\" + fileName, true))
                            {
                                file.WriteLine(time.ToString(format));
                                file.WriteLine(ErrorMessage2);
                            }
                        MyGlobals.mainThread.Abort();
                        MyGlobals.tRobot.Abort();
                        CommunicationRC_SendData.CommunicationRC_Send("STOP");
                        MessageBox.Show("SYSTEM FATAL MESSAGE ERROR!!");
                        Console.WriteLine("FATAL ERROR STOP!");


                    }
```

9 Konklusjon

ACT programvaren har nå en god verifiseringsmetode ved hjelp av å lese ut tilstander fra våpenstasjonen. Det er også en bildegjenkjenningsfunksjon som verifiserer de mest vanlige tilstandene. Verifikasjonen er godt implementert og vi føler at verifiseringen er stabil nok til å stole på. Vi fikk implementert feilhåndtering, men under integrasjonen viste det seg at feilhåndteringen ikke ville helt samarbeide med resten av programmet.

Kort sagt så fungerer verifikasjonen til alt vi har bruk for og trenger bare å videreutvikles med nye verifiseringsmetoder, slik at den i senere tid kan bli brukt til alle tester som KPS skal utføre.

Når programmet har verifisert alle resultater skrives dette til en rapport som lagres lokalt på disk.

10 Referanser

REFERANSE	DOKUMENTTITTEL	VERSJON
[1]	Kravspesifikasjon	2.0

Tabell 3: Referanser

ACT AUTOMATED CROWS TESTING

TESTSPESIFIKASJON

AUGUST KIND SVENDSEN
EIRIK LIEN ROA
HENRIK BERGE SØRUM
STÅLE RUDIN

KONGSBERG

HBV
HØGSKOLEN I BUSKERUD OG VESTFOLD · PROFESJONSHØGSKOLEN ·

Testspesifikasjon

PROSJEKT	ACT – Automated CROWS Testing		
OPPDRAGSGIVER	Kongsberg Protech Systems		
UTFØRT VED	HBV – Høgskolen i Buskerud og Vestfold		
GRUPPE	August Kind Svendsen, Eirik Lien Roa, Henrik Berge Sørum og Ståle Rudin		
VERSJON	3.0		
ANTALL SIDER	10		
DOKUMENTHISTORIE	VERSJON	UTGITT	BESKRIVELSE
	1.0	17.02.14	Første utgivelse
	2.0	24.03.14	Andre utgivelse
	3.0	26.05.14	Tredje utgivelse

INNHOLDSFORTEGNELSE

LISTE OVER TABELLER

1 Om dokumentet

1.1 Dokumenthistorie

VERSJON	DATO	ENDRING	SIGNATUR
0.1	28.01.14	• Dokumentet ble opprettet	AKS, HBS
1.0	17.02.14	• Første utgivelse	AKS, ELR, HBS, SR
1.1	15.03.14	• Oppdatere tester etter nye krav satt av KPS	AKS, ELR, HBS, SR
2.0	21.03.14	• Skrive ferdig tester, oppdatert og se over dokumentet.	ELR
3.0	24.05.14	• Endret testmetode på RK.1, SK.2, FK.1 og FK.3 • Endret Godkjenningskriterium på RK.1, SK.2, FK.1 og FK.3 • Lagt til Opprettholdelse av utstyrskrav	HBS

Tabell 1: Dokumenthistorie

1.2 Definisjoner og forkortelser

Liste som beskriver ord og uttrykk som kan være brukt i dette dokumentet:

UTTRYKK	FORKLARING
HBV	Høgskolen i Buskerud og Vestfold
KPS	Kongsberg Protech Systems (oppdragsgiver)
ACT	Automated CROWS Testing
Industrirobot	Universal Robots 5
CROWS	Common Remotely Operated Weapon Station
RWS	Remote Weapon Station
DCP	Display Control Panel
MPU	Main Processing Unit
Konfigurasjon	Forskjellige CROWS oppsett
CG	Controll Grip

Tabell 2: Definisjoner og forkortelser

2 Innledning

Dette dokumentet består av tester som skal evaluere tilstanden til kravene gitt i kravspesifikasjonen [1]. Enkelte krav må opprettholdes for at prosjektet skal gjennomføres, derfor er det viktig at vi tester disse på en strukturert og systematisk måte.

2.1 Oppbygging av spesifikasjonen

Testspesifikasjonen tar for seg hvordan man skal verifisere de forskjellige kravene gitt i kravspesifikasjonen. Vi har sett på de forskjellige kravene og evaluert hvilken metode som vil egne seg best for å kunne teste at kravet blir opprettholdt. Det står også et tilhørende kriterium som vi kan sjekke opp mot for å anslå om testen kan sees på som godkjent eller ikke.

3 Test av krav

Dette kapittelet tar for seg hvordan de forskjellige kravene skal testes, og hva godkjenningskriteriet er.

3.1 Test av rammekrav

Her følger en oversikt over testing av rammekravene, testmetode for disse og hva som skal til for å få testen godkjent.

3.1.1 Test av RK.1

KRAV	PRIO	UTSTEDER	BESKRIVELSE
RK.1	A	KPS	Selvstendig system med minst mulig manuell assistanse. Systemet skal gjennomføre testsekvens selvstendig etter kalibrering og igangsetting.

3.1.1.1 Testmetode

Lage en testsekvens som består av de første sekvensene i bit testen. Testsekvensen skal kjøres minimum 10 ganger uten manuell assistanse. Alle kommandoer og verifiseringsmetoder skal loggføres. Når kjøringen av testsekvensene er fullført skal loggen visuelt sjekkes for å se at alle kommandoer og verifiseringsmetoder er gjennomført.

3.1.1.2 Godkjenningskriterium

Loggen stemmer 100 % med input.

3.1.2 Test av RK.2

KRAV	PRIO	UTSTEDER	BESKRIVELSE
RK.2	A	KPS	Systemet skal håndtere feilsituasjoner på en forutsigbar og sikker måte.

3.1.2.1 Testmetode

Her skal det settes i gang testsekvenser og simulere feil av forskjellig type feil og se at den da håndterer disse riktig. For eks. under fatal feil skal den stoppe og avslutte kjøring. Her skal vi da visuelt se at den stopper å kjøre og se om den kommer med riktig feilmelding. Det skal også loggføres så man kan sjekke loggen og se hvor det feilet.

3.1.2.2 Godkjenningskriterium

Industriroboten skal komme med riktige feilmeldinger og stoppe kjøring ved fatal error.

3.1.3 Test av RK.3

KRAV	PRIO	UTSTEDER	BESKRIVELSE
RK.3	B	KPS	Skal være mulig å legge til flere tester uten å endre kompilert kode.

3.1.3.1 Testmetode

Prøve å legge inn flere scripts og se om de kjører uten å bygge ny kode. Her må man loggføre kjøringen og se at den kjører riktig og kommer med riktige testresultater.

3.1.3.2 Godkjenningskriterium

Godkjent hvis testen lar seg kjøre uten feil og produserer forventede resultater.

3.1.4 Test av RK.4

KRAV	PRIO	UTSTEDER	BESKRIVELSE
RK.4	A	KPS	Testsekvenser skal loggføres og testresultat dokumenteres.

3.1.4.1 Testmetode

Prøve å kjøre en test, gi forskjellige inputs. Deretter skal det sjekkes at alt av testresultater dokumenteres og loggføres riktig i forhold til input.

3.1.4.2 Godkjenningskriterium

Testresultater blir loggført og dokumentert 100% i forhold til input.

3.2 Test av sikkerhetskrav

Her følger en oversikt over testing av sikkerhetskravene, testmetode for disse og hva som skal til for å få testen godkjent.

3.2.1 Test av SK.1

KRAV	PRIO	UTSTEDER	BESKRIVELSE
SK.1	A	KPS	Systemet skal avbryte kjøring ved fatal feil under test.

3.2.1.1 Testmetode

Sette i gang en test og simulere fatale feil. Deretter loggføre, se at den stopper å kjøre på riktig sted i testen og kommer med riktig feilmelding.

3.2.1.2 Godkjenningskriterium

Godkjent hvis systemet loggfører og roboten slutter å kjøre ved fatale feil.

3.2.2 Test av SK.2

KRAV	PRIO	UTSTEDER	BESKRIVELSE
SK.2	A	KPS	Software må kunne måle systemtilstander.

3.2.2.1 Testmetode

Systemet skal måle relevante systemtilstander under en testsekvens. Disse tilstandene skal testes og resultatet skal loggføres av systemet og visuelt verifiseres av testpersonene.

3.2.2.2 Godkjenningskriterium

Godkjent hvis logg stemmer med visuell sjekk.

3.2.3 Test av SK.3

KRAV	PRIO	UTSTEDER	BESKRIVELSE
SK.3	A	KPS	Systemet skal kunne avbrytes manuelt.

3.2.3.1 Testmetode

Sette i gang testsekvenser og teste ved flere anledninger at systemet avbrytes via stoppknapp.

3.2.3.2 Godkjenningskriterium

Godkjent hvis systemet avbrytes med 100% nøyaktighet hver gang vi trykker på stoppknapp.

3.3 Test av funksjonelle krav

Her følger en oversikt over testing av de funksjonelle kravene, testmetode for disse og hva som skal til for å få testen godkjent.

3.3.1 Test av FK.1

KRAV	PRIO	UTSTEDER	BESKRIVELSE
FK.1	A	KPS	Systemet skal utføre tester på CROWS automatisk.

3.3.1.1 Testmetode

Det skal kjøres diverse testsekvenser med loggføring av det som skjer. Det skal skrives ut en rapport når testsekvensene er utført. Verifiseringen skal godkjennes og rapport og logg skal stemme med hverandre.

3.3.1.2 Godkjenningskriterium

Godkjent hvis rapport og logg stemmer med hverandre.

3.3.2 Test av FK.2

KRAV	PRIO	UTSTEDER	BESKRIVELSE
FK.2	A	KPS	Systemet skal verifisere resultatene iht. CROWS software-krav.

3.3.2.1 Testmetode

Her skal vi kjøre i gang scripts i form av testsekvenser og se at loggen og testresultater stemmer overens med hverandre. Vi kommer også til å sjekke logg og testresultater opp mot forventede resultater.

I tillegg kommer vi til å teste det først manuelt og så automatisk for å se at man får samme resultater.

3.3.2.2 Godkjenningskriterium

Godkjent dersom testene stemmer 100% overens med forventede resultater og manuelle resultater.

3.3.3 Test av FK.2.1

KRAV	PRIO	UTSTEDER	BESKRIVELSE
FK.2.1	B	KPS	Systemet skal kunne verifisere CROWS software-krav ved bruk av bildegjenkjenning.

3.3.3.1 Testmetode

Sjekke bilde fra test opp mot et bilde fra en mappeplassering med korrekte bilder fra STD eller 2STD. Dette skal utføres ved hjelp av en bildesammenligningsalgoritme. Denne bildesammenligningen skal sjekke hver piksel i begge bildene og sammenligne disse med hverandre.

3.3.3.2 Godkjenningskriterium

Godkjent hvis pikslene i testbildene stemmer helt overrens med pikslene i det korrekte bildet.

3.3.4 Test av FK.2.2

KRAV	PRIO	UTSTEDER	BESKRIVELSE
FK.2.2	A	KPS	Systemet skal kunne verifisere CROWS software-krav ved bruk av data mottatt fra våpenstasjon.

3.3.4.1 Testmetode

Sammenligne data mottatt fra våpenstasjonen opp mot riktig resultat i forhold til STD eller 2STD.

3.3.4.2 Godkjenningskriterium

Godkjent hvis data mottatt fra våpenstasjonen stemmer 100 % med riktig resultat fra STD eller 2STD.

3.3.5 Test av FK.3

KRAV	PRIO	UTSTEDER	BESKRIVELSE
FK.3	B	KPS	Systemet må kunne lese av elektriske tilstander(strøm/spenning) via eksternt utstyr på våpenstasjonen.

3.3.5.1 Testmetode

Det eksterne utstyret skal kontinuerlig sende beskjeder til systemet vårt med spenninger hentet fra solenoiden. Det skal testes at det er spenning når fyring er aktivert, og at det ikke er spenning når fyring er aktivert når ”SYSTEM” bryteren er i ”SAFE”.

3.3.5.2 Godkjenningskriterium

Godkjent hvis vi kan måle at det er spenning når ”SYSTEM” bryteren er i ”ARM” og at det ikke er spenning når ”SYSTEM” bryteren er i ”SAFE”.

3.3.6 Test av FK.4

KRAV	PRIO	UTSTEDER	BESKRIVELSE
FK.4	B	KPS	Systemet må ha en hensiktsmessig oppdateringsfrekvens til ulike deler av systemet.

3.3.6.1 Testmetode

Her skal vi analysere om oppdateringsfrekvensene vi har er hensiktsmessige og eventuelt ikke går glipp av data p.g.a. for lav oppdatering, eller er for høy og bruker unødige ressurser.

3.3.6.2 Godkjenningskriterium

Systemet har tilstrekkelig oppdateringsfrekvens for at systemet skal fungere som ønsket.

3.4 Opprettholdelse av utstyrskrav

Her følger en oversikt over utstyrskrav som må opprettholdes gjennom hele prosjektet.

3.4.1 Opprettholdelse av UK.1

KRAV	PRIO	DATO	UTSTEDER	BESKRIVELSE
UK.1	B	27.01.14	Gruppa	Må ha tilgang til samme Protector CROWS konfigurasjon gjennom hele prosjektet.

3.4.1.1 Godkjenningskriterium

Beskrivelsen av kravet må være opprettholdt gjennom hele prosjektet.

3.4.2 Opprettholdelse av UK.2

KRAV	PRIO	DATO	UTSTEDER	BESKRIVELSE
UK.2	A	14.02.14	Gruppa	Får arbeide på samme versjon av software til CROWS systemet gjennom hele prosjektet. Versjonen som vil bli brukt er CROWS 3.12.

3.4.2.1 Godkjenningskriterium

Beskrivelsen av kravet må være opprettholdt gjennom hele prosjektet.

3.4.3 Opprettholdelse av UK.3

KRAV	PRIO	DATO	UTSTEDER	BESKRIVELSE
UK.3	A	20.03.14	KPS	Må bruke vernesko med ESD.

3.4.3.1 Godkjenningskriterium

Beskrivelsen av kravet må være opprettholdt gjennom hele prosjektet.

3.4.4 Opprettholdelse av UK.4

KRAV	PRIO	DATO	UTSTEDER	BESKRIVELSE
UK.4	A	20.03.14	Gruppa	Ha tilgang til robot gjennom hele prosjektet.

3.4.4.1 Godkjenningskriterium

Beskrivelsen av kravet må være opprettholdt gjennom hele prosjektet.

3.4.5 Opprettholdelse av UK.5

KRAV	PRIO	DATO	UTSTEDER	BESKRIVELSE
UK.5	A	20.03.14	Gruppa	Ha nødvendig tilgang til utstyr og fasiliteter.

3.4.5.1 Godkjenningskriterium

Beskrivelsen av kravet må være opprettholdt gjennom hele prosjektet.

4 Referanser

REFERANSE	DOKUMENTTITTEL	VERSJON
[1]	Kravspesifikasjon	2.0

Tabell 3: Referanser

ACT AUTOMATED CROWS TESTING

TESTRAPPORT

AUGUST KIND SVENDSEN
EIRIK LIEN ROA
HENRIK BERGE SØRUM
STÅLE RUDIN

KONGSBERG

HØGSKOLEN I BUSKERUD OG VESTFOLD · PROFESJONSHØGSKOLEN · HBV

Testrapport			
PROSJEKT	ACT – Automated CROWS Testing		
OPPDRAGSGIVER	Kongsberg Protech Systems		
UTFØRT VED	HBV – Høgskolen i Buskerud og Vestfold		
GRUPPE	August Kind Svendsen, Eirik Lien Roa, Henrik Berge Sørum og Ståle Rudin		
VERSJON	1.0		
ANTALL SIDER	14		
DOKUMENTHISTORIE	VERSJON	UTGITT	BESKRIVELSE
	1.0	26.05.14	Første utgivelse

INNHOLDSFORTEGNELSE

LISTE OVER TABELLER

1 Om dokumentet

1.1 Dokumenthistorie

VERSJON	DATO	ENDRING	SIGNATUR
0.1	13.05.14	• Dokumentet ble opprettet	AKS, ELR, HBS, SR
1.0	24.05.14	• Første utgivelse	HBS

Tabell 1: Dokumenthistorie

1.2 Definisjoner og forkortelser

Liste som beskriver ord og uttrykk som kan være brukt i dette dokumentet:

UTTRYKK	FORKLARING
HBV	Høgskolen i Buskerud og Vestfold
KPS	Kongsberg Protech Systems (oppdragsgiver)
ACT	Automated CROWS Testing
Gruppa	Prosjektgruppa
Industrirobot	Universal Robots 5
CROWS	Common Remotely Operated Weapon Station
RWS	Remote Weapon Station
DCP	Display Control Panel
MPU	Main Processing Unit
Konfigurasjon	Forskjellige CROWS oppsett
CG	Control Grip
Bit test	Funksjonstest gjort på CROWS av KPS

Tabell 2: Definisjoner og forkortelser

2 Innledning

I dette dokumentet vil vi dokumentere testresultatene gjort etter testspesifikasjonen[1]. Testene i testspesifikasjonen er ment for å verifisere om kravene i kravspesifikasjonen[2] blir oppfylt eller ikke. Dette dokumentet holder på resultatet til siste test utført, dersom testene har måtte kjøres flere ganger. Viktigst står A-kravene og deretter B-kravene.

3 Testmetoder

Vi har brukt flere forskjellige testmetoder for å teste om kravene har blitt oppfylt. Vi har laget egne XML-dokumenter for å teste spesifikke krav og kjørt gjennom denne testen flere ganger for å forsikre oss om at kravet godkjennes. Denne måten å teste på gir oss et resultatet vi kan være trygge på, og alle kravene som er godkjent har måtte vært godkjent minimum de fem siste gangene testen ble utført.

4 Test av krav til ACT systemet

4.1 Rammekrav

Test av rammekrav 1					
Krav:	RK.1	Prioritet:	A	Utsteder:	KPS
Beskrivelse:	Selvstendig system med minst mulig manuell assistanse. Systemet skal gjennomføre testsekvens selvstendig etter kalibrering og igangsetting.				
Testmetode:	Lage en testsekvens som består av de første sekvensene i bit testen. Testsekvensen skal kjøres minimum 10 ganger uten manuell assistanse. Alle kommandoer og verifiseringsmetoder skal loggføres. Når kjøringen av testsekvensene er fullført skal loggen visuelt sjekkes for å se at alle kommandoer og verifiseringsmetoder er gjennomført.				
Godkjenningskriterium:	Loggen stemmer 100 % med input.				
Kommentar:	Testsekvensene er kjørt flere ganger og loggen stemmer 100% med input. Systemet uforer alle oppgaver selv og verifiserer disse med data hentet fra våpenstasjon.				
Utført av:	August Kind Svendsen og Henrik Berge Sørum				
Dato utført:	23.05.14		Resultat:	Godkjent	

Tabell 3: Test av rammekrav 1

Test av rammekrav 2					
Krav:	RK.2	Prioritet:	A	Utsteder:	KPS
Beskrivelse:	Systemet skal håndtere feilsituasjoner på en forutsigbar og sikker måte.				
Testmetode:	Her skal det settes i gang testsekvenser og simulere feil av forskjellig type feil og se at den da håndterer disse riktig. For eks. under fatal feil skal den stoppe og avslutte kjøring. Her skal vi da visuelt se at den stopper å kjøre og se om den kommer med riktig feilmelding. Det skal også loggføres så man kan sjekke loggen og se hvor det feilet.				
Godkjenningskriterium:	Industriroboten skal komme med riktige feilmeldinger og stoppe kjøring ved fatal error.				
Kommentar:	Feilhåndtering er ikke ferdig implementert så dette er ikke testet. Kravet burde vært et B krav på grunn av at FK2.1 er B krav. Hensikten med prosjektet er å lage et rammeverk til funksjonstesting og både bildegjenkjenning og feilhåndtering må være med for at KPS skal kunne bruke ACT systemet til testing.				
Utført av:	Eirik Lien Roa og Henrik Berge Sørum				
Dato utført:	23.05.14		Resultat:	Feilet	

Tabell 4: Test av rammekrav 2

Test av rammekrav 3					
Krav:	RK.3	Prioritet:	B	Utsteder:	KPS
Beskrivelse:	Skal være mulig å legge til flere tester uten å endre kompilert kode.				
Testmetode:	Prøve å legge inn flere scripts i testsekvensen og se om de kjører uten å bygge ny kode. Her må man loggføre kjøringen og se at den kjører riktig og kommer med riktige testresultater.				
Godkjenningskriterium:	Godkjent hvis testen lar seg kjøre uten feil og produserer forventede resultater.				
Kommentar:	Ved hjelp av XML kan vi nå legge til eller endre tester uten å endre kompilert kode.				
Utført av:	Ståle Rudin og August Kind Svendsen				
Dato utført:	18.05.14		Resultat:	Godkjent	

Tabell 5: Test av rammekrav 3

Test av rammekrav 4					
Krav:	RK.4	Prioritet:	A	Utsteder:	KPS
Beskrivelse:	Testsekvenser skal loggføres og testresultat dokumenteres.				
Testmetode:	Prøve å kjøre en test med forskjellige inputs. Deretter skal det sjekkes at alt av testresultater dokumenteres og loggføres riktig i forhold til input.				
Godkjenningskriterium:	Testresultater blir loggført og dokumentert 100% i forhold til input.				
Kommentar:	Alle testresultater og sekvenser blir loggført i en egen txt fil og en testrapport blir lagret som en pdf fil.				
Utført av:	Alle				
Dato utført:	22.05.14		Resultat:	Godkjent	

Tabell 6: Test av rammekrav 4

4.2 Sikkerhetskrav

Test av sikkerhetskrav 1					
Krav:	SK.1	Prioritet:	A	Utsteder:	KPS
Beskrivelse:	Systemet skal avbryte kjøring ved fatal feil under test.				
Testmetode:	Sette i gang en test og simulere fatale feil. Deretter loggføre og se at den stopper å kjøre på riktig sted i testen og kommer med riktig feilmelding.				
Godkjenningskriterium:	Godkjent hvis systemet loggfører og roboten slutter å kjøre ved fatale feil.				
Kommentar:	Manglende feilhåndtering gjør at dette kravet ikke kan testes. "Fatale feil" som at våpenstasjonen beveger seg eller fyrer når dette ikke skal skje er ikke ferdig implementert.				
Utført av:	Eirik Lien Roa og Henrik Berge Sørum				
Dato utført:	23.05.14		Resultat:	Feilet	

Tabell 7: Test av sikkerhetskrav 1

Test av sikkerhetskrav 2					
Krav:	SK.2	Prioritet:	A	Utsteder:	KPS
Beskrivelse:	Software må kunne måle systemtilstander.				
Testmetode:	Systemet skal måle relevante systemtilstander under en testsekvens. Disse tilstandene skal testes og resultatet skal loggføres av systemet og visuelt verifiseres av testpersonene.				
Godkjenningskriterium:	Godkjent hvis logg stemmer med visuell sjekk.				
Kommentar:	Sjekket systemtilstandene Palm, Fire og Move med Arm = On og Arm = Off. Alle sekvenser gikk som forventet. Logg stemmer 100% med forventet resultat.				
Utført av:	August Kind Svendsen og Henrik Berge Sørum				
Dato utført:	23.05.14		Resultat:	Godkjent	

Tabell 8: Test av sikkerhetskrav 2

Test av sikkerhetskrav 3					
Krav:	SK.3	Prioritet:	A	Utsteder:	KPS
Beskrivelse:	Systemet skal kunne avbrytes manuelt.				
Testmetode:	Sette i gang testsekvenser og teste ved flere anledninger at systemet avbrytes via stoppknapp.				
Godkjenningskriterium:	Godkjent hvis systemet avbrytes med 100% nøyaktighet hver gang vi trykker på stoppknapp.				
Kommentar:	Systemet har en stoppknapp som fungerer med 100% nøyaktighet.				
Utført av:	Ståle Rudin og August Kind Svendsen				
Dato utført:	18.05.14		Resultat:	Godkjent	

Tabell 9: Test av sikkerhetskrav 3

4.3 Funksjonelle krav

Test av funksjonskrav 1					
Krav:	FK.1	Prioritet:	A	Utsteder:	KPS
Beskrivelse:	Systemet skal utføre tester på CROWS automatisk.				
Testmetode:	Det skal kjøres diverse testsekvenser med loggføring av det som skjer. Det skal skrives ut en rapport når testsekvensene er utført. Verifiseringen skal godkjennes og rapport og logg skal stemme med hverandre.				
Godkjenningskriterium:	Godkjent hvis rapport og logg stemmer med hverandre.				
Kommentar:	Det er kjørt flere testsekvenser flere ganger for å sjekke at alt stemmer selv om testene blir kjørt flere ganger. Logg stemmer med rapport og all verifisering er riktig.				
Utført av:	Eirik Lien Roa og Henrik Berge Sørum				
Dato utført:	23.05.14		Resultat:	Godkjent	

Tabell 10: Test av funksjonskrav 1

Test av funksjonskrav 2					
Krav:	FK.2	Prioritet:	A	Utsteder:	KPS
Beskrivelse:	Systemet skal verifisere resultatene iht. CROWS software-krav.				
Testmetode:	Her skal vi kjøre i gang scripts i form av testsekvenser og se at loggen og testresultater stemmer overens med hverandre. Vi kommer også til å sjekke logg og testresultater opp mot forventede resultater. I tillegg kommer vi til å teste det først manuelt og så automatisk for å se at man får samme resultater.				
Godkjenningskriterium:	Godkjent dersom testene stemmer 100% overens med forventede resultater og manuelle resultater.				
Kommentar:	De første sekvensene i bit testen er kjørt flere ganger. Oppnådde samme resultat hver gang. Loggen stemmer overens med forventede resultater og de manuelle resultatene.				
Utført av:	August Kind Svendsen og Henrik Berge Sørum				
Dato utført:	23.05.14		Resultat:	Godkjent	

Tabell 11: Test av funksjonskrav 2

Test av funksjonskrav 2.1					
Krav:	FK.2.1	Prioritet:	B	Utsteder:	KPS
Beskrivelse:	Systemet skal kunne verifisere CROWS software-krav ved bruk av bildegjenkjenning.				
Testmetode:	Sjekke bilde fra test opp mot et bilde fra en mappeplassering med korrekte bilder fra STD eller 2STD. Dette skal utføres ved hjelp av en bildesammenligningsalgoritme. Denne bildesammenligningen skal sjekke hver piksel i begge bildene og sammenligne disse med hverandre.				
Godkjenningskriterium:	Godkjent hvis pikslene i testbildene stemmer helt overrens med pikslene i det korrekte bildet.				
Kommentar:	Det er kjørt flere testsekvenser der bildegjenkjenning er brukt som en del av verifikasjonen. Ved alle utførte tester har bildegjenkjenningen stemt 100% med "korrekte" bilder som blir hentet fra lokal mappe				
Utført av:	Eirik Lien Roa og Henrik Berge Sørum				
Dato utført:	23.05.14		Resultat:	Godkjent	

Tabell 12: Test av funksjonskrav 2.1

Test av funksjonskrav 2.2					
Krav:	FK.2.2	Prioritet:	A	Utsteder:	KPS
Beskrivelse:	Systemet skal kunne verifisere CROWS software-krav ved bruk av data mottatt fra våpenstasjon.				
Testmetode:	Sammenligne data mottatt fra våpenstasjonen opp mot riktig resultat i forhold til STD eller 2STD.				
Godkjenningskriterium:	Godkjent hvis data mottatt fra våpenstasjonen stemmer 100 % med riktig resultat fra STD eller 2STD.				
Kommentar:	Det er kjørt flere testsekvenser der det har blitt verifisert data mottatt fra våpenstasjonen opp mot riktig resultat i forhold til STD eller 2STD.				
Utført av:	Eirik Lien Roa og Henrik Berge Sørum				
Dato utført:	23.05.14		Resultat:	Godkjent	

Tabell 13: Test av funksjonskrav 2.2

Test av funksjonskrav 3					
Krav:	FK.3	Prioritet:	B	Utsteder:	KPS
Beskrivelse:	Systemet må kunne lese av elektriske tilstander(strøm/spenning) via eksternt utstyr på våpenstasjonen.				
Testmetode:	Det eksterne utstyret skal kontinuerlig sende beskjeder til systemet vårt med spenninger hentet fra solenoiden. Det skal testes at det er spenning når fyring er aktivert, og at det ikke er spenning når fyring er aktivert når ”SYSTEM” bryteren er i ”SAFE”.				
Godkjenningskriterium:	Godkjent hvis vi kan måle at det er spenning når ”SYSTEM” bryteren er i ”ARM” og at det ikke er spenning når ”SYSTEM” bryteren er i ”SAFE”.				
Kommentar:	Måleutstyr er ikke 100% ferdig og derfor får vi ikke testet dette kravet.				
Utført av:	August Kind Svendsen og Henrik Berge Sørum				
Dato utført:	23.05.14		Resultat:	Feilet	

Tabell 14: Test av funksjonskrav 3

Test av funksjonskrav 4					
Krav:	FK.4	Prioritet:	B	Utsteder:	KPS
Beskrivelse:	Systemet må ha en hensiktsmessig oppdateringsfrekvens til ulike deler av systemet.				
Testmetode:	Her skal vi analysere om oppdateringsfrekvensene vi har er hensiktsmessige og eventuelt ikke går glipp av data p.g.a. for lav oppdatering, eller er for høy og bruker unødige ressurser.				
Godkjenningskriterium:	Systemet har tilstrekkelig oppdateringsfrekvens for at systemet skal fungere som ønsket.				
Kommentar:	Systemets oppdateringsfrekvens er tilstrekkelig for at informasjonen funksjonene krever fanges opp tidsnok, og ingen er unødige høye.				
Utført av:	August Kind Svendsen og Henrik Berge Sørum				
Dato utført:	23.05.14		Resultat:	Godkjent	

Tabell 15: Test av funksjonskrav 4

4.4 Utstyrskrav

Test av utstyrskrav 1					
Krav:	UK.1	Prioritet:	B	Utsteder:	Gruppa
Beskrivelse:	Må ha tilgang til samme Protector CROWS konfigurasjon gjennom hele prosjektet.				
Godkjenningskriterium:	Beskrivelsen av kravet må være opprettholdt gjennom hele prosjektet.				
Kommentar:	Har hatt tilgang til samme Protector CROWS konfigurasjon gjennom hele prosjektet.				
Utført av:	Alle				
Dato utført:	24.05.14		Resultat:	Godkjent	

Tabell 16: Test av utstyrskrav 1

Test av utstyrskrav 2					
Krav:	UK.2	Prioritet:	A	Utsteder:	Gruppa
Beskrivelse:	Får arbeide på samme versjon av software til CROWS systemet gjennom hele prosjektet. Versjonen som vil bli brukt er CROWS 3.12.				
Godkjenningskriterium:	Beskrivelsen av kravet må være opprettholdt gjennom hele prosjektet.				
Kommentar:	Har arbeidet på samme versjon av software til CROWS-konfigurasjon gjennom hele prosjektet.				
Utført av:	Alle				
Dato utført:	24.05.14		Resultat:	Godkjent	

Tabell 17: Test av utstyrskrav 2

Test av utstyrskrav 3					
Krav:	UK.3	Prioritet:	A	Utsteder:	KPS
Beskrivelse:	Må bruke vernesko med ESD.				
Godkjenningskriterium:	Beskrivelsen av kravet må være opprettholdt gjennom hele prosjektet.				
Kommentar:	Alle har brukt vernesko med ESD innenfor markert område på kontoret gjennom hele prosjektet.				
Utført av:	Alle				
Dato utført:	24.05.14		Resultat:	Godkjent	

Tabell 18: Test av utstyrskrav 3

Test av utstyrskrav 4					
Krav:	UK.4	Prioritet:	A	Utsteder:	Gruppa
Beskrivelse:	Ha tilgang til robot gjennom hele prosjektet.				
Godkjenningskriterium:	Beskrivelsen av kravet må være opprettholdt gjennom hele prosjektet.				
Kommentar:	Har hatt tilgang tilroboten gjennom hele prosjektet.				
Utført av:	Alle				
Dato utført:	24.05.14		Resultat:	Godkjent	

Tabell 19: Test av utstyrskrav 4

Test av Utstyrskrav 5					
Krav:	UK.5	Prioritet:	A	Utsteder:	Gruppa
Beskrivelse:	Ha nødvendig tilgang til utstyr og fasiliteter.				
Godkjenningskriterium:	Beskrivelsen av kravet må være opprettholdt gjennom hele prosjektet.				
Kommentar:	Har hatt tilgang på nødvendig utstyr og fasiliterer gjennom hele prosjektet.				
Utført av:	Alle				
Dato utført:	24.05.14		Resultat:	Godkjent	

Tabell 20: Test av utstyrskrav 5

5 Konklusjon

All testing har gått som forventet. På grunn av dårlig tid så har ikke alle krav blitt oppfylt. Det manglende er feilhåndtering av systemet og eksternt måleutstyr. Feilhåndteringen er noe som er vanskelig å få på plass før de andre komponentene av systemet fungerer godt. Derfor har det ikke vært mulig å utvikle dette grundig nok, på grunn av prioritering av et godt fungerende fundament. Feilhåndteringen er implementert og fungerte alene, men det fungerte ikke som håpet når det ble integrert inn i systemet. Det eksterne måleutstyret er et prosjekt i seg selv, og måtte legges på is på grunn av hvor mye ressurser utviklingen krevet i forhold hvor stor rolle det spilte totalt sett. Både feilhåndtering og eksternt måleutstyr må på plass før KPS kan bruke systemet til testing, og vi har lagt grunnlagt for begge til å utvikles videre.

Noe som har kommet litt feil fram i krav spesifikasjonen er at kravene teoretisk sett tilsier at hvis alle krav er godkjent så er systemet ferdig og kan brukes til testing. På grunn av manglende tid og at prosjektet går ut på å lage et godt rammeverk som skal være lett å videreutvikle til et ferdig produkt så burde noen av kravene vært gjort om på tidligere så disse passet bedre til bachelorprosjektet. Systemet fungerer generelt bra og vi har ikke hatt noen feilsituasjoner under testingen. Eksternt måleutstyr er også på god vei, men er ikke helt ferdig implementert så det kan brukes i ACT systemet.

6 Referanser

REFERANSE	DOKUMENTTITTEL	VERSJON
[1]	Testspesifikasjon	3.0
[2]	Kravspesifikasjon	2.0

ACT AUTOMATED CROWS TESTING

SLUTTRAPPORT

AUGUST KIND SVENDSEN
EIRIK LIEN ROA
HENRIK BERGE SØRUM
STÅLE RUDIN

KONGSBERG

HBV
HØGSKOLEN I BUSKERUD OG VESTFOLD
PROFESJONSHØGSKOLEN

Sluttrapport			
PROSJEKT	ACT - Automated CROWS Testing		
OPPDRAGSGIVER	Kongsberg Protech Systems		
UTFØRT VED	HBV - Høgskolen i Buskerud og Vestfold		
GRUPPE	August Kind Svendsen, Eirik Lien Roa, Henrik Berge Sørum og Ståle Rudin		
VERSJON	1.0		
ANTALL SIDER	20		
DOKUMENTHISTORIE	VERSJON	UTGITT	BESKRIVELSE
	1.0	26.05.14	Første utgivelse

INNHOLDSFORTEGNELSE

LISTE OVER TABELLER

LISTE OVER FIGURER

1 Om dokumentet

1.1 Dokumenthistorie

VERSJON	DATO	ENDRING	SIGNATUR
0.1	21.05.14	• Dokumentet ble opprettet	HBS
1.0	25.05.14	• Første utgivelse	AKS, ELR, HBS, SR

Tabell 1: Dokumenthistorie

1.2 Definisjoner og forkortelser

Liste som beskriver ord og uttrykk som kan være brukt i dette dokumentet:

UTTRYKK	FORKLARING
HBV	Høgskolen i Buskerud og Vestfold
KPS	Kongsberg Protech Systems (oppdragsgiver)
ACT	Automated CROWS Testing
UR5	Universal Robots 5
CROWS	Common Remotely Operated Weapon Station
RWS	Remote Weapon Station
DCP	Display Control Panel
MPU	Main Processing Unit
Konfigurasjon	Forskjellige CROWS oppsett
Firing controller	Eksternt instrument som måler fyringsspenningen
CG	Control grip

Tabell 2: Definisjoner og forkortelser

2 Innledning

Denne rapporten beskriver oppgaven vi har fått, hvordan vi har valgt å løse den og hvilke resultater vi fikk.

Oppgaven var å lage et fundament for automatisering av funksjonstester som blir utført på CROWS systemer. Dette skulle vi gjøre ved hjelp av en innkjøpt robotarm og software. Robotarmen skulle gjøre det fysiske arbeidet og programmet skulle styre hele systemet. Programmet skulle sende beskjeder til robot om hvilke oppgaver den skulle utføre, verifisere informasjon hentet fra våpenstasjon og robot, gjenkjenne bilder hentet ut fra DCP og sammenligne disse med "riktige" bilder, skrive logg til en fil og til slutt skrive ut en rapport med resultatene til test sekvensene.

For å løse oppgaven har vi lært oss hvordan systemet og testene fungerer og ut i fra dette laget et program til robot og PC som skal utføre disse testene automatisk.

3 Problemstilling

Hvordan kan vi produsere et godt rammeverk for et program som skal automatisere funksjonstester på CROWS konfigurasjoner med hjelp av en industrirobot, og kunne loggføre hendelser og verifisere sekvenser?

4 Sammendrag

Oppdragsgiver for oppgaven er Kongsberg Protech Systems. Kongsberg Protech Systems (KPS) er en bedrift i teknologikonsernet Kongsberg Gruppen. Bedriften leverer teknologi og produkter hovedsakelig innenfor forsvarsmateriell.

Oppgaven var å lage et rammeverk til et system som skal automatisere funksjonstester som blir utført på CROWS systemene. Dette skulle gjøres ved hjelp av en industrirobot og software. Det var også stort fokus på at systemet skulle klare å verifisere tilstandene til systemet, loggføre alt som blir utført under en test, og lagre en rapport med oppnådde resultater.

ACT systemet er bygget opp av flere deler og vi har gått inn på hver av dem og vurdert hvordan disse teoretisk burde fungere. Roboten skal utføre fysiske oppgaver på DCP og CG. Verifikasjonen skal verifisere tilstander til CROWS systemet og bruke bildegjenkjenning som en ekstra verifisering slik at det er sikkert at en test blir utført korrekt. Feilhåndteringen skal håndtere feilsituasjoner på en forutsigbar og sikker måte. Det eksterne måleutstyret skal måle spenning på solenoiden slik at det kan avleses om avfyringen fyrer eller ikke. Systemet skal loggføre alle bevegelser og verifisering som skal lagres slik at det kan dokumentere alt som blir utført. Rapporten skal fylles ut underveis under en test og skal brukes til å dokumentere om sekvenser blir godkjent eller ikke.

De forskjellige delene i ACT systemet har blitt laget separat og integrert sammen til slutt. Det er derfor forskjellige metoder på framgangsmåten til de forskjellige delene.

Roboten var ganske lett å sette seg inn i og vi brukte ikke så lang til på å få til et utkast av programmet vi trengte, resultatet ble bra og roboten utfører alle nødvendige oppgaver.

Verifikasjonen har tatt lang tid og vi startet på denne tidlig. Vi bruker to forskjellige verifikasjons metoder, den ene er å sammenligne et bilde som er hentet ut fra DCP med et korrekt bilde og den andre verifikasjonen er å hente ut tilstander fra våpenstasjonen og se at disse stemmer med den tilstanden det skal være på det tidspunktet. Vi så på flere to metoder for verifisere bildeuthentingen men endte opp med å ta ut overlayet til bildet.

Vi trengte en måte å passe på at bare en ting kjører om ganger, derfor lagde vi en kontroller klasse som holder styr på hvem metoder som får lov å kjøre. Kontrolleren bruker informasjon fra XML dokumentet til å starte metoder, når en metode har fått lov til å kjøre blir tråden til kontroller klassen satt på pause slik at den metoden som har fått lov til å kjøre må kjøre ferdig før kontroller klassen sier at en ny metode får kjøre.

All kommunikasjon mellom programmet og eksterne komponenter går over lokalt nettverk med TCP/IP protokollen, med TCP/IP er det lett å overføre informasjon og bestemte oss fort for å bruke dette til kommunikasjonen.

Det må være lett for brukeren å legge til tester og det er det ved hjelp av XML. Dette er utrolig enkelt og lite tidskrevende i forhold til å endre eller lege til kode i programmet.
Feilhåndteringen skulle håndtere alle feilsituasjoner som kunne oppstå på en sikker og god måte. Vi fikk feilhåndteringen til å fungere alene, men det oppsto noen problemer når dette ble integrert med resten av programmet.

Det var viktig for KPS at alt som blir gjort under en test loggføres, så vi lagde en metode som loggfører alle bevegelser og all verifikasjon som skjer mens en test kjøres, i tillegg er det en rapport som skriver ut resultatene til sekvensene i en test.

5 Motivasjon

Når det lanseres nytt programvare for våpenstasjonene trenger KPS å gjennomgå grundige funksjonstester for å sjekke at de fungerer som de skal. Testene som blir utført er veldig tidkrevende og repetive, og per dags dato blir de gjort manuelt. Siden dette arbeidet er bortimot en ren gjørejobb så er det veldig egnet som en jobb for en robot. En regner med ca. 40 timer med testing på en konfigurasjon, og dette må gjøres på minimum tre forskjellige konfigurasjoner. Da skal det ikke mye til før et automatisert system vil spare bedriften for store kostnader.

En viktig del av funksjonstestene er å verifisere at resultatene stemmer overrens med det en forventer. Dette er ikke noe roboten kan gjøre på egenhånd og derfor vil det være nødvendig med et program som kan gjøre disse verifiseringene. Dette er bakgrunnen for oppgaven vi har tatt fatt på, og hvordan vi har løst den legger vi frem i dette dokumentet.

6 Teori

ACT systemet skal være et fundament av et system som skal kunne kjøre alle funksjonstester som er mulig å automatisere. Når en slik test skal automatiseres er det mange ting som må med i systemet. Robot skal utføre fysiske oppgaver, systemet skal være sikkert og håndtere potensielle feilsituasjoner på en bra måte, alt som blir gjort under en test må verifiseres og dokumenteres og til slutt lagre en rapport med oppnådde resultater.

6.1 Robot

Roboten skal utføre de fysiske oppgavene som vanligvis blir utført av mennesker. Oppgaver som trykke på knapper, flytte på CG, avfyre systemet osv. Dette må programmeres slik at en oppgave gjøres om gangen. Skal roboten trykke på av/på-knappen skal den utføre bare den bevegelsen. I tillegg må roboten holde resten av systemet oppdatert med hva den gjør og gi beskjed når den er ferdig slik at systemet kan fortsette sekvensen.

6.2 Verifikasjon

For at KPS skal kunne bruke ACT systemet må alle tilstander verifiseres i henhold til KPS sine krav. Det vil si at programmet må ha metoder som kjører sekvensielt slik at programmet kan verifisere hva som har skjedd i hver metode. Et eksempel på en verifikasjon er hvis roboten flipper bryteren ”SYSTEM” til ”ARM”. Det må hentes ut informasjon om hvilken tilstand våpenstasjonen er i og deretter sjekkes om den er i ønsket tilstand. Det må også utføres en bildeverifisering der det skal hentes ut et overlay fra DCP og deretter sammenligne dette bildet med et bilde som inneholder korrekt informasjon.

6.3 Feilhåndtering

Det er viktig at feilsituasjoner som kan oppstå behandles på en god måte. Hvis systemet utfører en test og en feilsituasjon oppstår så må systemet klare å håndtere denne feilen. Hvis det oppstår en kritisk feilsituasjon så må systemet stoppe og kjøre. Feilsituasjoner som at våpenstasjonen beveger seg når den ikke skal eller den fyrer når den ikke skal er kritiske feilsituasjoner og hele systemet må stoppe hvis dette skjer.

6.4 Eksternt måleutstyr

For å forbedre verifikasjonen så trenger vi et eksternt måleutstyr. Dette skal kobles til våpenstasjon og skal kontinuerlig måle spenning på solenoiden. Solenoiden er den som fyrer hvis ”SYSTEM” bryteren er i ”ARM” og palm+trigger blir trykket inn på CG. Solenoiden vil da ha spenning når det fyres og ingen spenning når det fyres. Hensikten med dette måleutstyret er å hente ut spenninger kontinuerlig slik at systemet kan stoppe å kjøre hvis solenoiden har spenning når den ikke skal ha det og for å gjøre verifikasjonen til fyring bedre.

6.5 Legge til nye tester

Det skal være lett å legge til nye tester. Det er mye jobb å endre kode eller legge til ny kode for å legge til nye tester. Derfor vil vi legge tester inn i XML-dokumenter som programmet bruker til å utføre testsekvensene. En test skal være delt opp i sekvenser og kommandoer. En test vil bestå av flere sekvenser og hver sekvens vil inneholde flere kommandoer. Hvis vi kan lage testene på denne måten så skal det ikke trenges å endre eller legge til kode for å legge til nye tester så lenge metodene testen trenger allerede er lagt inn.

6.6 Logg

Alt som ACT systemet utfører skal loggføres. Dette trengs for å kunne gå inn å sjekke om alt i en test har blitt utført og om det har skjedd en feil. Det skal lages en logg for hver gang programmet kjøres og det skal legges inn hva som har blitt gjort i loggen. En logg skal inneholde informasjon om når roboten startet med en bevegelse, når den er ferdig, når det skal verifiseres om noe stemmer, resultatet til verifisering og om alle sekvenser i en test er godkjent eller ikke.

6.7 Rapport

Det skal også lages en rapport som fylles ut av programmet. Denne rapporten skal være en ”erstatning” til KPS sin rapport som skal fylles ut når man utfører tester. Rapporten skal inneholde en sekvens id, test input, forventet resultat og om det forventede resultatet er godkjent eller ikke. Rapporten skal autogenereres ut ifra verifikasjonen og deretter fylle inn i om det forventede resultatet er godkjent eller ikke.

7 Metode

I dette kapittelet vil det være beskrevet hvordan vi har gått fram for å løse oppgaven. Det vil også stå hvordan løsninger vi har kommet fram til.

Vi har delt inn oppgaven i følgende deler:

- Hvordan skal vi få roboten til å utføre oppgaver?
- Hvordan skal vi verifisere tilstander?
- Hvordan skal programmet styre hva som kjøres?
- Hvordan skal de forskjellige delene i systemet kommunisere med hverandre?
- Hvordan legge til nye tester uten å endre kompilert kode?
- Hvordan skal systemet håndtere feilsituasjoner?
- Hvordan lage en logg som dokumenterer alt som utført?
- Hvordan lage en rapport som resultatene lagres i?

7.1 Hvordan skal vi få roboten til å utføre oppgaver?

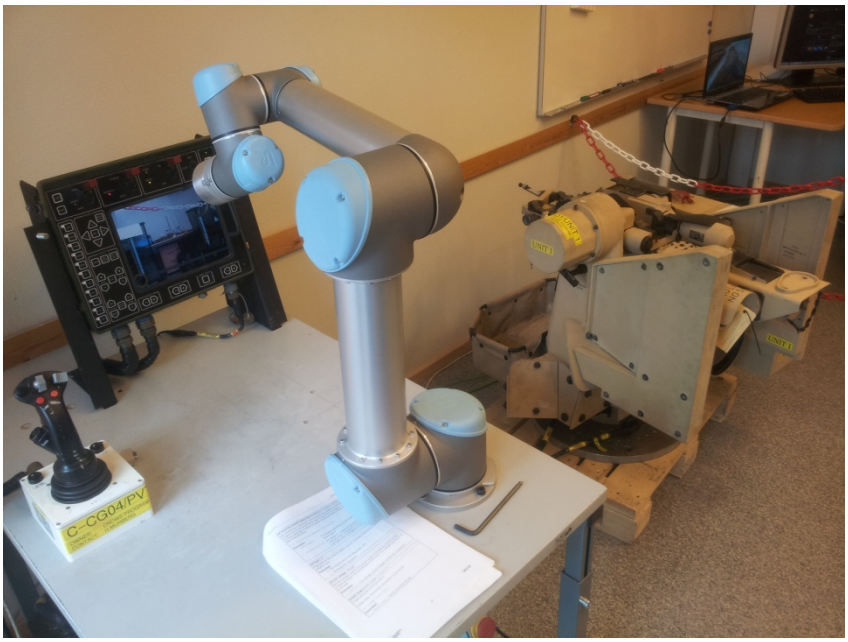
7.1.1 Fremgangsmåte

For å skjønne hvordan vi skulle få roboten til å utføre oppgaver måtte vi starte med å lese dokumentasjonen til roboten. Når vi hadde fått et overblikk over hvordan robotprogrammeringen fungerte så lagde vi enkle programmer og testet disse slik at vi skjønte hvordan de forskjellige funksjonene fungerte.

Når vi hadde laget noen små programmer og testet hvordan de ulike funksjonene fungerte startet vi å lage programmet som skulle brukes i vårt system. Det første som ble laget her var en TCP socket som skulle kommunisere med PC. Roboten skulle være klient og bare koble seg på PCen som var server.

TCP socketen fikk vi til å fungere ganske fort så når denne fungerte måtte vi finne ut hvordan vi skulle bygge opp programmet. Roboten skulle ha flere forskjellige bevegelser, skulle kunne sende og motta

beskjeder fra en PC og ha metoder som valgte hvem bevegelser som skulle kjøre ut ifra hvilken beskjed som ble mottatt fra PC.



Figur 1: Robot med DCP og CG

7.1.2 Løsning

Vi klarte å få robotprogrammet til å gjøre alt vi var ute etter ved hjelp av to tråder, flere metoder og subprogrammer.

Vi fant ut at vi trengte flere tråder for og kontinuerlig skulle sjekke meldingene mottatt fra PC-en. Derfor satt vi inn en tråd som leser beskjeder fra PC og bruker denne til forskjellige funksjoner. Beskjeden som blir hentet fra PC-en blir brukt til å kalle på subprogrammer og sjekke om det fortsatt er kommunikasjon mellom robot og PC. Hvis kommunikasjonen feiler så vil roboten motta en tom string og hvis den gjør dette to ganger så lukker den TCP socketen og prøver å koble til igjen.

For å kalle subprogrammer og starte en bevegelse så bruker vi en loop som går så lenge det er kommunikasjon mellom PC og robot. Denne loopen inneholder IF-setninger som kaller på subprogrammer. For at et bestemt subprogram skal starte må IF-setningen til det bestemte subprogrammet oppfylles.

Et subprogram er en oppgave roboten skal utføre. For eksempel hvis roboten skal trykke på av/på knappen på DCP. Med denne måten var det enkelt å lage unike bevegelser til hver enkelt oppgave roboten skal utføre. Subprogrammene består av flere script som sender beskjeder til PC og en bevegelsesfunksjon som utfører alle bevegelsene som trengs for oppgaven. Det er også en kraftfunksjon i noen av subprogrammene, som holder orden på at det ikke blir trykket for hardt.

7.2 Hvordan skal vi verifisere tilstander?

7.2.1 Fremgangsmåte

Verifiseringen er en viktig del av ACT systemet. Dette var noe vi tenkte på tidlig i prosjektet og som KPS nevnte gang på gang at var veldig viktig at vi fikk til. Vi valgte å se på to måter å verifisere på; bildegjenkjenning og se på tilstandene til våpenstasjonen.

Verifisering biten av softwaren vår skal verifisere tilstander til våpenstasjonen, hente ut bilder under en test og sammenligne dette med et bilde som inneholder korrekt informasjon. Deretter skal resultatet av disse to verifiseringene sammenlignes og sendes videre til testrapporten.

Når det gjelder bildegjenkjenning så vi på forskjellige metoder for å ta ut bilde, men endte opp med å ta ut kun «overlayet» av bilde, det vil si at vi tar ut kun det som blir printet på video feeden fra kameraet på våpenstasjonen. Vi så også på metoder for å ta ut teksten på skjermen, men lykkes ikke ordentlig i dette. Dermed endte vi opp med å ta ut delene av bilde vi var ute etter og sammenligne dette opp mot bilder med informasjon man er ute etter, og se at disse er identiske.

Kanskje den viktigste verifiseringsmetoden vi måtte se på var å verifisere signalene som våpenstasjonen sender. Her må vi lese ut tilstandene til våpenstasjonen og verifisere disse opp mot forventede resultat. Her prøvde vi og feilet en del, men fant ut at den letteste måten var å sende inn beskjeder til våpenstasjonen, få tilstandene tilbake og «parse» dette ved hjelp av «regular expressions» så vi kunne lese ut det vi var ute etter og deretter verifisere dette. Vi så også på å lese ut tilstander ved hjelp av eksternt måleutstyr, men dette er ikke implementert i softwaren vår enda. Dette vil være en klar forbedring av systemet vårt.

Vi må også ha en metode for å sammenligne resultatene av disse to verifiseringene. Her så vi på litt ulike metoder, men fant ut at vi setter globale variabler OK eller IKKE OK, og sammenligner disse. Denne metoden verifiserer en kommando.

I tillegg så må vi ha en måte å verifisere en hel sekvens. En sekvens er en rekke med kommandoer. Her kom vi opp med en måte som sjekker om alle kommandoene er OK og godkjenner eller ikke godkjenner sekvensen avhengig av om alle er OK eller ikke.

7.2.2 Løsning

Bildegjenkjenningen fungerer med at man har lagret bilder med forventet eller riktig resultat i en gitt fil destinasjon på PCen. Når testen kjøres henter man ut bilde fra våpenstasjonen og sammenligner opp mot et av bildene med forventet resultat for den testen man er på. Vi tar ut kun ut «overlayet» og dette valgte vi så vi ikke trenger å ta hensyn til bakgrunnen(«video feeden» fra våpensasjonen). Som regel er man ute etter bare en del av bilde og må derfor ta ut kun en del av hvert bilde og verifisere at disse er 100% like. Når begge bildene er funnet og klippet til kjøres en algoritme som sammenligner pixel for pixel. Hvis pixlene i bildene er identiske så godkjennes bildeverifiseringen og sender positivt resultat videre. Hvis pixlene ikke er identiske sendes negativt resultat.

Verifiseringen av tilstander på våpenstasjonen fungerer ved at man sender inn en kommando til våpenstasjonen som returnerer informasjon om forskjellige tilstander. Disse kommandoene kan variere fra hvordan tilstander du er ute etter. Når man får inn disse tilstandene fra våpenstasjonen går disse gjennom en «regular expression» som henter ut den tilstanden man er ute etter. Når den får match sendes tilstanden til en verifiseringsmetode som deretter verifiserer tilstanden opp mot forventet tilstand. Hvis dette stemmer så sender den positivt resultat, hvis ikke så blir resultatet negativt.

Når disse to verifiserings metodene er ferdig verifisert kjøres det en metode for å sammenligne disse resultatene. Bildegjenkjenningen og tilstands verifiseringen setter globale variable OK eller IKKE OK, denne metodene sjekker og sammenligner disse. Her godkjenner vi om vi har begge ok, og godkjenner ikke om en eller begge er IKKE OK. Her blir kommandoen satt til OK eller IKKE OK.

Den siste verifikasjonen er hvordan man verifiserer en hel sekvens av kommandoer. Her har vi da et array som tar inn alle kommando resultatene og sjekker om alle plassene i arrayet inneholder OK. Hvis den gjør det så godkjenner den sekvensen og sender «passed» til rapporten som godkjenner hele sekvensen. Om den inneholder IKKE OK, så sendes «failed» og sekvensen godkjennes ikke.

7.3 Hvordan skal programmet styre hva som kjøres?

7.3.1 Fremgangsmåte

Hvordan systemet skal bestemme hva som skal kjøres når og ha en oversikt over hva som kjører i programmet er et problem vi tenkte på tidlig i prosjektet. Det er viktig at vi vet hva som kjøres når vi kjører en test. Alt kan ikke kjøres samtidig og derfor bestemte vi oss tidlig for å kjøre alt sekvensielt, slik at en oppgave må bli helt ferdig før neste kan starte. Dette valgte vi fordi det ikke er noe tidskrav på hvor fort en test skal utføres.

Vi startet med å lage en klasse som skulle være ”hjernen” i systemet. Denne skulle holde styring på alt som skjer i systemet og bestemme hvem som har lov til å kjøre når. Siden vi lagde et XML dokument som holder på tester fant vi ut at denne klassen kan hente alle kommandoene samtidig og utføre disse sekvensielt.

7.3.2 Løsning

Når vi starter en test er det systemkontrolleren som holder styr på at testen kjører sekvensielt. Det systemkontrolleren trenger er et to-dimensjonalt array som holder på testen den skal kjøre. Når testen er startet vil kontrolleren kjøre gjennom dette arrayet med hjelp av to for løkker. Hvor den ytterste for-løkka bestemmer sekvensnummer og den innerste for-løkka bestemmer hvilke kommando som skal kjøres i den sekvensen programmet skal kjøre.

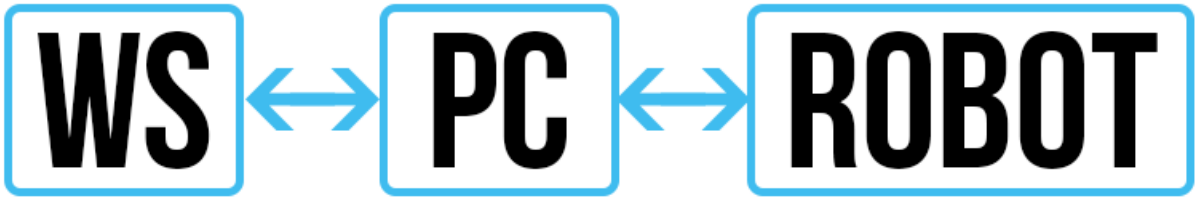
Når kontrolleren skal kjøre en kommando i en sekvens vil den gjøre et metodekall på en metode som heter det samme som kommandoen. Når denne metoden starter blir tråden som systemkontrolleren er i pauset helt til kommandoen er ferdig, og kan sette i gang neste kommando.

Når en hel sekvens er ferdig vil det bli skrevet til en database om sekvensen ble godkjent eller ikke. Når denne metoden blir kalt blir systemkontrolleren pauset til oppgaven er utført. Deretter kan kontrolleren starte på neste sekvens og kjøre samme prosedyre helt til testen er fullført.

7.4 Hvordan skal de forskjellige delene i systemet kommunisere med hverandre?

7.4.1 Fremgangsmåte

Noe av det første vi startet på var hvordan vi skulle kommunisere mellom de forskjellige enhetene. Det vi trengte var å kunne sende og motta stringer til robot og til våpenstasjonen. For å få til dette valgte vi å bruke en TCP/IP kommunikasjon over en socket på et lokalt nettverk.



Figur 2: Kommunikasjon

7.4.2 Løsning

Mellom PC og robot satte vi PC-en som en server og robot som en klient. Serveren vil lytte etter aktivitet på et bestemt portnummer. Når det er aktivitet på porten vil serveren godkjenne klienten og sette opp en toveis kommunikasjon.

Mellom PC og våpenstasjon er våpenstasjonen server og PC en klient. All kommunikasjon mellom PC og våpenstasjonen skjer over Telnet med portnummer 23.

7.5 Hvordan legge til nye tester uten å endre kompilert kode

7.5.1 Fremgangsmåte

Et ønske var at systemet skulle behandle tester på en modulær måte. Det vil at at man skulle kunne legge til og endre tester uten å måtte tukle med programkoden. Etter litt research fant vi ut at XML teknologien var godt egnet til dette. Dette er også veldig enkelt for en person å skrive, og kan gjøres rett i Notepad.

Etter det måtte vi finne ut hva testdokumentet skulle inneholde. Vi fant ut at det ville vært oversiktlig og lurt dersom vi etterlignet oppsettet på testene KPS brukte. De har en struktur som er at testen består av flere sekvenser, og hver sekvens består av flere steg eller kommandoer. Etter en sekvens er gjort så skal den verifiseres som godkjent eller ikke.

```
<test testname="bit test">
   <sequence id="1">
    <cmd part="1">dcp_press_on</cmd>
    <cmd part="2">connect_WS</cmd>
    <cmd part="3">verify_system_on</cmd>
    <cmd part="4">dcp_press_sel</cmd>
    <cmd part="5">dcp_press_safety_open</cmd>
    <cmd part="6">dcp_set_arm_on</cmd>
    <cmd part="7">cg_press_palm_fire</cmd>
    <cmd part="8">verifySequence</cmd>
   </sequence>
   <sequence id="2">
    <cmd part="1">verifyMessage</cmd>
    <cmd part="2">verifySequence</cmd>
   </sequence>
   <sequence id="3">
    <cmd part="1">dcp_set_arm_off</cmd>
    <cmd part="2">cg_press_palm</cmd>
    <cmd part="3">dcp_warm_reset</cmd>
    <cmd part="4">verifySequence</cmd>
   </sequence>
```

Figur 3: XML utdrag

7.5.2 Løsning

Vi brukte tags for å strukturere testen inn i sekvenser og kommandoer, som da ble <sequence> og <cmd>. Etter å ha laget et XML dokument som var strukturert slik måtte vi prøve å få programmet til å lese av dette, og lagre informasjonen som var der. I starten spilte det ingen rolle hva kommandoene inneholdt, vi ville bare hente informasjonen. Selve innholdet kunne vi tenke på senere.

Det finnes flere måter å hente informasjon fra et XML ark, men vi endte opp med å bruke LINQ to XML. Grunnen til dette var at da kunne vi bruke noen av kunnskapene fra et annet fag til å lage spørringer for å hente ut den informasjonen vi ville ha.

Når spørringene var på plass kunne vi bruke «for each» løkker til å skille mellom innholdet, og deretter putte dette inn i et todimensjonalt array, et for sekvenser og et for kommandoene. Nå trengte vi bare å fylle opp innholdet i tagsene med navn på de metodene vi skulle bruke for å utføre testen.

Prosessen ovenfor er beskrevet nærmere i teknologidokumentet for XML[2].

7.6 Hvordan skal systemet håndtere feilsituasjoner?

7.6.1 Fremgangsmåte

En funksjon som er viktig i dette systemet er å kunne håndtere feilsituasjoner. Det vil si å detektere feil på våpenstasjonen under kjøring av testen. Her så vi på ulike scenarioer og kom opp med at vi skulle prøve å se om vi fikk til å detektere uønsket bevegelse, uønsket fyring og sjekke opp feilmeldings registre over fatale feil.

Målet var å måle fyringstilstanden ved hjelp av et eksternt måleutstyr og la dette detektere spenning på fyrings solenoiden og si fra til programmet vårt. Vi skulle i programmet ha en sjekk på når det skulle være spenning og ikke.

Når det gjelder bevegelse og sjekk av feilmeldings registre skal dette sjekkes gjennom hele kjøringen med unntak av når vi verifiserer feilmeldinger eller bevegelse. Dette skulle vi gjøre ved å sende inn kommandoer så vi får ut riktig tilstand, for så å se at den ikke bevegde seg uønsket eller hadde fatale feil. Her skulle vi ha en sjekk på om feilhåndteringen skulle kjøres eller ikke. For noen ganger skal man kunne bevege våpenstasjonen osv.

7.6.2 Løsning

Vi lykkes ikke helt med å håndtere feilsituasjoner, men vi lagde en klasse som tar for seg dette, men som ikke er ordentlig implementert i systemet vårt. Den fungerte når vi hadde den som en egen del, men den skapte litt kluss i programmet vårt når vi testet den sammen med systemet.

Den klassen tar for seg sjekk av fatale feil og uønsket bevegelse. Det den gjør er å sende inn kommandoer som gir tilbake tilstander, deretter kjøres disse inn i en «regular expression» som henter ut tilstander om feilmeldinger og bevegelse og sender dette videre til en metode som sjekker av disse tilstandene. Hvis disse metodene enten detekterer bevegelse eller fatale feil så vil de rapportere systemfeil og kutte programmet. Her har vi laget en sjekk i programmet for når denne feilhåndtering klassen skal kjøre og ikke. Slik at den skal kjøres når vi ikke ønsker at det skal være

bevegelse eller fatale feil i systemet. Denne metoden for sjekk av bevegelse og fatale feil kjøres i en egen tråd.

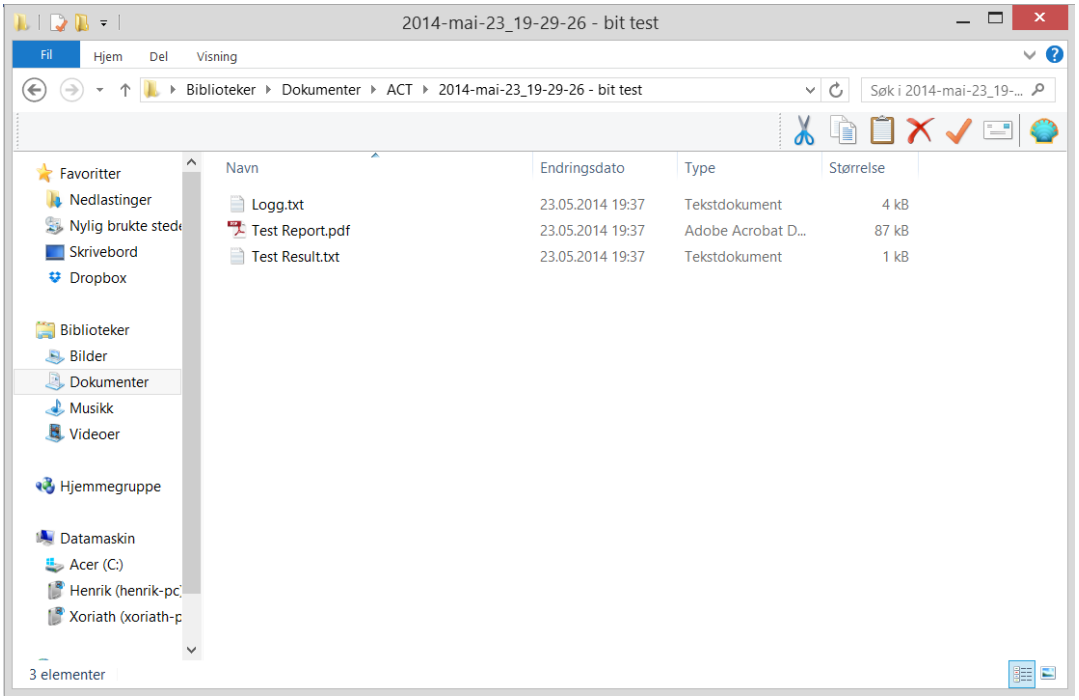
Når det gjelder fyring som kanskje er det viktigste å sjekke så lykkes vi ikke med å få ferdig det eksterne måleutstyret, så dette er ikke implementert enda, men vil være med i fremtiden. Dette utstyret vil da måle spenningen på fyringssolenoiden og rapportere til systemet vårt når det er spenning der. Systemet vårt vil dermed bestemme om det er riktig at det skal være spenning der eller ikke. Hvis det ikke skal være spenning og utstyret rapporterer spenning vil det komme fatal system feil og programmet avsluttes. Dette er som sagt ikke implementert enda, men denne ville da bli kjørt i en egen tråd som sjekker fyring gjennom hele testen.

For mer informasjon om feilhåndteringer, les teknologisk dokument om verifikasjon.

7.7 Hvordan lage en logg som dokumenterer alt som blir utført?

7.7.1 Fremgangsmåte

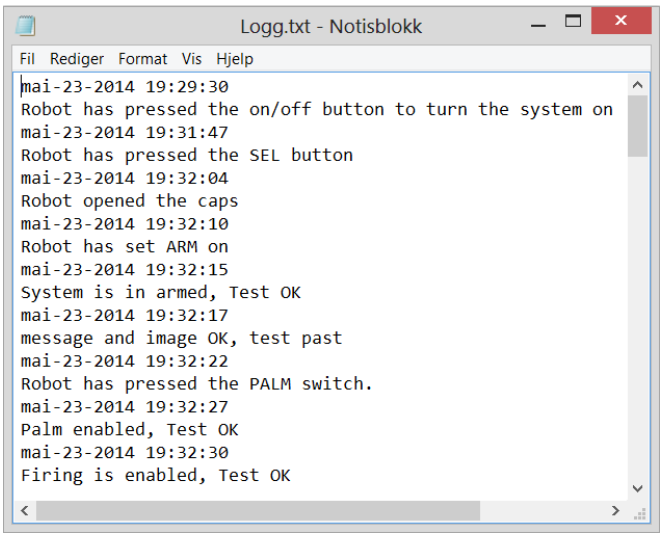
En viktig funksjon i systemet er å dokumentere de aktivitene som hender underveis i testingen. Derfor er det viktig at systemet loggfører alt som blir utført under en test. Denne loggen skal brukes som en del av dokumentasjonen for testen. Loggen må bli lagret slik at personer kan gå inn i denne og se etter feil etter testen er utført. Hvis en test blir kjørt og det oppstår en feil, så må testen kjøres på nytt. Uten en logg blir det umulig å vite om testen har blitt utført riktig eller ikke.



Figur 4: Mappe med loggfil og rapport

7.7.2 Løsning

For at programmet skal loggføre alt den gjør har vi satt inn i hver metode hvor enten roboten skal bevege seg eller noe skal verifiseres at det skal bli skrevet til en egen fil. For eksempel når roboten skal sette våpenstasjonen i «armed», vil det først bli loggført at roboten har beveget seg. Deretter vil det bli verifisert om våpenstasjonen faktisk ble satt i armed og da loggføre om den ble satt til armed eller ikke. Det blir også loggført hvilke tidspunkt alle bevegelser eller verifikasjoner har skjedd med både klokkeslett og dato.



Figur 5: Utdrag av loggfil

7.8 Hvordan lage en rapport som resultatene lagres i?

7.8.1 Fremgangsmåte

Når funksjonstester blir utført manuelt på CROWS systemene skal det sendes en input til systemet, deretter er det skrevet opp et forventet resultat som man skal godkjenne eller ikke. Det vil si at man utfører det som skal gjøres på systemet og sammenligner resultatet som man oppnår med det forventede resultatet. Hvis resultatene er like så godkjennes sekvensen.

Siden ACT systemet skal utføre disse testene automatisk så må det lages en rapport som fyller in resultatene underveis i testen. Vi bestemte oss for å lage en database som har fire kolonner som ”sekvens id”, ”input”, ”forventet resultat” og ”godkjent/feilet”. Vi ville at systemet automatisk skulle fylle inn sekvens id og godkjent/feilet slik at rapporten vil være helt lik den som man fyller inn manuelt.

Det første vi prøvde var å lage en SQLite database som var lokalt på PC-en så det ikke var nødvendig med internett tilkobling for å kjøre systemet. Vi trengte også et program for å bruke informasjonen i databasen og lage en rapport. Da så vi på noe som het Ireport som vi hadde fått vite var et bra program for å generere rapporter.

Vi møtte på en hel del problemer med denne løsningen så vi forhørte oss litt rundt og fant ut at vi burde brukt en annen database og gikk derfor over til PostGreSQL og vi prøvde dette med Ireport. Databasen fungerte bra, men vi fant ingen måte å få Ireport til å automatisk generere rapporter og lagre disse.

Derfor startet vi på nytt igjen og så på andre mulige løsninger, her var det en måte som vi var ganske sikre på ville fungere. Det var å bruke Microsoft SQL Server som lagres lokalt på Pcen og den innebygde funksjonen i Microsoft Visual Studio til å generere rapport. Det var litt problematisk å få Microsoft SQL serveren oppe å gå, men når dette var gjort fungerte alt ganske fort

7.8.2 Løsning

Rapporten endte med å bli generert med ReportViewer som er et verktøy som lot oss generere rapport ut fra en database med Microsoft Visual Studio. Rapporten blir designet ved hjelp av Report Designer som er inkludert i Visual Studio versjonen vi bruker.

Det første vi fikk til å fungere var databasen. Vi installerte Microsoft SQL Server og lagde en database med to kolonner. Siden vi begynte å få dårlig tid så valgte vi bort de to statiske kolonnene som skulle vært med ”input” og ”forventet resultat”. Disse skulle være ferdig fylt inn og vi nedprioriterte å se noe på det. De to kolonnene som er i databasen nå er ”SeqID” og ”Result”. Der sekvens id blir fylt in automatisk for hver sekvens som kjøres og resultatet blir fylt inn når sekvensen er ferdig. Resultatet vil alltid være p, f eller ingen ting. P står for pass og blir fylt inn hvis sekvensen er godkjent i forhold til det forventede resultatet. F står for fail og blir fylt inn hvis sekvensen feiler. Hvis det er en tom rute så vil det si at det ikke er mulig å automatisere denne sekvensen og derfor blir den bare ”hoppet” over.

For å kunne lagre denne informasjonen noe annet sted en i databasen brukte vi ReportViewer som genererer en rapport ut ifra informasjonen i databasen. Dette blir automatisk generert etter at det er satt om en kommunikasjon mellom den lokale databasen og ReportViewer som er innebygd i Microsoft Visual Studio.

Til slutt lagde vi kode som lagrer rapporten til et bestemt sted lokalt på Pcen som kjører programmet. Rapporten vil også vises på skjermen når en test er ferdig utført.

Seq Id	Testresult
1	p
2	p
3	p
4	p
5	p
6	
7	p
8	p

Figur 6: Utseende på en rapport

8 Resultater

ACT systemet er et system som nå er i stand til å utføre de fleste funksjonstester automatisk. Hele systemet består av forskjellige komponenter og programvare. ACT systemet består av et hovedprogram som inneholder en kontroller som styrer hvilke metoder og funksjoner som får lov til å kjøre og holder oversikt over hvor langt programmet har kommet til enhver tid. Programmet har også en verifikasjonsdel som verifiserer tilstander til våpenstasjon og bruker bildegjenkjenning til å sammenligne bilder hentet ut fra DCP med bilder som inneholder korrekt informasjon. Det er enkelt å legge til nye tester siden programmet har metoder for å hente ut informasjon fra XML dokumenter og bruker informasjonen til å bestemme hvilen metoder som skal kjøres.

Programmet kommuniserer med våpenstasjon og robot over TCP/IP kobling og kan lese og sende informasjon over denne. Dette har gjort at systemet klarer å sende beskjeder om hva som skal gjøres til robot og få tilbake beskjeder når en oppgave er utført. I tillegg kan programmet hente ut informasjon eller bilder fra våpenstasjon som nevnt over, så dette kan brukes til å verifisere at alle tilstander er riktige og bildene sammenlignes.

Roboten har et eget program som er koblet opp mot hovedprogrammet som styrer systemet. Robotprogrammet inneholder alt som trengs for at roboten skal kunne utføre oppgaver som den mottar fra hovedprogrammet. Roboten vil gjøre det fysiske arbeidet i systemet.

ACT programmet klarer også å loggføre alle kommandoer som blir utført i en testsekvens. Denne vil da skrive en logg kontinuerlig under en test og lagre denne lokalt på Pcen som kjører programmet slik at en bruker lett kan finne denne og sjekke hva som har skjedd når en test har blitt utført. I tillegg er det en rapport funksjon som fungerer som en erstatning til den manuelle rapporten KPS fyller ut når de utfører tester manuelt. Rapporten i ATC blir generert ut ifra resultatene til hver test sekvens. Det vil si at når en sekvens er fullført vil det fylles inn et sekvens id og resultatet til sekvenses(p for pass eller f for fail). Når alle testsekvenser er utført vil den lage en layout på rapporten og lagre denne slik at en bruker lett kan se hvordan testresultatene ble.

Til slutt har vi noen resultater som ikke ble helt som forventet. ACT systemet mangler noen funksjoner når det kommer til feilhåndtering. Dette ble helt ferdig så det er ingen fungerende feilhåndtering til den leverte versjonen. Dette er noe som må gjøres ferdig hvis ACT systemet skal brukes til funksjonstesting. Det er ingen stor faktor at feilhåndteringen ikke er ferdig siden fokuset med denne oppgaven var å lage et rammeverk slik at KPS skal kunne ta en vurdering på om det er verdt å videreutvikle ACT systemet slik at de kan ta det i bruk.
Det eksterne måleutstyret skulle også vært med. Denne skal fungere som en dobbelt verifisering til kritiske deler av systemet, det vil si at den skulle kontinuerlig måle spenningen på solenoiden som har spenning hvis systemet fyrer og ikke har spenning hvis systemet ikke fungerer. Dette eksterne måleutstyret var et lite ”ekstra” prosjekt ved siden av bachelorprosjektet som vi håpet vi skulle få inn, men er ikke nødvendig for at prosjektet skal være vellykket.

Alt i alt så er ACT systemet et godt rammeverk for videre utvikling, og det fungerer til de testene vi har utført gjennom testperioden.

9 Konklusjon

Bachelorprosjektet har vært utrolig spennende og lærerikt. KPS ga oss ett prosjekt som har vært utrolig tidskrevende og utfordrende som har gjort at vi har lært mye nytt i form av nye programmeringsspråk og forskjellige programmer.

Resultatet ble veldig nære som planlagt. Det er en funksjon og et produkt som ikke er hundre prosent ferdig utviklet som mangler. Det er feilhåndtering og eksternt måleutstyr. Dette er noe som trengs for at det skal være en mulighet for at KPS kan bruke systemet til testing. Ellers fungerer systemet veldig bra og alle testene som er utført for å teste ACT systemet har fungert fint.

Vi ønsker å takke våre veiledere Dag Christian Nygaard og Daniel Larsson for støtte og god hjelp underveis i prosjektet. Vi ønsker også å takke alle personer fra KPS som har vært veldig behjelpelige til å lære oss diverse programmer og metoder som har gjort vi har klart å lage programmet.

Det er diverse som burde optimaliseres/forbedres som nevnt i Fremtidsdokumentet[1].

10 Referanser

REFERANSE	DOKUMENTTITTEL	VERSJON
[1]	Fremtidsdokument	1.0
[2]	Teknologidokument - XML	1.0

Tabell 3: Referanser

ACT AUTOMATED CROWS TESTING

FREMTIDSDOKUMENT

AUGUST KIND SVENDSEN
EIRIK LIEN ROA
HENRIK BERGE SØRUM
STÅLE RUDIN

KONGSBERG

HØGSKOLEN I BUSKERUD OG VESTFOLD · PROFESJONSHØGSKOLEN · HBV

Fremtidsdokument

PROSJEKT	ACT - Automated CROWS Testing		
OPPDRAGSGIVER	Kongsberg Protech Systems		
UTFØRT VED	HBV - Høgskolen i Buskerud og Vestfold		
GRUPPE	August Kind Svendsen, Eirik Lien Roa, Henrik Berge Sørum og Ståle Rudin		
VERSJON	1.0		
ANTALL SIDER	9		
DOKUMENTHISTORIE	VERSJON	UTGITT	BESKRIVELSE
	1.0	26.05.14	Endelig utgivelse

INNHOLDSFORTEGNELSE

LISTE OVER TABELLER

LISTE OVER FIGURER

1 Om dokumentet

1.1 Dokumenthistorie

VERSJON	DATO	ENDRING	SIGNATUR
0.1	16.05.14	• Dokumentet ble opprettet	AKS
1.0	24.05.14	• Dokument fullført	AKS

Tabell 1: Dokumenthistorie

1.2 Definisjoner og forkortelser

Liste som beskriver ord og uttrykk som kan være brukt i dette dokumentet:

UTTRYKK	FORKLARING
HBV	Høgskolen i Buskerud og Vestfold
KPS	Kongsberg Protech Systems (oppdragsgiver)
ACT	Automated CROWS Testing
Industrirobot	Universal Robots 5
CROWS	Common Remotely Operated Weapon Station
VS	Våpenstasjon
DCP	Display Control Panel
MPU	Main Processing Unit
Konfigurasjon	Forskjellige CROWS oppsett
Firing controller	Eksternt instrument som måler fyringsspenningen
CG	Control Grip

Tabell 2: Definisjoner og forkortelser

2 Innledning

Hensikten med dette dokumentet er å ta for seg eventuelle utvidelser eller forbedringer som kunne vært gjort i fremtiden. Dette kan være nyttige funksjoner som gjør systemet enklere, mer praktisk eller mer stabilt. Årsakene for at disse ikke er implementert er flere, men det som går igjen er nytten av funksjonen forhold til hvor mye tid det krever, samt at ideen for funksjonen ofte har dukket opp underveis i utviklingen. Mange av funksjonene drøftet her vil være «quality of life» funksjoner som ikke trenger å være nødvendig for at systemet skal fungere, men det vil heve brukervennligheten og mulighetene for brukeren av programmet.

3 Sammendrag

Det er mulig å forbedre og optimalisere flere aspekter ved dette systemet, og vi har mange ideer på hva og hvordan dette kan gjøres. Når det gjelder roboten kunne først og fremst verktøyet montert på enden av armen vært mer egnet. Siden dette blir utsatt for repetive fysiske påkjenninger vil det nok tenkes at det utvikles et nytt og mer robust verktøy i fremtiden. Hastigheten og akselerasjonen på robotbevegelsene kunne også vært gjort raskere, som kan spare betydlig tid i det lange løp.

Hvordan data blir håndtert i databasen etter endt test kunne vært gjort om slik at gammel data fra tidligere tester blir lagret og dokumentert. Nå har vi en rapport som tar vare på data fra testen, men selve tabellen som fylt ut i løpet av en test blir gjort tom.

Vi kunne også tenkt oss at systemt blir så godt at tredjeparts programvare ikke lenger ville være en nødvendighet. En dedikert ACT server ville også gjort oppsettet av systemet enklere, i motsetning til hvordan det er med lokal database.

Tekstuthentingen ville åpnet for flere muligheter innen verifisering og dokumentering, og vi har trua på at dette vil være mulig å realisere en gang i fremtiden.

En ide relatert til hvordan brukeren lager tester i XML er en applikasjon eller funksjon i programmet som vil assistere denne prosessen.

4 Forbedre robotverktøyet

Ytterst på robotarmen er det festet et tilpasset verktøy som KPS hadde satt sammen før vi startet på prosjektet. Dette verktøyet er det som skaper den fysiske kontakten mellom robot og våpenstasjon, det vil si trykker på knappene på DCP-en og manipulerer CG-en.

Når vi startet med å teste systemet vårt viste det seg at dette verktøyet ikke ville holde. Under bruk oppstår det press på den ene fingeren på verktøyet som fører til at den blir vridd, sånn at den ikke lenger peker dit den skal. Når en da prøver å bruke denne fingeren til noe annet, så vil ikke roboten treffe målet.

Figur 1: Robotverktøy

I samarbeid med vår eksterne veileder har vi fått laget en mer robust løsning som vil holde til vårt bruk, men i fremtiden bør det vurderes å lage en helt ny løsning på dette. Vi byttet posisjon på fingrene og strammet til med stoppskiver, i tillegg til å gjøre om måten den brukte CG-en. Før ville presset presse i samme retning som vil løse opp fingrene, men nå er det motsatt.

Det beste ville vært å ha et verktøy som ikke har mulighet til å vende på seg på grunn av press. Da måtte hver finger enten låses på fler enn et punkt, eller vært sammenkoblet slik at en vending ikke ville vært mulig.

Et annet forbedringspotensial relatert til roboten er at den skal kunne lage forskjellige plan på egenhånd. Planene er parallelle med DCP-en, og brukes til å koordinere roboten når den skal gjøre bevegelser. En mer detaljert beskrivelse av robotens plan er beskrevet i teknologidokumentet for robot[2]. Løsningen kunne vært ved å få den til å trykke i hvert av hjørnene på DCP-en, og brukt dette som referansepunkter for et plan. Da kunne den brukt DCP-er i forskjellig tykkelse, uten at det manuelt må lages et plan hver gang.

5 Optimalisere robotbevegelser og ventetid

Vi føler at vi har laget gode bevegelsesscripts til roboten, men det finnes rom for forbedring. En har nemlig muligheten til å justere hastighet og akselerasjon på bevegelsene til roboten. Hastigheten vi har gått for er hverken spesielt rask eller langsom, men en hastighet som vi føler er kontrollert og enkel å holde styr på. Men det er snakk om en robot i ypperste kvalitet, så det burde være mulig å skru opp både hastighet og akselerasjon uten at det skulle påvirke presisjonen. I fremtiden kan dette eksperimenteres på, for å finne eventuelle ulemper eller fordeler med en annen hastighet, sett bort i fra at ting blir gjort raskere.

Når det gjelder ventetiden så er også dette et punkt vi ikke har optimalisert. Mellom enkelte kommandoer må det verifiseres og dokumenteres at hendelsen har funnet sted. Derfor har vi blitt nødt til å legge inn punkter der eksekveringer må «sove / satt i pause» for å vente på at verifiseringen skjer. Hvor lang tid dette tar varierer fra verifisering til en annen, men vi har alltid satt på litt ekstra tid for å være sikre. Denne tiden kunne sikkert blitt justert slik at en hel test totalt sett kunne kanskje spart noen minutter.

6 Databasebehandling

Slik det står nå så vil data som lagres i databasen bli slettet dersom en ny test startes. Det ville vært ønskelig å bevare denne dataen, og hvis en faktisk ville slette noe så ble det gjort manuelt. For å ta vare på testdata kunne det vært slik at istedenfor å gjenbruke den samme tabellen i databasen, at det heller ble lagd en ny tabell. Eventuelt at databasen med resultater ble lagret som en spørring i en tekstfil.

En utfordring som kunne oppstått med nye tabeller er hvis f. eks. brukeren stopper testen, og etterpå prøver å starte fra en bestemt sekvens. Da måtte en også passet på at den ikke startet på en ny tabell, eller begynte å overskrive resultater fra tidligere testsekvenser, men fortsatte der den slapp.

7 Database på ekstern server

Under prosjektarbeidet har vi ikke hatt tilgang til en server som vi kunne brukt for å holde på databasen. Vi har derfor blitt nødt til å benytte oss av en lokal database som kjører på samme datamaskin som resten av programvaren. Dette er ikke helt ideelt fordi for å kjøre en lokal database trenger en ekstra programvare, i dette tilfellet «Microsoft SQL Server Management Studio». Det er ikke store arbeidet å laste ned og installere dette, men det er ekstraarbeid som vi ønsker at brukeren helst kunne vært foruten. Fremgangsmåte for å installere og sette opp slik som det er nå er forklart stegvis i Installasjonsguiden[1].

Det å omgjøre ACT systemet slik at det bruker en database som kjører et annet sted vil ikke være mye arbeid. Det vil holde med å bytte spørringen der den kobler seg til. Så om KPS noen gang ønsker å gjøre databaseoppsettet mer brukervennlig, så bør en server løse dette uten større utfordringer.

8 Uavhengighet fra FileZilla

En metode vi bruker for å verifisere feilmeldinger er å hente ut bildet som vises på DCP-en og så lagre det som et .png bilde. For å gjøre dette trenger en på forhånd å installere programmet «FileZilla Server Interface». Vi bruker FileZilla for kunne laste opp og ned filer mellom våpenstasjonen og datamaskinen.

Figur 2: FileZilla

For å kunne hente ut skjermdump fra våpenstasjonen trenger en først å laste opp noen filer KPS har utviklet. Disse blir borte etter en skrur av våpenstasjonen, så en er nødt til å laste opp disse for hver gang. Programmet vårt har en metode som vil gjøre dette automatisk dersom en har konfigurert den delte mappa og filene riktig. Det vil si at en oppretter en delt mappe som FileZilla vet om, der en legger filene «capture.o», «capturedh.o» og «capture_vxworks55.o» (filene lagd av KPS som gjør bildeuthenting mulig).

Dette er ikke vanskelig å sette opp og ligger veldig tilrettelagt dersom en allerede har gjort det en gang. Likevel strider det litt i mot enkelheten vi har ellers i programmet. Ønskelig hadde det vært at en kun trengte ACT programvaren for å kunne gjennomføre testen, men for øyeblikket virker det som vi er avhengig av tredjeparts programvare. Et alternativ er å kun bruke dataverifiseringen ved å deaktivere bildeverifiseringen dersom en har problemer med å sette opp FileZilla.

9 Tekstuthenting og sammenligning

Planen var først å kunne hente ut selve teksten på bildet, slik at vi kunne lagre den informasjonen i en string og deretter sammenligne. Det viste seg å være mer problematisk enn forventet så etter utallige forsøk gikk vi vekk fra dette, men vi vet det skal være mulig. Derfor endte vi heller opp med å sammenligne bildet vi hentet ut med et bilde som vi har tatt på forhånd. Så bruker vi en algoritme for å finne likheten mellom disse bildene ved å sammenligne pikslene. Dette er en god løsning som fungerer godt og gir resultater en kan stole på.

Med tekstuthentingen kunne vi likevel fått flere muligheter med hvordan vi behandler dataen etter verifiseringen. Vi kunne brukt teksten andre steder i programmet, for eksempel i hendelsesloggen dersom noe ikke bestod verifiseringen. Da kunne brukeren raskt lese hva som ble hentet ut, og hva som var forventet skulle stå. Slik det er nå er blir vi nødt til å referere til et bilde, der brukeren må bla seg frem til det for å sjekke.

10 Starte på bestemt sekvens

Hvis det skulle oppstå en situasjon der brukeren må avbryte testen og ønsker å starte fra samme punkt etterpå, så er det for øyeblikket ikke mulig. Selv om det teoretisk sett skal være enkelt å implementere så har ikke dette vært en funksjon som har vært prioritert å få til i første omgang. Likevel ser vi at dette kan være nyttig, spesielt om en er et stykke ute i testen og blir nødt til å stoppe. Da vil en kunne spare mye tid hvis det var mulig å velge startsekvens ved omstart.

En utfordring relatert til dette er hva som vil skje med data som blir lagret i databasen. Hadde vi bare startet på en vilkårlig sekvens etter å ha kjørt noen andre først vil de gamle dataene overskrives og mest sannsynlig vil ikke sekvensnummer stemme overens lenger. Men så lenge en er oppmerksom på dette så skal en implementering av denne funksjonen være fult mulig.

11 Hjelpeapplikasjon for XML-testene

Selv om det er relativt enkelt å lage en test som systemet kan bruke ønsker vi å gjøre det enda enklere. Dette kunne vært realisert ved å lage en funksjon i programmet, eventuelt en egen applikasjon, som hjelper brukeren med dette. I stedenfor å bruke notepad, der en må passe på riktig syntaks for hver sekvens og kommando, kunne vært gjort enkelere med litt hjelp fra programvare. Tanken er at brukeren enkelt kan dragge og droppe kommandoer fra en liste, der attributter blir autogenerert og alle tags blir åpnet og lukket automatisk. Det kunne også kommet opp en rask beskrivelse av den valgte kommandoen slik at brukeren kunne forsikre seg om riktig valg av kommando.

Det som kan være en utfordring for ferske brukere av systemet slik det er nå er det å sette kommandoene i riktig rekkefølge, spesielt når det kommer til verifiseringen. Dette kunne vært løst i hjelpeapplikasjonen ved at en får valget om å huke av en boks ved siden av kommandoen dersom en ønsker at det skal bli verifisert.

12 Flere feilhåndteringer

I kravspesifikasjonen[3] har vi kravet RK.2 som sier «Systemet skal håndtere feilsituasjoner på en forutsigbar og sikker måte.» Dette er et nokså åpent og stort krav som vi føler at det kunne vært arbeidet mer på. Grunnen til at dette kravet ikke er oppnådd slik vi ønsker er på grunn av det krever et nokså ferdig system. For å håndtere feilene på en god måte trenger alle systemets komponenter å reagere på en kontrollert og sikker måte. Dette vil være problematisk før en har et bortimot ferdig produkt. Likevel la vi et godt grunnlag da vi implementerte feilhåndteringen. Vi klarte å detekere feil, og kunne deretter håndtere de. Det var integreringen med resten av programmet som var utfordringen.

Et annen relatert utfordring det krevende arbeidet med å oppdage de feilene som kan oppstå. Det vi satte i fokus i den perioden vi rakk å jobbe med feilhåndtering var å legge inn en konstant sjekk av fyring. Dette mener vi vil være en ekstremt viktig feil å detektere dersom våpenstasjonen skulle finne på å fyre når den ikke skal. Vi kom ikke helt i mål i arbeidet med denne utfordringen, men fikk startet på noe som vil være mulig fult å utvikle videre.

13 Konklusjon

Som det kommer frem i dette dokumentet er det mange muligheter med tanke på utvikling av systemet. Nye funksjoner kan legges til, nåværende kan forbedres, og det er rom for å gjøre brukervennligheten bedre. Flere av funksjonene skal være nokså enkle å få på plass, mens andre vil nok kreve mer arbeid. Heldigvis for oss så har arbeidsgiver gitt oss muligheten til å gjøre disse forbedringene som en sommerjobb, noe vi alle takket ja til.

14 Referanser

REFERANSE	DOKUMENTTITTEL	VERSJON
[1]	Installasjonsguide	1.0
[2]	Teknologidokument - Robot	1.0
[3]	Kravspesifikasjon	2.0

Tabell 3: Referanser

ACT AUTOMATED CROWS TESTING

ETTERANALYSE

AUGUST KIND SVENDSEN
EIRIK LIEN ROA
HENRIK BERGE SØRUM
STÅLE RUDIN

KONGSBERG

HBV
HØGSKOLEN I BUSKERUD OG VESTFOLD · PROFESJONSHØGSKOLEN ·

Etteranalyse

PROSJEKT	ACT - Automated CROWS Testing		
OPPDRAGSGIVER	Kongsberg Protech Systems		
UTFØRT VED	HBV - Høgskolen i Buskerud og Vestfold		
GRUPPE	August Kind Svendsen, Eirik Lien Roa, Henrik Berge Sørum og Ståle Rudin		
VERSJON	1.0		
ANTALL SIDER	12		
DOKUMENTHISTORIE	VERSJON	UTGITT	BESKRIVELSE
	1.0	26.05.14	Første utgivelse

INNHOLDSFORTEGNELSE

LISTE OVER TABELLER

LISTE OVER FIGURER

1 Om dokumentet

1.1 Dokumenthistorie

VERSJON	DATO	ENDRING	SIGNATUR
0.1	03.05.14	• Dokumentet ble opprettet	AKS
0.2	24.05.14	• Omstrukturering av innhold	AKS
1.0	25.05.14	• Dokument fullført	AKS

Tabell 1: Dokumenthistorie

1.2 Definisjoner og forkortelser

Liste som beskriver ord og uttrykk som kan være brukt i dette dokumentet:

UTTRYKK	FORKLARING
HBV	Høgskolen i Buskerud og Vestfold
KPS	Kongsberg Protech Systems (oppdragsgiver)
ACT	Automated CROWS Testing
Industrirobot	Universal Robots 5
CROWS	Common Remotely Operated Weapon Station
RWS	Remote Weapon Station
DCP	Display Control Panel
MPU	Main Processing Unit
Konfigurasjon	Forskjellige CROWS oppsett
Firing controller	Eksternt instrument som måler fyringsspenningen
CG	Control Grip

Tabell 2: Definisjoner og forkortelser

2 Introduksjon

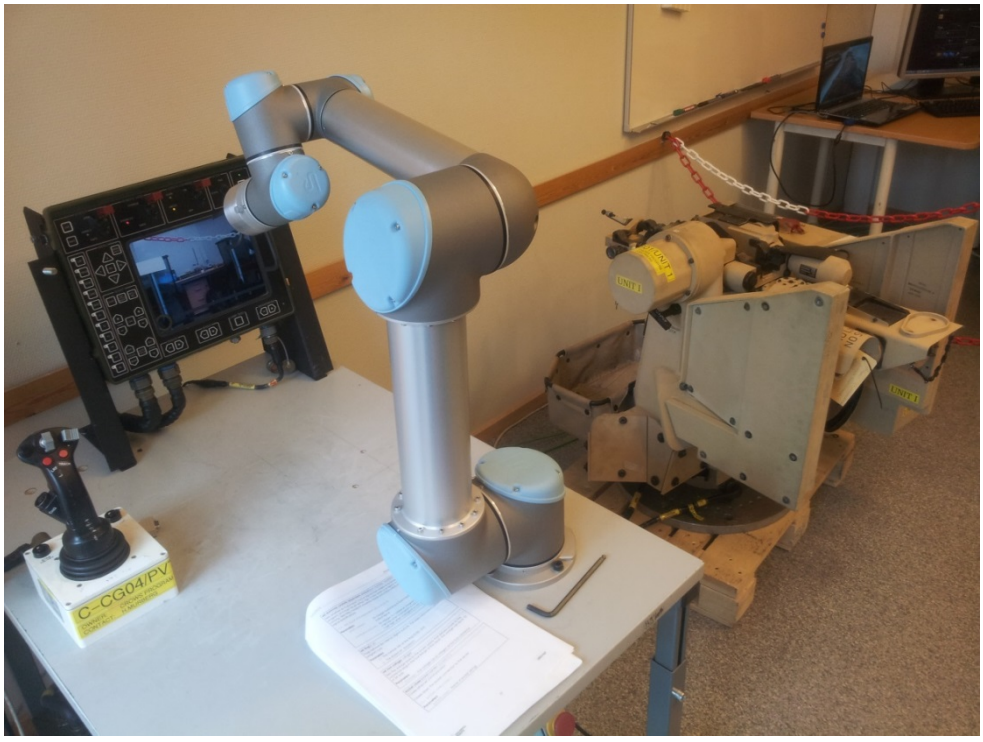
Etter endt prosjektarbeid har vi utarbeidet et etteranalysedokument. Her reflekterer prosjektgruppa på individuelt og kollektivt grunnlag over det arbeidet vi har vært med på og de erfaringene vi har fått ut av arbeidet.

2.1 Innledning

I utgangspunktet var det gruppeleder Eirik Lien Roa som via arbeidsplassen sin fikk vite om dette hovedprosjektet. Han har personlig erfaring fra Kongberg Protech Systems på hvor repetive og tidkrevende de manuelle testene kan være. Å automatisere testprosessen på våpenstasjonene har vært noe KPS har ønsket en god stund, men hvordan dette skulle løses har aldri vært helt sikkert. Derfor fikk han tilbudet om å ta denne utfordringen som et hovedprosjekt. Høsten 2013 samlet han en liten gjeng med datastudenter som senere skulle utgjøre prosjektgruppa.

Selve problemstillingen var klar, vi skulle automatisere testprosessen på CROWS ved hjelp av en industrirobot, dog veien dit var det ingen som visste. En ting var sikkert, arbeidet ville kreve mye datakunnskaper, perfekt for en gjeng som snart var ferdig med dataingeniørlinja. Ambisiøse og fulle av selvtillit gikk vi til verks på oppgaven.

Vi ble imponert over utstyret vi fikk presentert i oppstartsfasen. KPS nølte ikke med å gi oss vårt eget kontor med plass til alle fire og med kort vei til kaffemaskinen. Og inn på dette kontoret fikk vi til og med vår egen våpenstasjon og det utstyret som hører til. Ved siden av våpenstasjonen hadde vi roboten vi skulle jobbe med. En UR5 fra Universal Robots. Vi følte oss privilgerte etter å ha fått tilgang til alt dette, og det klødde i fingrene etter å starte.



Figur 1: Utstyrs oppsett

3 Prosjektevaluering

3.1 Veiledere

I starten av prosjektet støtet vi på litt problemer i utdelingen av intern veileder. Vi hadde avklart med oppdragsgiver at vi skulle levere prosjektet på norsk, og da vi fikk utdelt en veileder som ikke forstod norsk hadde vi en liten utfordring. Dette ville sansynligvis endt i et vanskeligere samarbeid veileder, så vi spørte derfor HBV om vi kunne få tildelt en ny. Denne prosessen tok lengre tid enn forventet og vi gikk muligens glipp av litt veiledning tidlig i prosjektfasen. Heldigvis ordnet dette seg tilslutt og den interne veilederen vi endte opp med har bidratt med det han kan hele veien. Ingen på gruppa har særlig erfaring med prosjektarbeid på denne størrelsen, så det har derfor vært veldig kjekt å ha en veileder som kan hjelpe oss med prosjektstyring. Intern veileder har også vært til assistanse når noe av det formelle vært uklart, noe som har vært fint at gruppa slapp å bruke tid på.

Vår eksterne veileder hadde sitt hovedprosjekt for ikke så lenge siden, og har hjulpet oss ved å dele erfaringer han hadde fra sitt prosjekt. I tillegg til dette har både han og andre ansatte ved KPS alltid vært til hjelp om noe teknisk ved prosjektet har vært vanskelig eller uklart. Uten denne veiledningen ville prosjektet blitt betydligere vanskeligere.

3.2 Arbeidsgiver

Som nevnt innledningsvis har Kongsberg Protech Systems vært veldig støttende og lette å kommunisere med hele veien. Fra starten av var de raske med å gi oss de midlene vi trengte for å komme i gang med prosjektet. De har vært veldig åpne for hvordan vi kunne løse prosjektet, men likevel vært klare for hva de ønsker å få ut av det hele. I kravsettingsmøtene kom de tydlig frem med kravene de hadde til produktet, og lot oss være med å definere dem.

3.3 Nytte av skolekunnskap

Gjennom prosjektet har vi fått nytte av flere ting som vi har tidligere lært på skolen i andre fag. Selvom vi ikke endte opp med å programmere i et språk vi har lært, så sitter vi på nok generell programmeringskunnskap til at vi tok språket C-Sharp uten særlige problemer. Og med kunnskapene vi har i UML kunne vi bruke dette til å visualisere produktdesignet i planleggingsfasen slik at vi var sikre på at vi jobbet mot det samme målet. Vi har også tatt nytte av det vi har lært om databaser, noe vi brukte som et viktig verktøy for å loggføre resultatene i ACT systemet.

Likevel skal det nevnes at vi har støtt på flere temaer der vi har hatt lite til ingen kunnskap. Flere tekniske temaer har vært nye for oss, og det har godt av mye tid på å bare lese seg opp på dette. Å lage og bruke XML-ark, bildesammenligning, TCP/IP-kommunikasjon, scripting av robot, parsing med regex og trådbehandling er bare noen av temaene vi tidligere var nokså grønne på, men nå kan si at vi kan en hel del. Det er nok det som er noe av meningen med dette hovedprosjektet. Å analysere en oppgave, gjøre research, og deretter velge ut en passende løsning er en prosess alle på gruppa har vært i gjennom flere ganger.

3.4 Prosjektmodell

Prosjektmodellen skulle vise seg å gi oss litt mere bry enn forventet. Ser vi tilbake la vi nok ikke nok tanke i hvor stor rolle denne egentlig ville spille i løpet av prosjektet. Har man en egnet prosjektmodell kan den gjøre arbeidet enklere, oversiktlig og mer kontrollert. Vi hadde egentlig bare krav om at vi skulle ha muligheten til å gå tilbake om nødvendig, og med en grov inndeling av forskjellige faser. En vannfallsmodell med tilbakesløyfing passet overens med disse, og vi endte opp med denne.

Under vår første presentasjon pekte mye av kritikken mot prosjektmodellen vi hadde valgt. Det gikk opp et lys for oss hvor betydlig denne egentlig var for prosjektet vårt, og vi bestemte oss for å undersøke grundig og finne en mer passende. Det var da vi kom over Unified Process modellen som viste seg å være akkurat det vi lette etter. Det vi spesielt likte med denne var at de forskjellige disiplinene eller aktivitene gikk over hele prosjektperioden, men der hvor fokuset på dem varierte med fasen og iterasjonen man befant seg i. Dette ville hjelpe oss med å vite hva vi burde jobbe med når og hvor mye ressurser som burde settes av til de forskjellige aktivitene.

I Unified Process hører det med å skrive en plan og rapport for de forskjellige iterasjonene. Dette har hjulpet oss å planlegge arbeidet bedre når den aktuelle iterasjonen har nærmet seg. I etterkant har vi da skrivd en rapport som dokumenterer hvor mye som gikk etter planen, tiltak på å ordne eventuelle ting som ikke gikk som det burde, og utfordringer vi møtte på underveis. Dette har vi brukt til å analysere og forbedre oss ettersom vi går fra iterasjon til iterasjon.

En av grunnene til at vi ikke la så mye tanke i hvilken prosjektmodell vi valgte første gangen er også på grunn av erfaring. I tidligere prosjektarbeid har vi bort i mot aldri brukt noen form for prosjektmodell, i hvert fall ikke der den har spilt en betydelig rolle. Og utenom den korte innføringen vi fikk i oppstarten så satt vi på lite kunnskap om hvilke prosjektmodeller som fantes der ute, og hva som gjorde at enkelte egnet seg mer enn andre. Men igjen, dette er kunnskap alle i gruppa sitter på nå og god lærdom å ta med videre.

3.5 Prosjektarbeid

Veien fra prosjektet startet til ferdig produkt kan sees på som å ha vært både lang og kort. Lang i den forstand at dette har vært et veldig tidkrevende prosjekt som har stått i fokus bortimot hver dag dette halvåret. Og kort på en måte der det føles at det har vært mye som forventes at vi får til på dette knappe halvåret. Likevel må det sies at vi har lært ekstremt mye i løpet av denne tiden. Ikke bare nye tekniske ting, men også ting som hvordan det er å gjennomføre et stort prosjekt fra start til slutt og erfaring med hvordan det kan være å jobbe som en ingeniør i en bedrift. Dette er erfaringer som en ikke kan tilegne seg ved skolebenken og som vi er veldig takknemmelige for at vi sitter igjen med.

Vi er alle klassekamerater fra før av, og i tillegg til å ha jobbet mye sammen tidligre, så ser vi hverandre mye utenom skoletid og. En risiko når vi kjenner hverandre så godt og skal jobbe sammen er at fokuset lett kan gli over til andre ting. Selvom dette har hendt til tider, har vi likevel vært flinke og klart å holde fokus. Grunnene til dette kan være flere, men vi tror siden alle har hatt samme mål om å gjøre det bra på dette prosjektet så har det fungert. Vi tror også at siden vi kjenner hverandre så godt så har det vært enklere å gi hverandre tilbakemeldinger og konstruktiv kritikk, og kommunikasjonen innad i gruppen har alltid vært veldig god. Dette er sentrale egenskaper for at en gruppe skal fungere godt sammen, og det har det gjort for oss.

Ansvarsfordelingene vi delte ut i starten av prosjektet har holdt seg godt gjennom prosjektperioden. Vi føler at vi fikk en jevn fordeling av oppgaver, og uventede oppgaver som har dukket opp har vi ikke hatt problemer med å fordele. Selv om hver aktivitet har en hovedansvarlig, så har alltid resten av gruppa vært til hjelp dersom den ansvarlige har trengt det, noe som har hendt på bortimot hvert eneste dokument og aktivitet.

Selve oppgaven har vært utfordrende, men har egnet seg veldig til oss som gruppe. Mesteparten av oppgaven har vært programmering eller en form for koding, noe som har vært stort fokus i løpet av skolegangen tidligere. Vanskelighetsgraden totaltsett har også vært midt i blinken. Vi regnet med at noe ville gjenstå ved endt prosjekt, men endte opp med å klare å implementere flere funksjoner enn forventet. Det er mulig det kunne vært med en elektrostudent som kunne hjelpet oss med det eksterne måleinstrumentet, men siden dette har vært en oppgave litt på siden av resten av prosjektet, så har det nok vært like greit med oss fire.

4 Produktet

Etter mye arbeid sitter vi igjen med produkt vi er godt fornøyd med. Arbeidsgiver sa i starten av prosjektet at de gjerne ønsket å se et godt rammeverk for systemet, og ikke nødvendigvis et komplett produkt. Vi mener vi trygt kan si at vi har et godt fundament for videre utvikling, og vel så det. På den korte tiden vi hadde til utvikling har vi laget et produkt som kan utføre enkle funksjonstester på arbeidsgivers våpenstasjoner. I tillegg klarer systemet å både loggføre aktiviteter og verifisere og loggføre testresultater. Og det hele kan organiseres fra et dokument lagd i notepad på få minutter.

Selvom vi fikk til mye og er stolte av det, så er det likevel noen av kravene som ikke ble oppfylt. To av kravene som vi ikke fikk til er relatert til håndtering av feilsituasjoner (RK.2 og SK.1). Dette er krav som er vanskelig å realisere før systemet nærmer seg en ferdig tilstand der resten av funksjonene er på plass. Det er et omfattende produkt, og som sagt var arbeidsgiver klare fra starten av på at de helst så oss legge et grunnlag for noe som kunne videreutvikles. Vi har hatt dette i bakhodet hele veien, men likevel jobbet vi etter kravene for et fult produkt. Dette har vært positivt for utviklingen for det har gitt oss noe å strekke oss etter.

Et annet krav omhandlet måling av fyringsspenningen på våpenstasjonen. For å kunne måle denne spenningen trengte vi et spesiallagd måleinstrument. Vi startet arbeidet på dette instrumentet med hjelp av vår eksterne veileder, og kom langt med designet. Men det viste seg at dette ville bli for mye arbeid med tanke på hvor viktig det egentlig var. I tillegg er dette arbeid som grenser utenfor det vi datastudenter har ferdigheter til.

Vi kan ikke regne med at KPS vil bruke systemet slik det er nå til testing av deres våpenstasjoner, for her er det høye krav til sikkerhet og kvalitet. Det vil likevel ikke kreve mye arbeid før alle kravene til produktet er møtt, og da vil det være en mulighet for at ACT systemet blir standard prosedyre for testing av CROWS og kanskje flere typer RWS.

5 Presentasjoner

Det er under presentasjonene vi får muligheten til å vise oss frem og skape et godt inntrykk av hva vi som gruppe har fått til under prosjektet. På disse presentasjonene sitter det både utenforstående og personer med påvirkning på hvordan karakter vi vil ende opp med. Derfor har disse tre presentasjonene vært veldig viktige for oss, og vi har lagt mye fokus på at disse skal bli så gode som mulig.

Om vi ser tilbake så kunne vi nok forbedret oss her ved å gjøre de to første presentasjonene mer spennende. Vi er vandt med forelesninger der lærdom blir lagt mer vekt på enn underholdningsaspektet, og denne presentasjonsformen kan ha smittet over til oss. I dette prosjektet gjelder det egentlig å fange publikum mest mulig for å imponere sensorer og publikum så godt vi kan på de få minuttene vi har. Dette gikk opp for oss før siste presentasjon, og vi håper at vi vil kunne gjøre den presentasjonen minneverdig.

6 Risikohåndtering

Under prosjektperioden har det vært få hendelser som har hindret prosjektets utvikling. Det var først i siste innspurt at vi støtet på to nevneverdige utfordringer. Det første som skjedde var at en på gruppa sølte et glass vann over laptoppen sin. Det resulterte i at tastaturet ikke ville fungere som normalt, ved at det ga tilfeldige tastetrykk til datamaskinen. I starten trengte vi ikke å gjøre noe for tastaturet ville oppføre seg normalt 90% av tiden, men det skulle vise seg å bli verre. Tilslutt måtte den utsatte bruke veldig lang tid på å skru maskinen på, men når den først var på kunne vi deaktivere det innebygde tastaturet og koble til et annet.

Det andre uheldige som inntraff var at verktøyet på roboten viste seg å ikke tåle påkjenningene, og dette hendte rett før vi skulle starte med testingen. Det ante oss ganske tidlig i prosjektet at verktøyet muligens ikke var robust nok ved at pinnene kanskje ville bli derformerte etterhvert. Det som faktisk skjedde var ikke at pinnene ble deformerte, men ville vri seg i festet. Sammen med vår eksterne veileder ordnet vi dette så raskt vi kunne ved å lage en stødigere versjon av verktøyet. Dette er nøyere beskrevet i Fremtidsdokumentet[2]. Konsekvensen hendelsen var at personen som skulle jobbe videre med det eksterne måleutstyret fikk mindre tid til det, men vi er rimelig sikre på at det utstyret ikke ville blitt ferdig uansett.

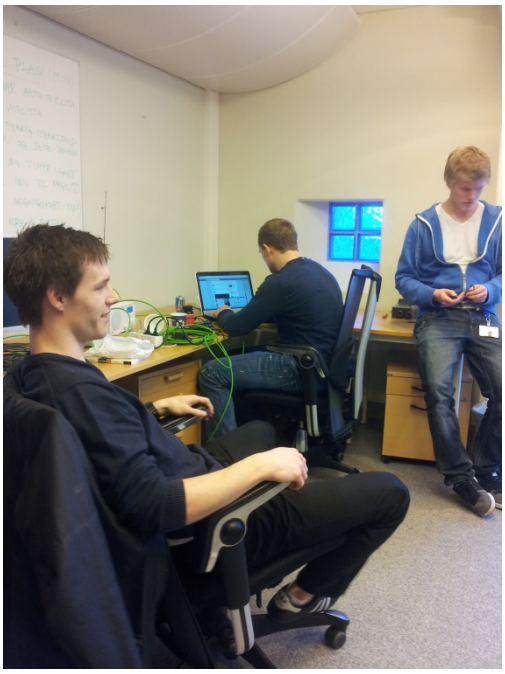
7 Økonomi

I prosjektplanen estimerte vi at det ville gå med en del penger til servering og printing av dokumentasjon og plakat. Vi har vært heldige for KPS har tilbudt seg å spandere kostnadene for printingen av dokumentasjonen og plakaten, noe en gjeng med studenter ikke kan takke nei til. I gjengjeld ønsker de å få en ekstra kopi av dokumentasjonen som de kan bruke senere. Siden vi slipper å betale for plakat og dokumentasjon så har vi tenkt å bruke litt ekstra på en fin kake til siste presentasjon.

Ellers har kostnadene for prosjektet vært nokså lave. Det blir brukt litt på komponenter til det eksterne måleinstrumentet, men siden dette har vært noe som har vært litt på grensa til prosjektet vårt, så arbeidsgiver tok seg av dette.

8 Siste ord og tanker

Kort sagt er prosjektgruppa veldig fornøyd med resultatet av oppgaven. Vi har lagd et produkt som vi er veldig stolte av, og i løpet av prosjektet har vi tilegnet oss mange nyttige erfaringer og ny kunnskap. Vi har virkelig fått testet oss på forskjellige fronter i løpet av denne oppgaven, og en kan si at vi sitter igjen som noen smartere og mer erfaringsrike studenter enn ved start.



Figur 2: Tre av gruppemedlemmene

8.1 Siste ord fra gruppemedlemmene

August Kind Svendsen: Som forfatteren av dette dokumentet har nok flere av mine refleksjoner satt sitt preg på innholdet, og som det sikkert kommer frem har jeg satt stor pris på å jobbe med dette prosjektet. Produktet vi har jobbet med har vært spennende, gruppemedlemmene har vært en fornøyelse å jobbe med, og nå kjenner jeg en lettelse og mestringsfølelse ettersom denne krevende oppgaven nærmer seg en slutt.

Eirik Lien Roa: Jeg syntes dette prosjektet har vært veldig tidkrevende og utfordrende, men samtidig ekstremt lærerikt. Jeg syntes produktet vi har jobbet med har vært veldig interessant, spennende og avansert. Det har vært kult å få være med å utvikle et system som samarbeider med og er laget opp mot RWS som er KPS sitt produkt. Det at dette systemet kan være veldig nyttig for KPS gjør det ekstra moro og spennende.

Som prosjektleder syntes jeg gruppa har fungert veldig bra. Alle medlemmene har stått på og bidratt masse, samtidig som det har vært veldig god kjemi. Med tanke på gruppa, stemning og arbeidsinnsats har dette prosjektet gått helt smertefritt.

Henrik Berge Sørum: Jeg syntes prosjektet har vært både tidskrevende og spennende. Det har vært mange forskjellige deler som skal kommunisere med hverandre noe som har gjort prosjektet utfordrene. Jeg føler selv at resultater vi har oppnådd er et bra utgangspunkt for KPS til å videreutvikle systemer, og jeg er bestemt på at mange av testene lar seg automatisere på en god og sikker måte. Ellers har prosjektgruppen vært bra å jobbe med, alle har gjort sine oppgaver og vi har vært behjelpelige med hverandre når dette har vært nødvendig. Alt i alt så har det vært et supert prosjekt som har lært meg masse nytt, både om programmeringsspråk og andre kunnskaper.

Ståle Rudin: Dette prosjektet har vært en lang og hard prosess, men nå nærmer slutten seg. Jeg syntes vi var veldig heldige som fikk en så interessant oppgaven fra KPS. Særlig gøy å kunne jobbe med enheter som man faktisk ser at beveger seg. Ellers føler jeg kjemien ikke kunne vært bedre innad i gruppa. Alt i alt er jeg veldig fornøyd med bachelor oppgaven vår.

Helt til slutt vil vi si takk til:
Karoline Moholth, for sin jobb som intern sensor.
Hallvard Murberg, for sin jobb som ekstern sensor.
Dag Christian Nygaard, for jobben som ekstern veileder.
Daniel Larsson, for jobben som intern veileder.
Michael Odden, for teknisk assistanse.
Joseph Piperakis, for teknisk veiledning.
Merete Hjelsvold, for hjelp med kravsetting og definering av prosjekt.
Joakim Bjørk, for hjelp med UML-diagrammer.

9 Referanser

REFERANSE	DOKUMENTTITTEL	VERSJON
[1]	Kravspesifikasjon	3.0
[2]	Fremtidsdokument	1.0

Tabell 3: Referanser

ACT AUTOMATED CROWS TESTING

INSTALLASJONSGUIDE

AUGUST KIND SVENDSEN
EIRIK LIEN ROA
HENRIK BERGE SØRUM
STÅLE RUDIN

KONGSBERG

HBV
HØGSKOLEN I BUSKERUD OG VESTFOLD · PROFESJONSHØGSKOLEN ·

Installasjonsguide			
PROSJEKT	ACT - Automated CROWS Testing		
OPPDRAGSGIVER	Kongsberg Protech Systems		
UTFØRT VED	HBV - Høgskolen i Buskerud og Vestfold		
GRUPPE	August Kind Svendsen, Eirik Lien Roa, Henrik Berge Sørum og Ståle Rudin		
VERSJON	1.0		
ANTALL SIDER	38		
DOKUMENTHISTORIE	VERSJON	UTGITT	BESKRIVELSE
	1.0	26.05.14	Første utgivelse

INNHOLDSFORTEGNELSE

LISTE OVER TABELLER

1 Om dokumentet

1.1 Dokumenthistorie

VERSJON	DATO	ENDRING	SIGNATUR
0.1	18.05.14	• Dokumentet ble opprettet	SR
1.0	26.05.14	• Første utgivelse	SR

Tabell 1: Dokumenthistorie

1.2 Definisjoner og forkortelser

Liste som beskriver ord og uttrykk som kan være brukt i dette dokumentet:

UTTRYKK	FORKLARING
HBV	Høgskolen i Buskerud og Vestfold
KPS	Kongsberg Protech Systems (oppdragsgiver)
ACT	Automated CROWS Testing
UR5	Universal Robots 5
CROWS	Common Remotely Operated Weapon Station
RWS	Remote Weapon Station
DCP	Display Control Panel
MPU	Main Processing Unit
Konfigurasjon	Forskjellige CROWS oppsett
Firing controller	Eksternt instrument som måler fyringsspenningen
CG	Control grip

Tabell 2: Definisjoner og forkortelser

2 Innledning

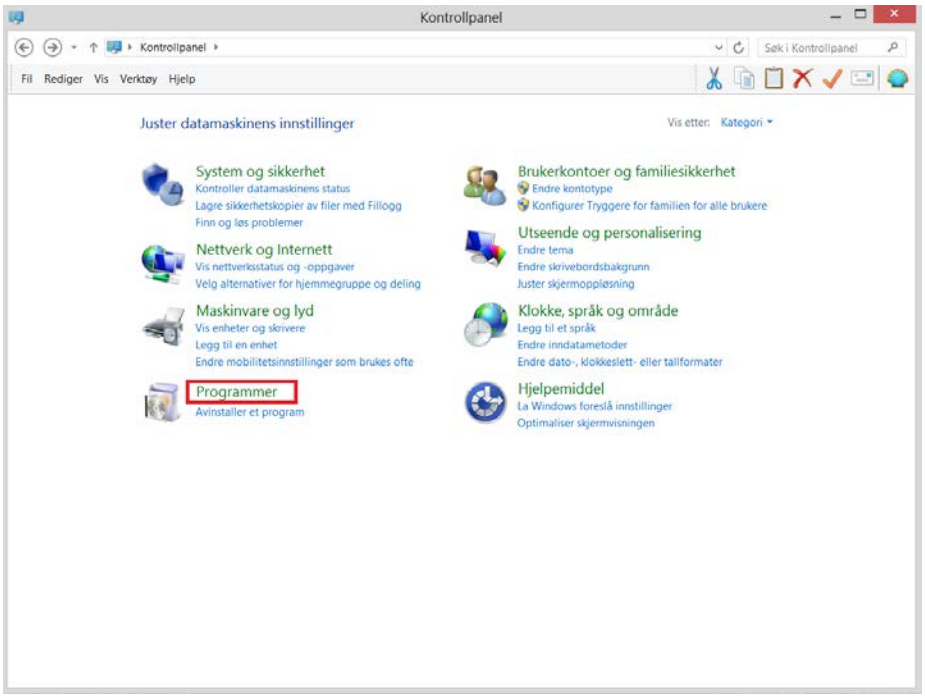
For å kunne kjøre ACT programet må man gjøre litt forarbeid. Dette dokumentet skal hjelpe deg med å installere de programmene du trenger og sette de instillingene som er nødvendig.

Det man må gjøre er å:

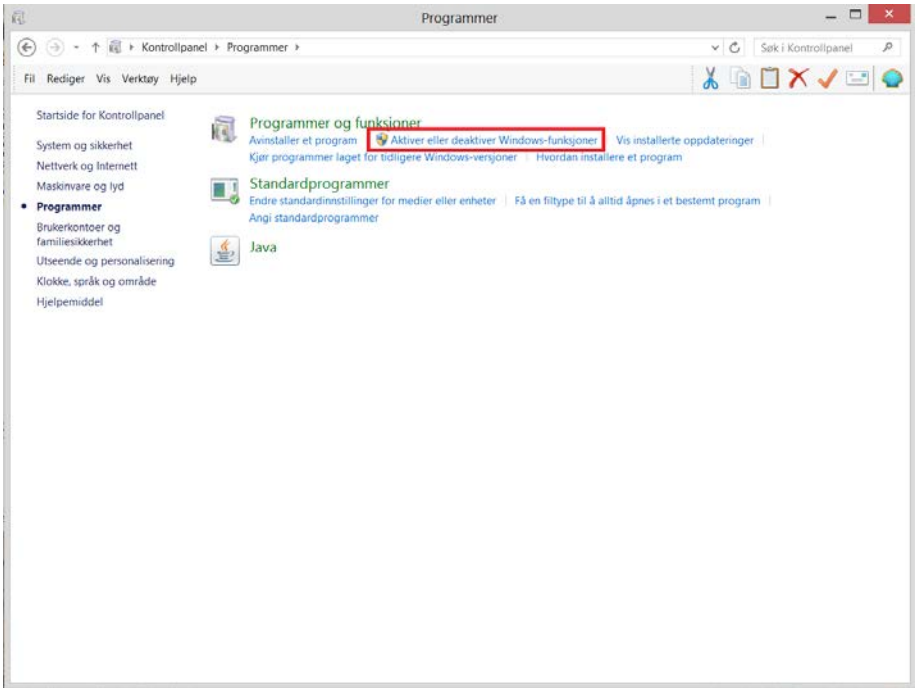
- Installere Microsoft SQL server og koble til vår database.
- Aktivere telnet og sette pc-en man kjører programmet fra til host på våpenstasjonen.
- Installere FileZilla.
- Legge til to regler i windows brannmur.

3 Activate Telnet and become the RWS host

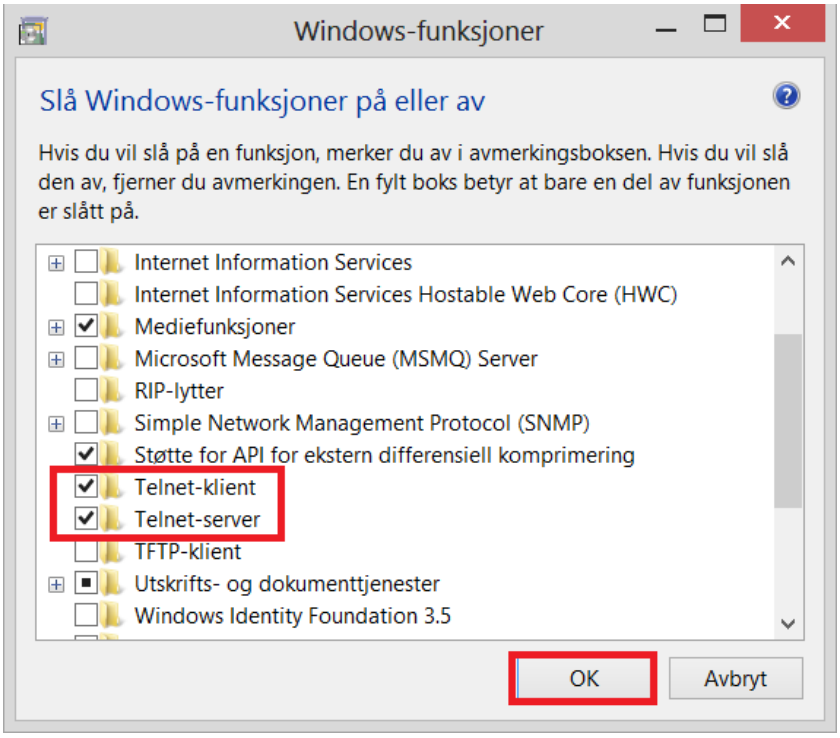
Gølg disse instruksjonene for å aktivere Telnet og bli RWS host.



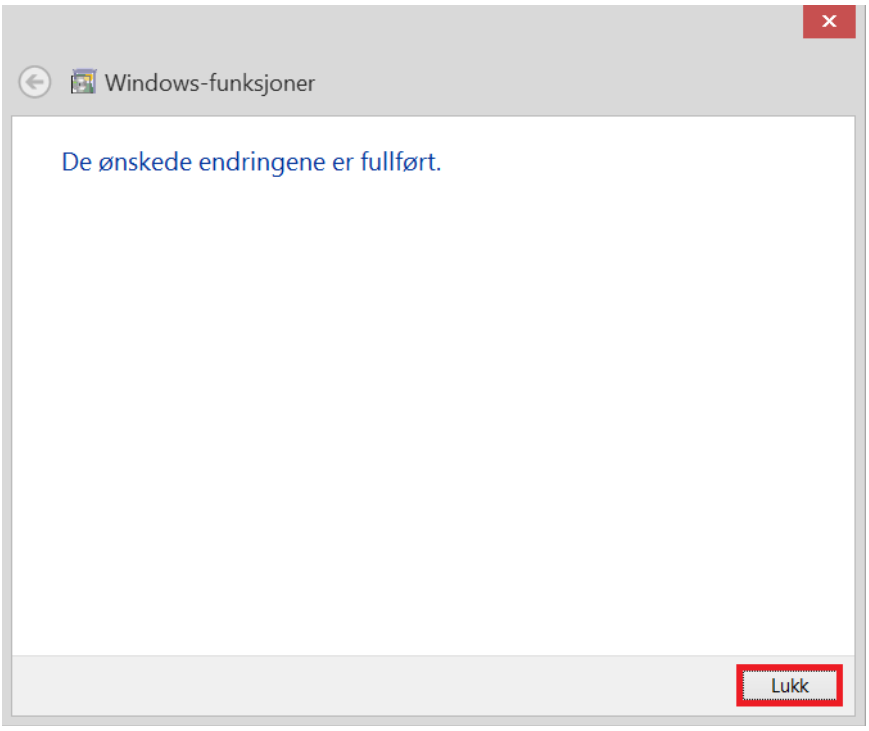
Gå til “Kontrollpanel”.
Så trykk på “Programmer”.



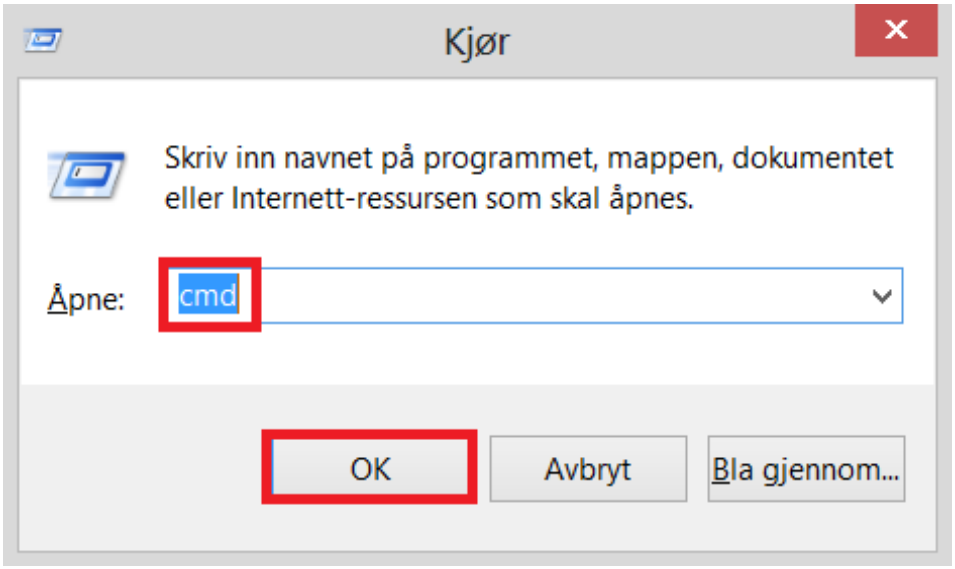
Trykk på “Aktiver eller deaktiver Windows-funksjoner”.



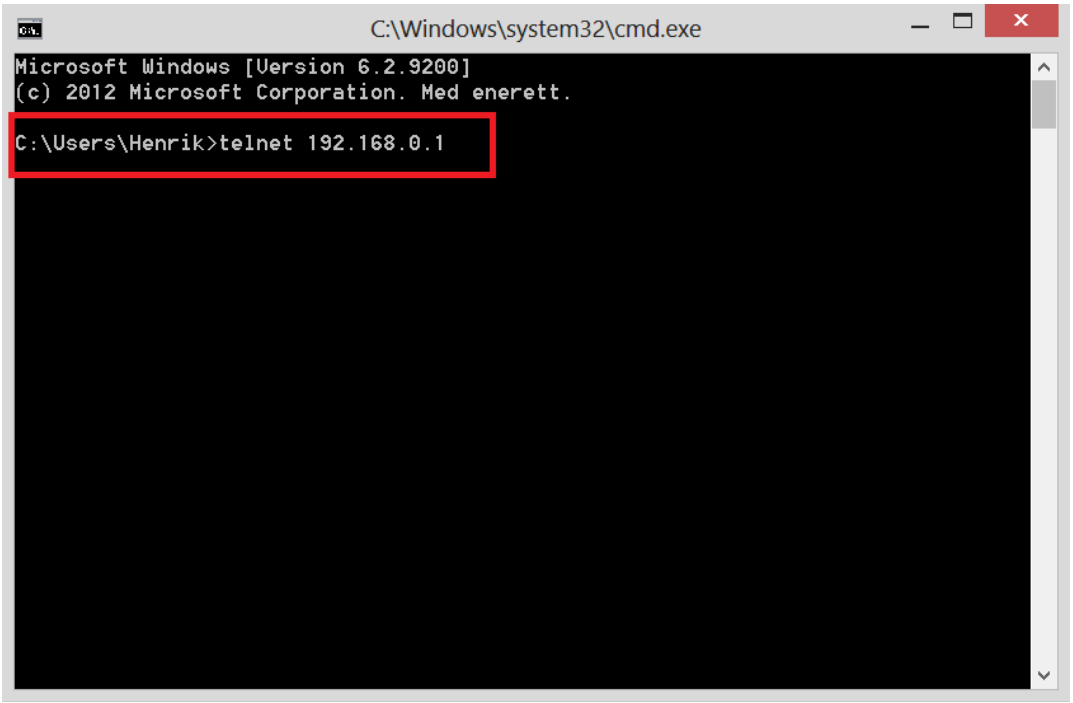
Aktiver “Telnet-klient” og “Telnet-server”.
Trykk “OK”.



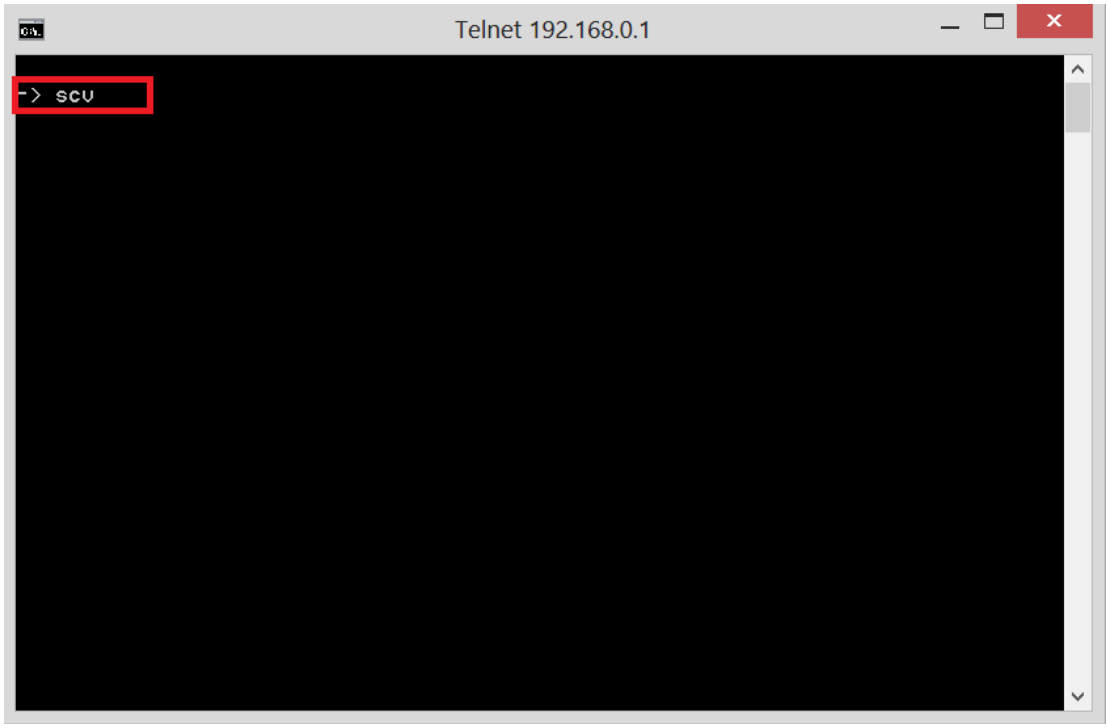
Trykk “Lukk”.



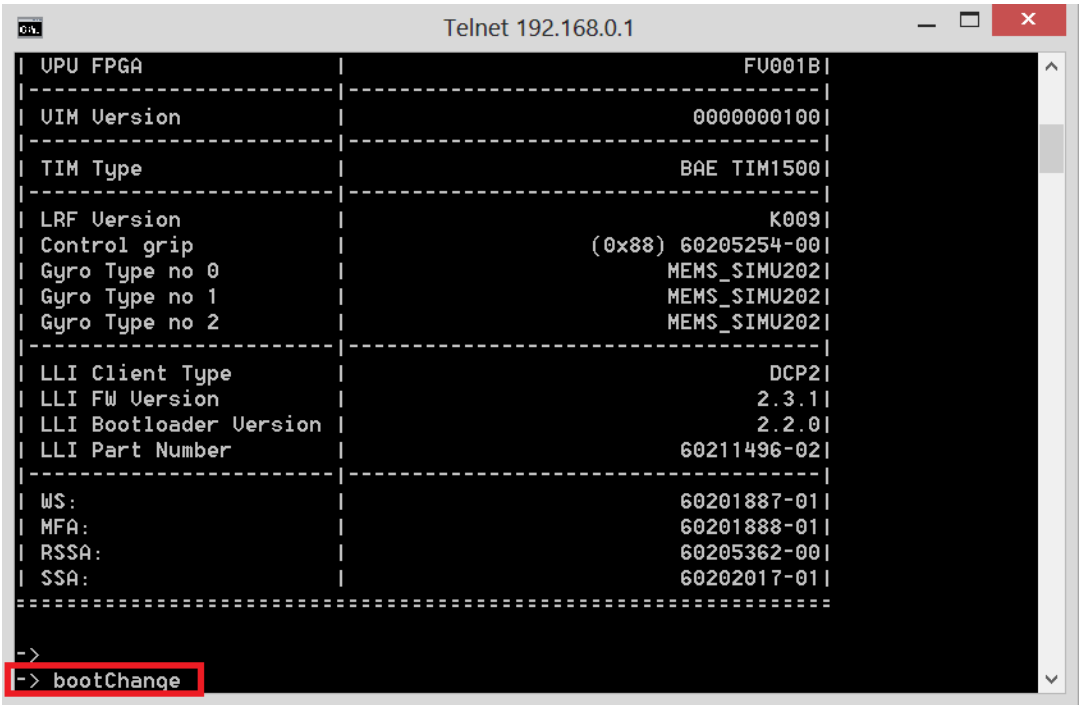
For å åpne dette vindu, trykk “⊞ + r” på tastaturet.
Skriv “cmd” og trykk “OK”.



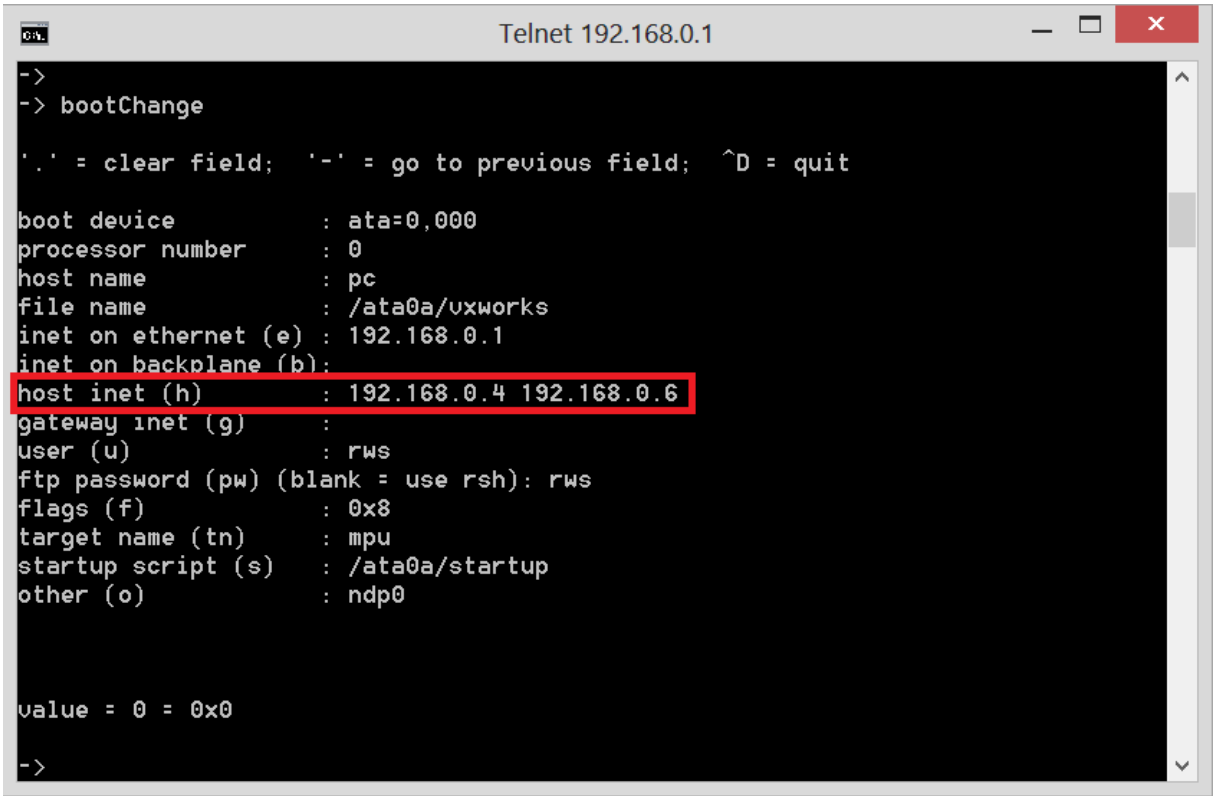
Skriv “telnet 192.168.0.1” (IP addressen til våpenstasjonen)
Trykk “enter” på tastaturet.



Skriv “scv” og trykk “enter” på tastaturet.



Skriv “bootChange” og trykk “enter ”på tastaturet.



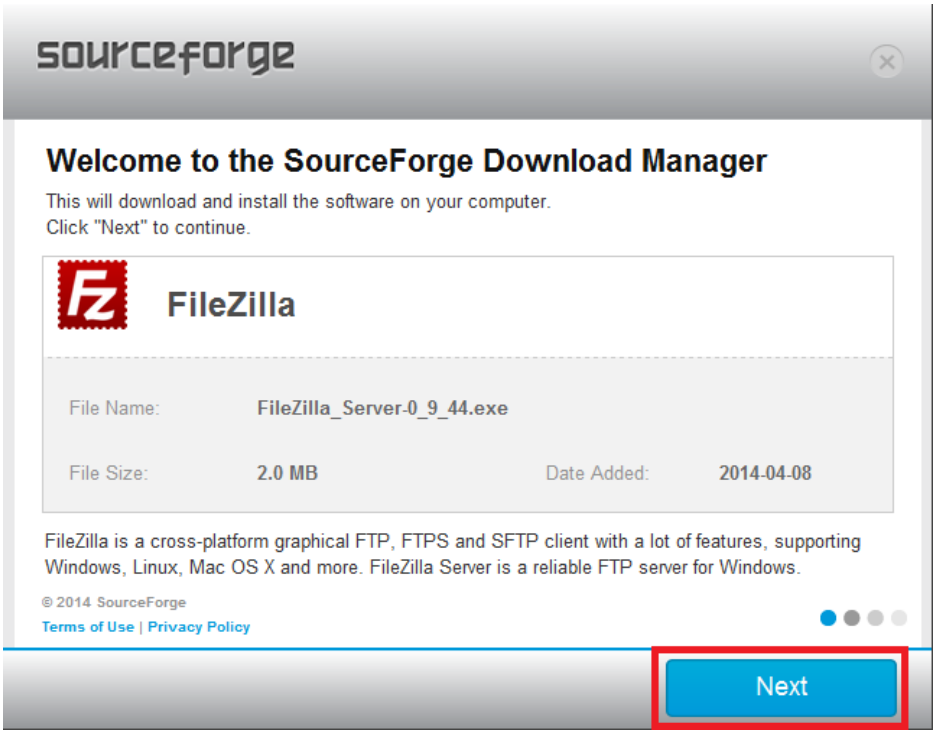
Trykk “enter ” flere ganger helt til du kommer til “host inet”.
Trykk “space” og din IP addresse.
Trykk “enter” helt til du ser “->”.
Og du er ferdig!

4 FileZilla

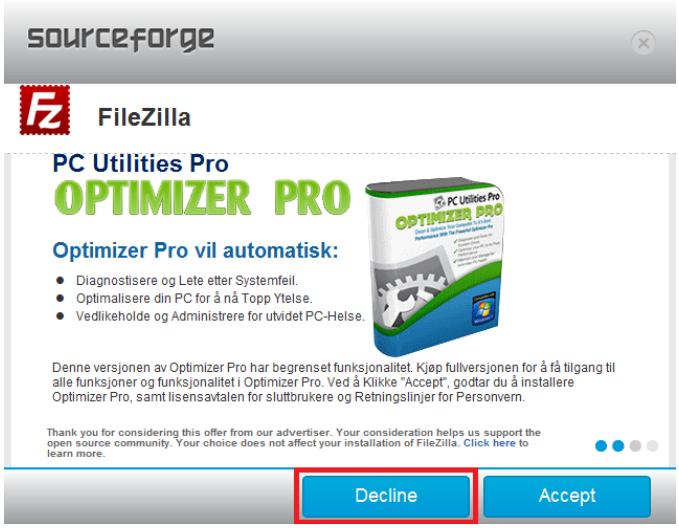
Følg disse instruksjonene for å installere FileZilla.

Gå til denne linken:
https://filezilla-project.org/download.php?type=server

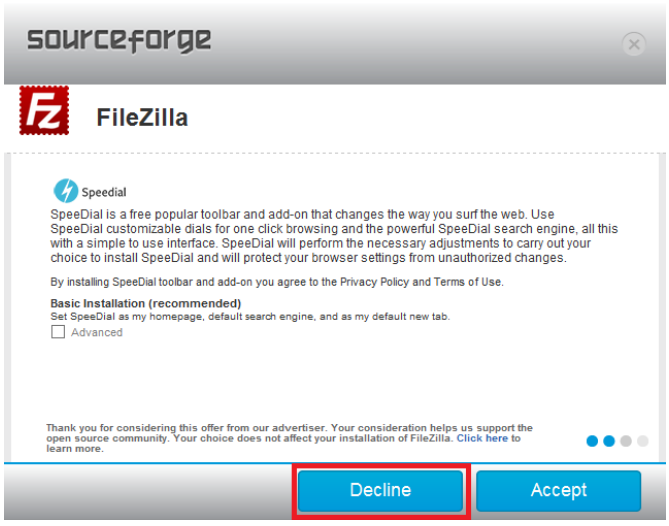
Trykk "Download Now".
Kjør fila "FileZilla_Server-0_9_44.exe" (or the latest verson) når den er ferdig nedlastet.



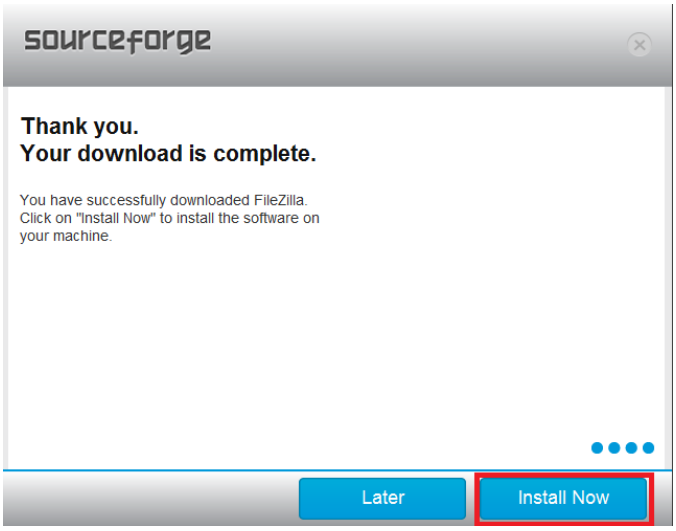
Trykk "Next".



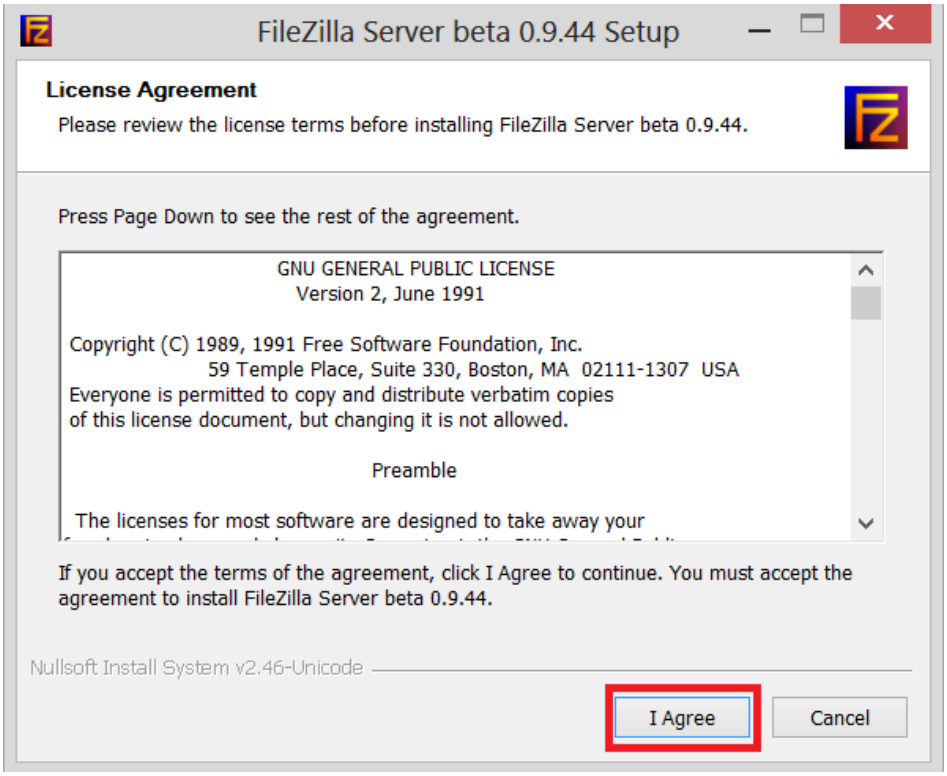
Trykk “Decline”.



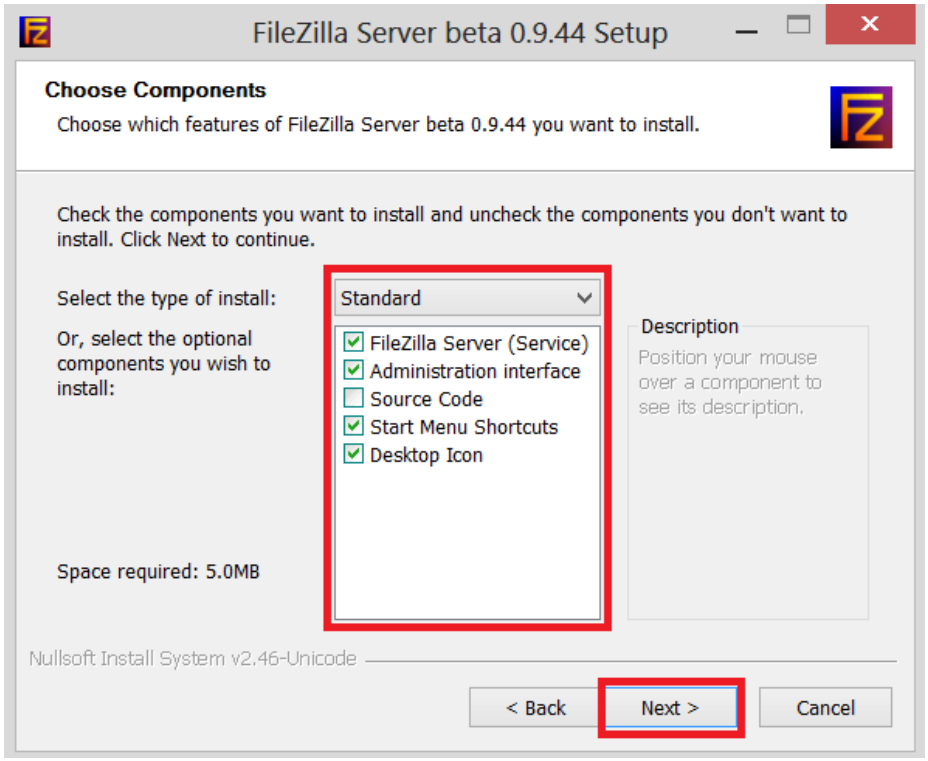
Trykk “Decline”.



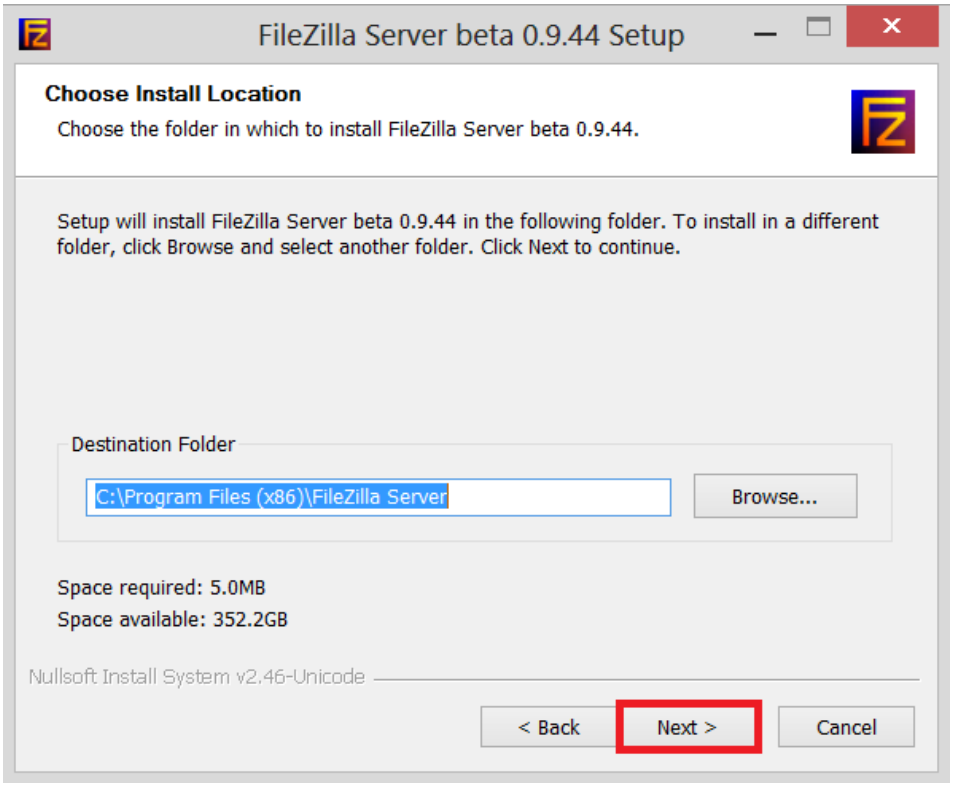
Trykk “Install Now”.



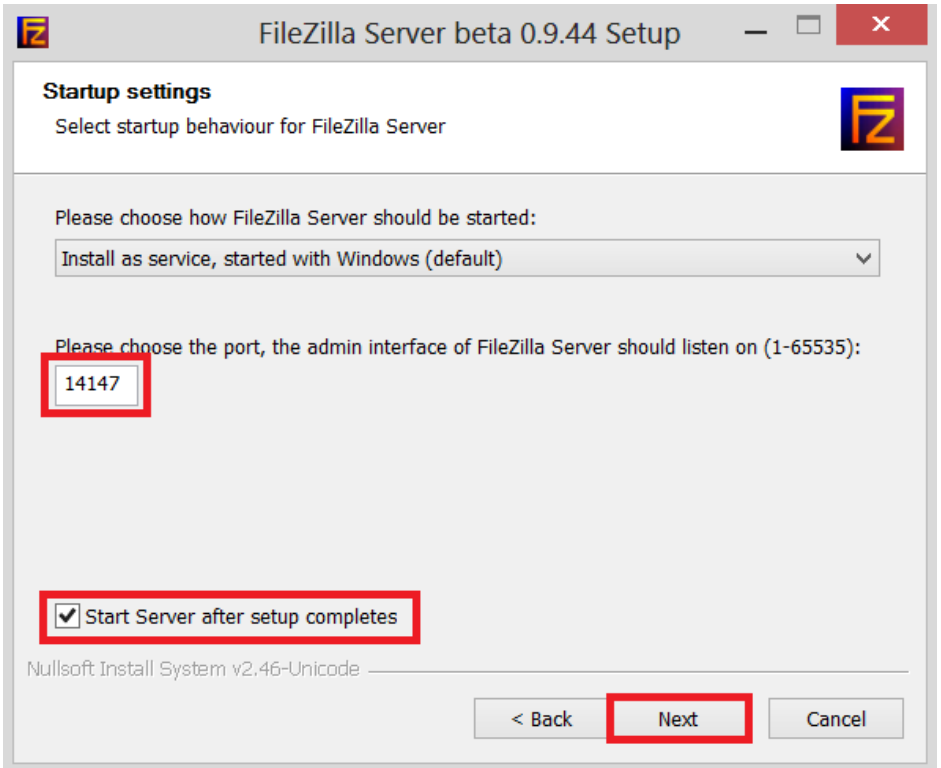
Trykk “I Agree”.



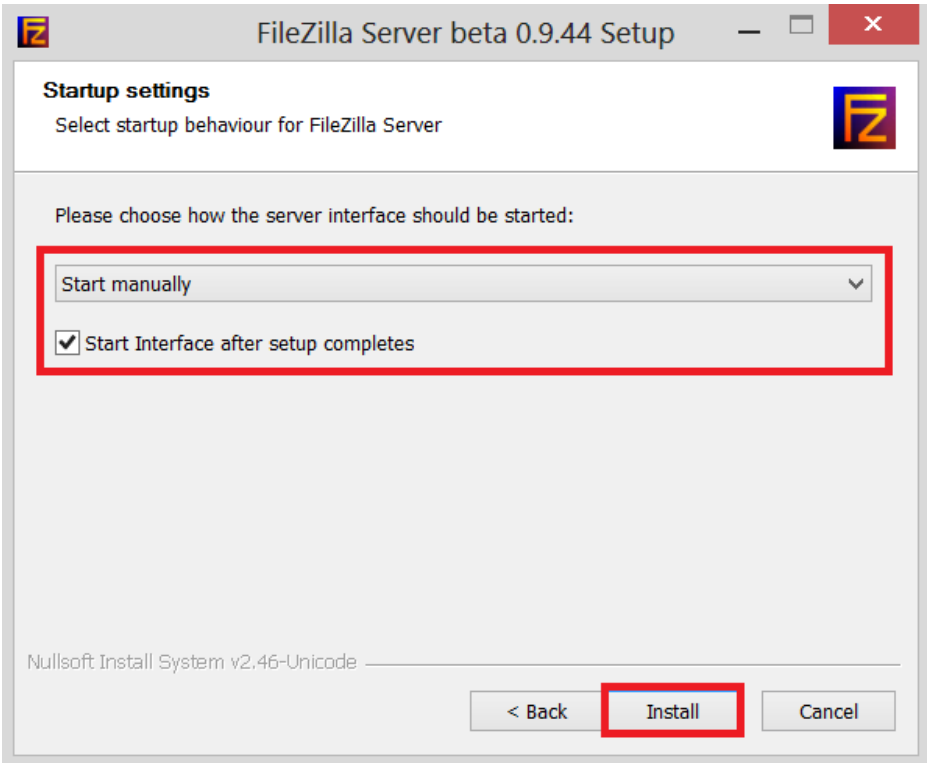
Sjekk at checkbox-ene er likt som ovenfor.
Deretter trykk “Next”.



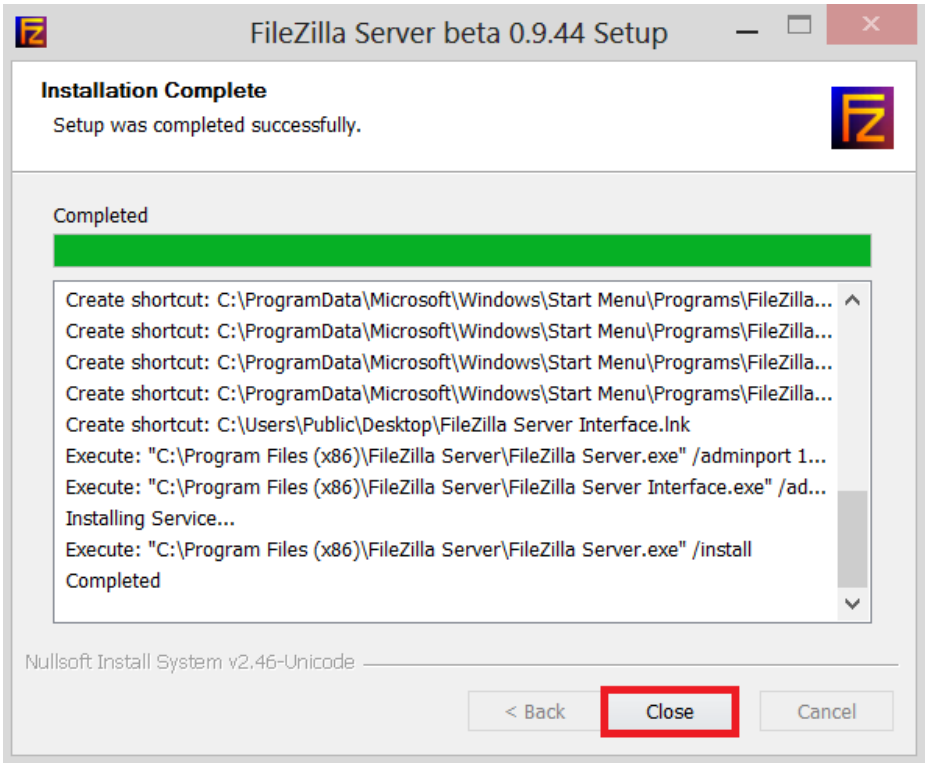
Trykk “Next”.



Port nr: 14147.
Huk av checkbox-en.
Deretter trykk “Next”.

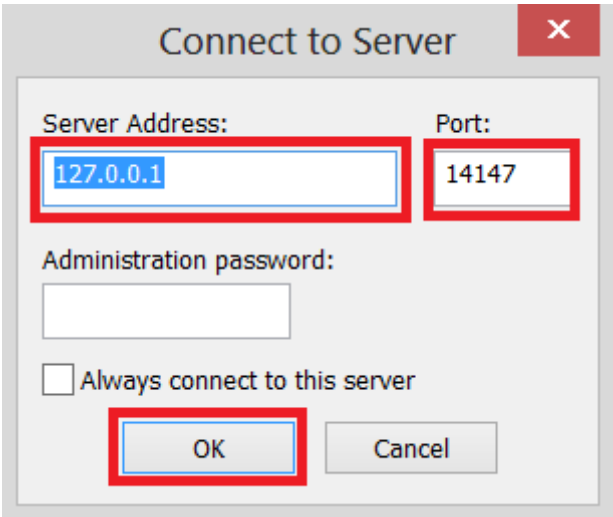


Velg “Start manually”.
Huk av checkbox-en.
Deretter trykk “Install”.

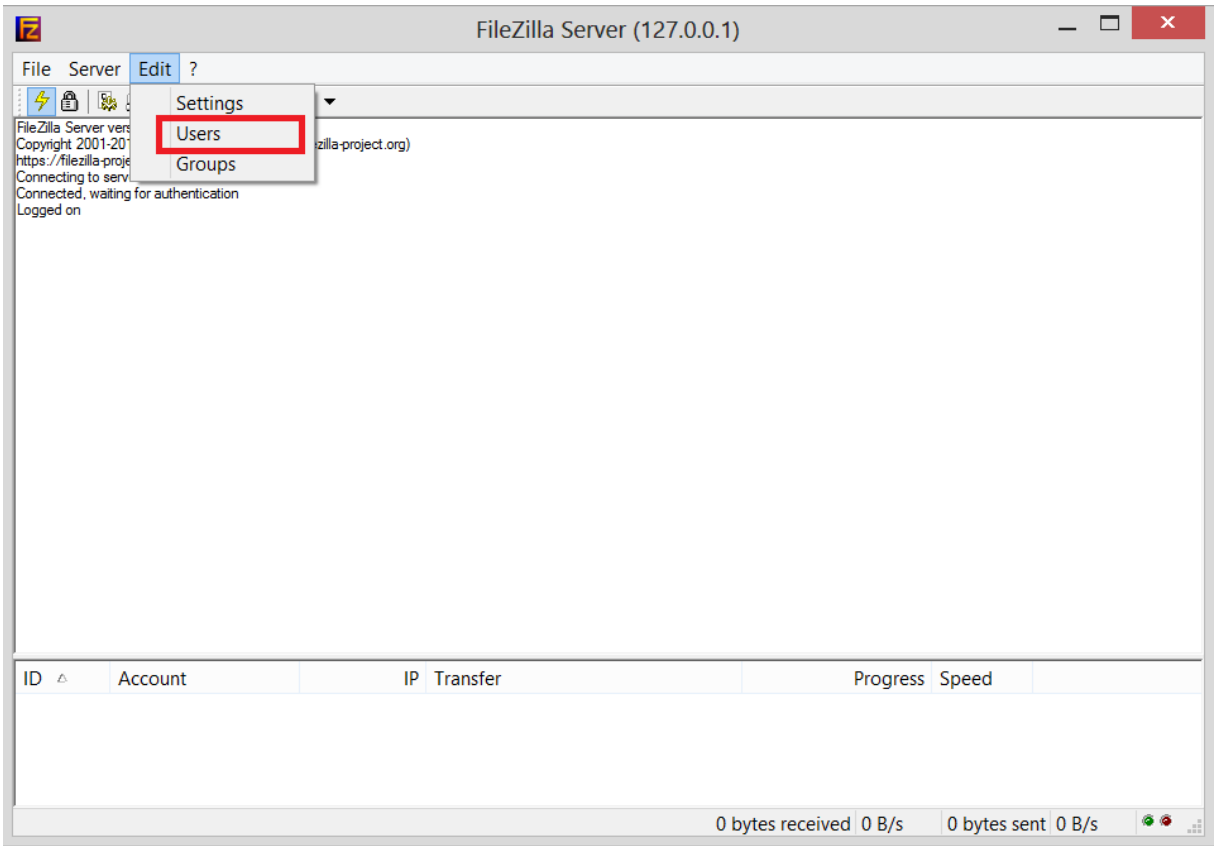


Trykk “Close”.

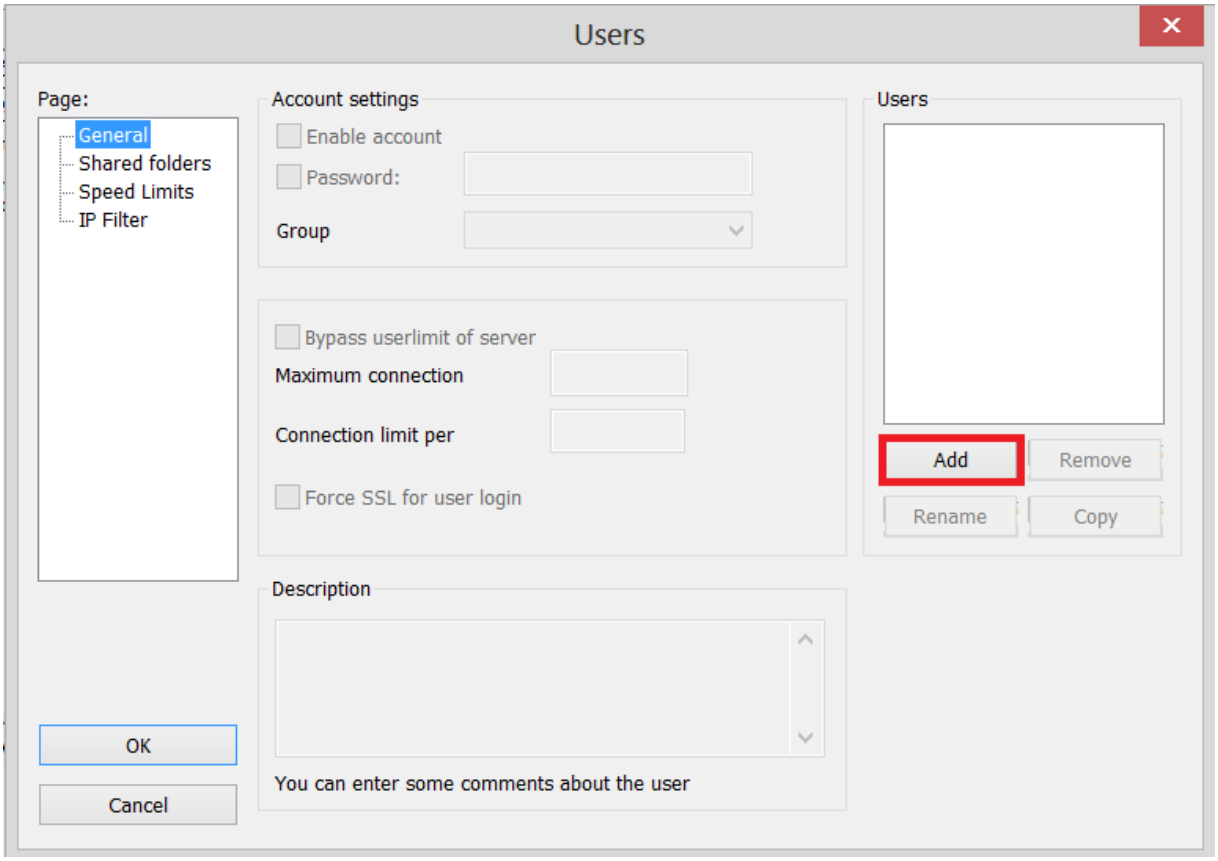
Deretter åpner du FileZilla.



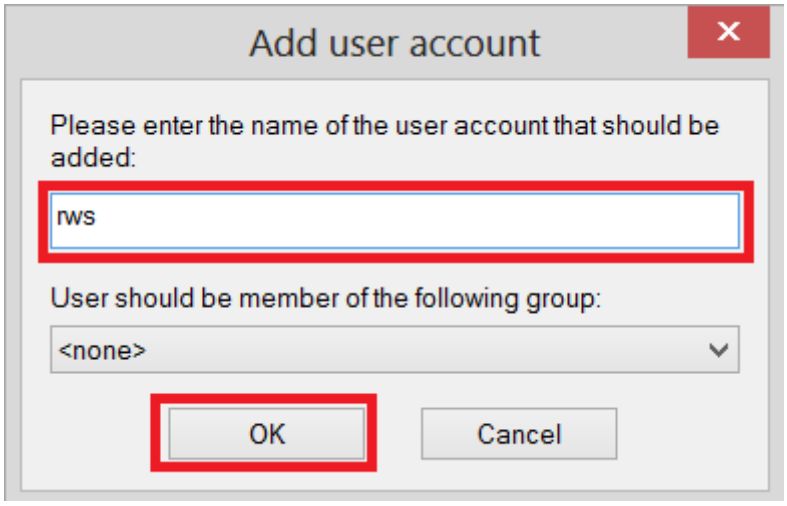
Server Address: 127.0.0.1
Port: 14147
Så trykker du “OK”.



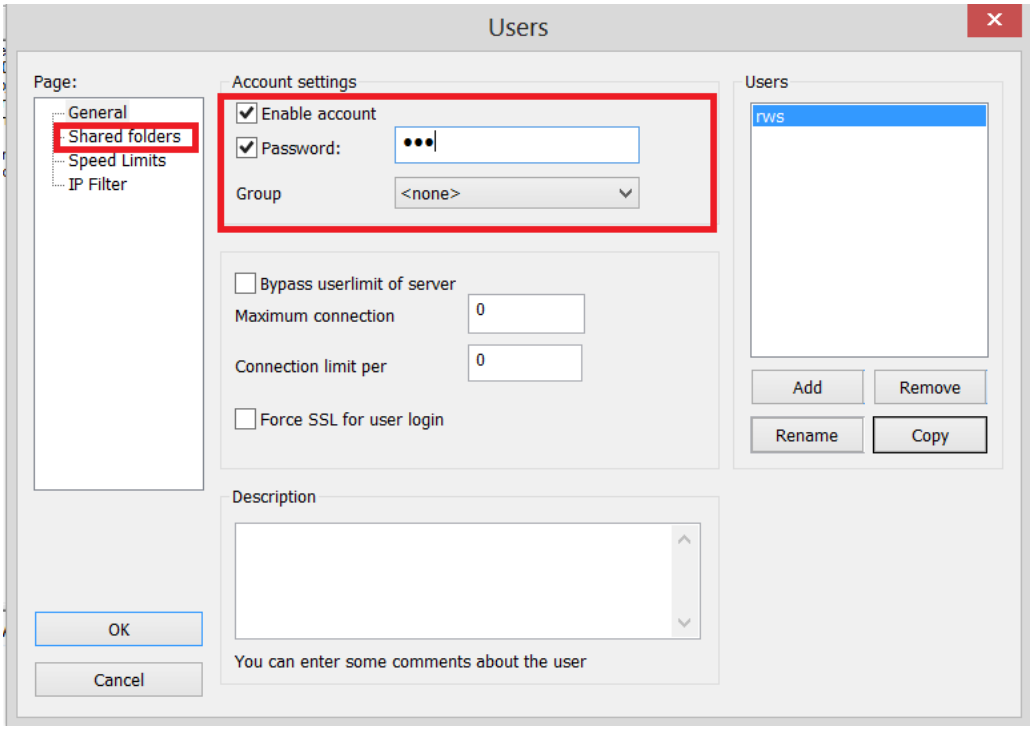
Trykk “Edit” deretter “Users”.



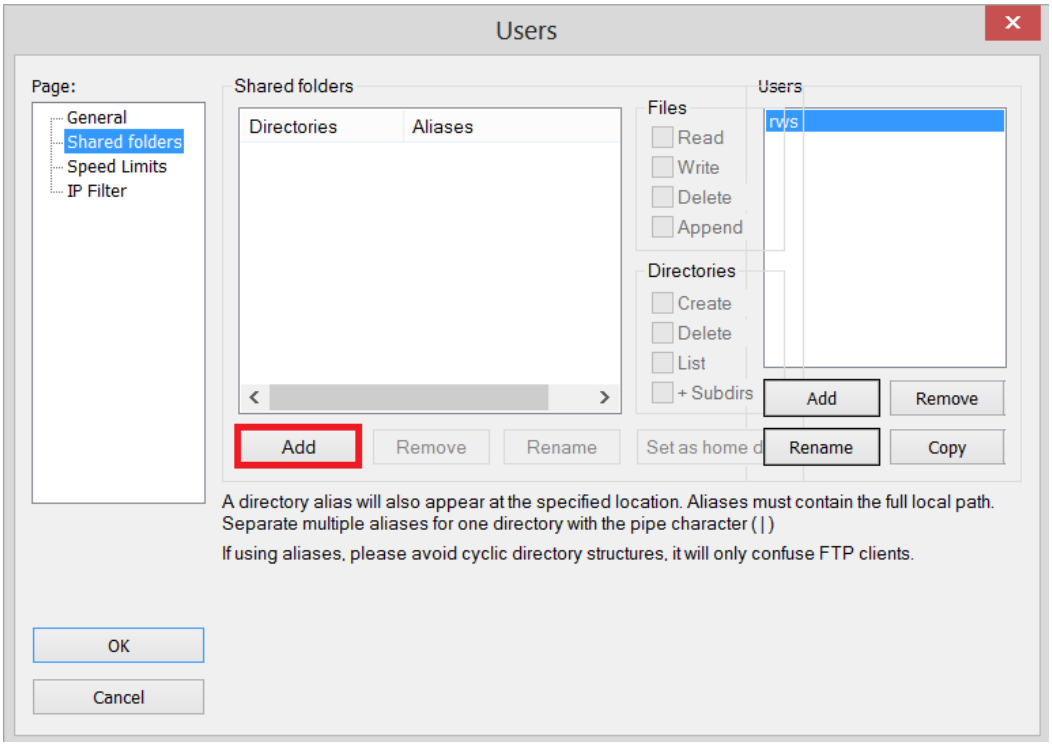
Trykk “Add”.



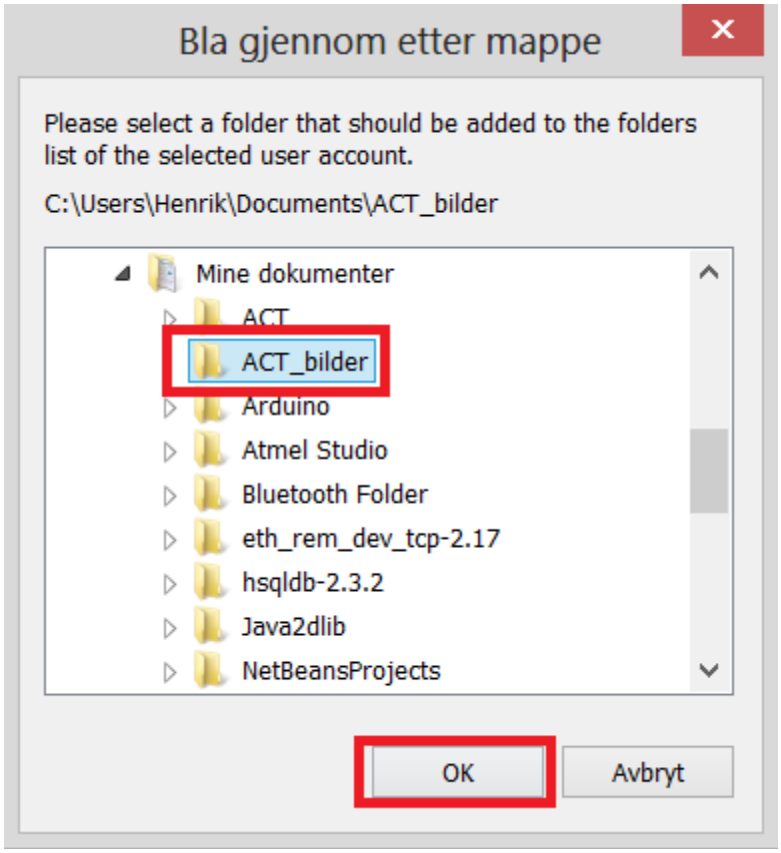
User: rws
Trykk “OK”.



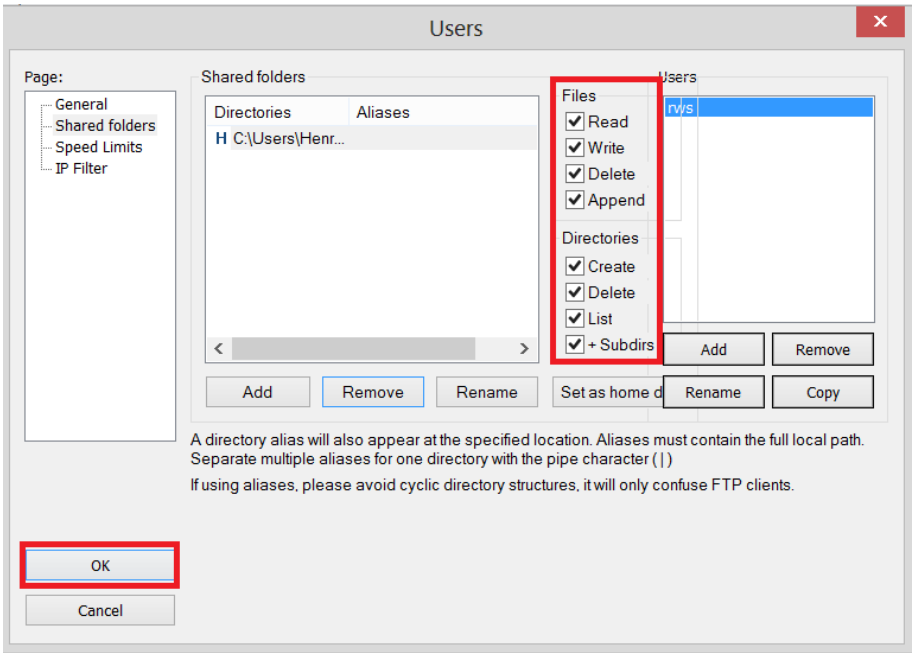
Sjekk at begge checkbox-ene er huket av.
Passord: “rws”.
Deretter trykk “Shared folders”.



Trykk “Add”.

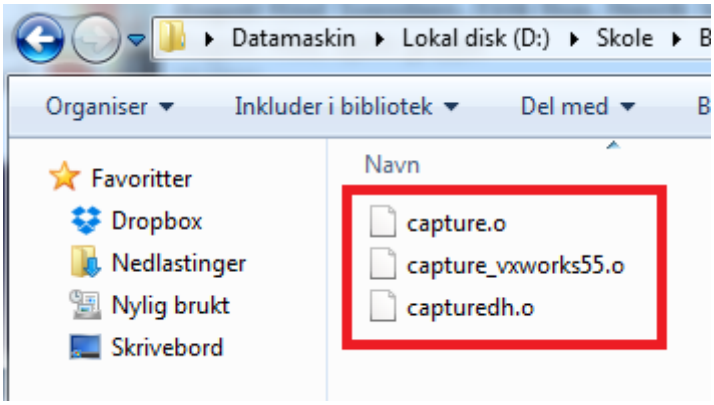


Lag en ny mappe inne i "Mine Dokumenter"-mappen og kall den "ACT_bilder" og velg denne.
Deretter trykker du "OK".



Huk av alle checkbox-ene.
Deretter trykker du "OK".

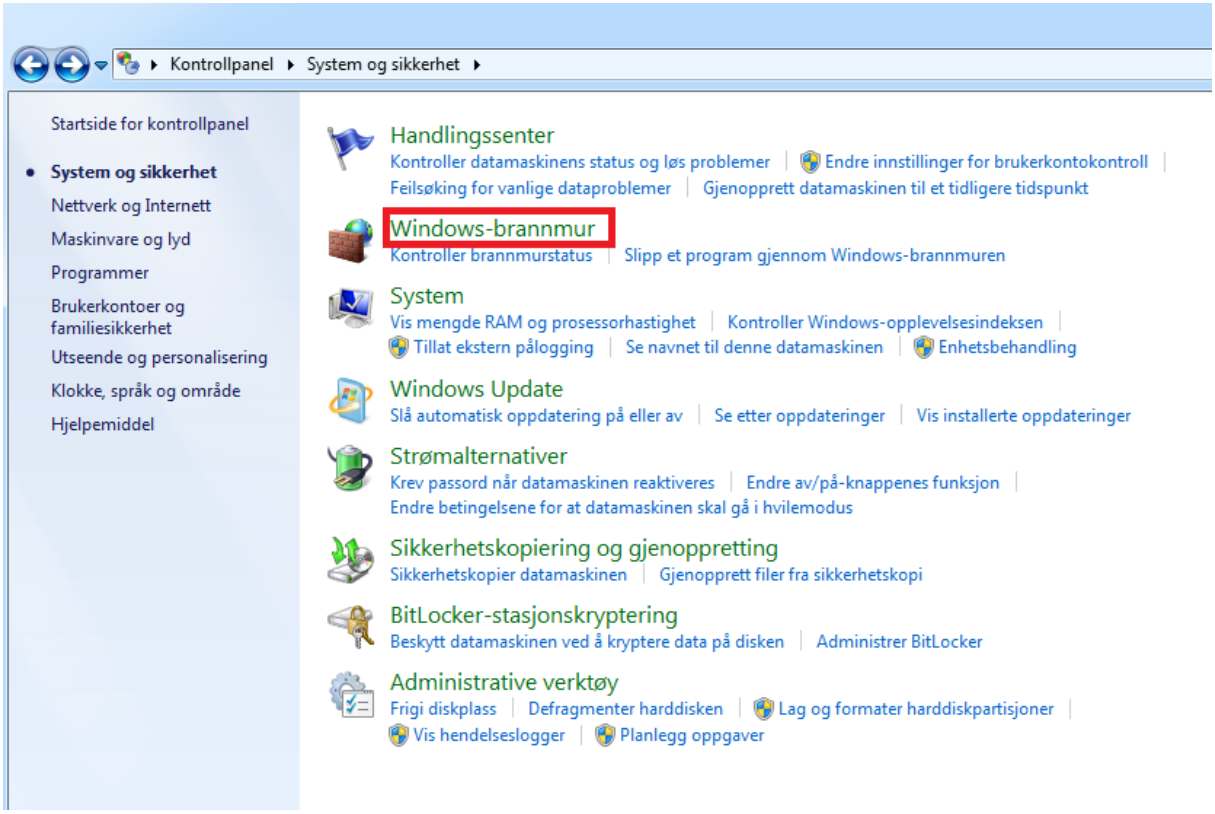
Installasjonen er ferdig!



Etter installasjonen er ferdig må du kopiere disse filene inn i “ACT_bilder ” mappen du opprettet tidligere.
På cd’en finner du disse to filene. (De kan også lastes ned fra hjemmesiden vår etter den 27.05.14)

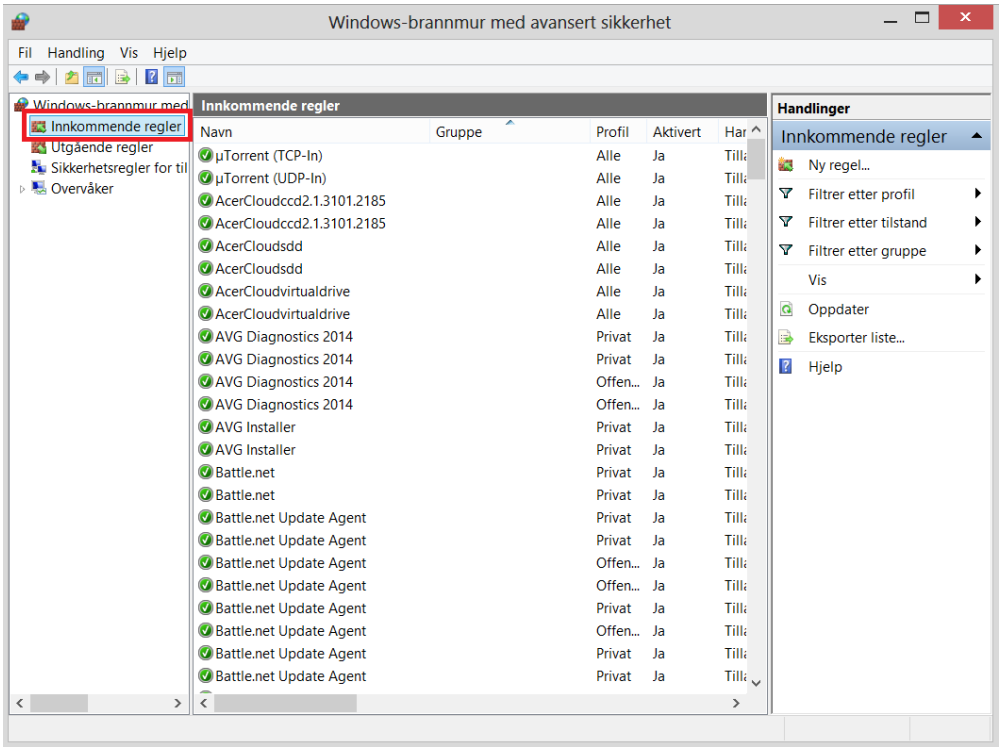
4.1 Brannmur instillinger

Følg disse instruksjonene for å sette de riktige brannmur innstillingene med FileZilla.

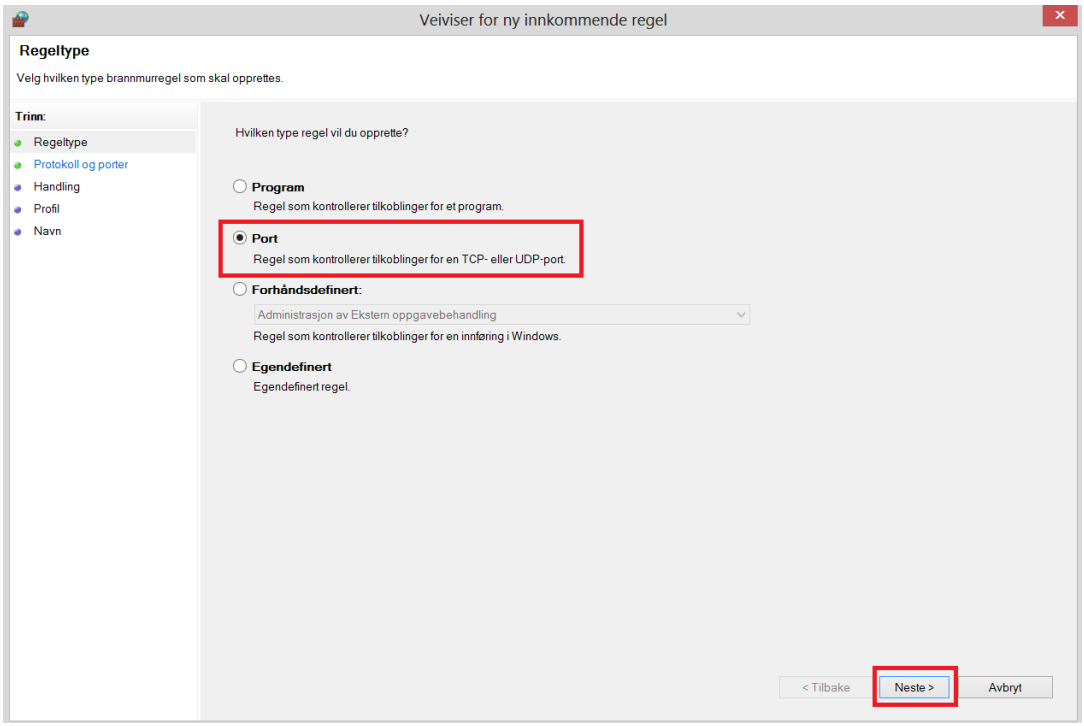


Gå til “Kontrollpanel”.
Så til “System og sikkerhet”.
Så “Windows-brannmur”.

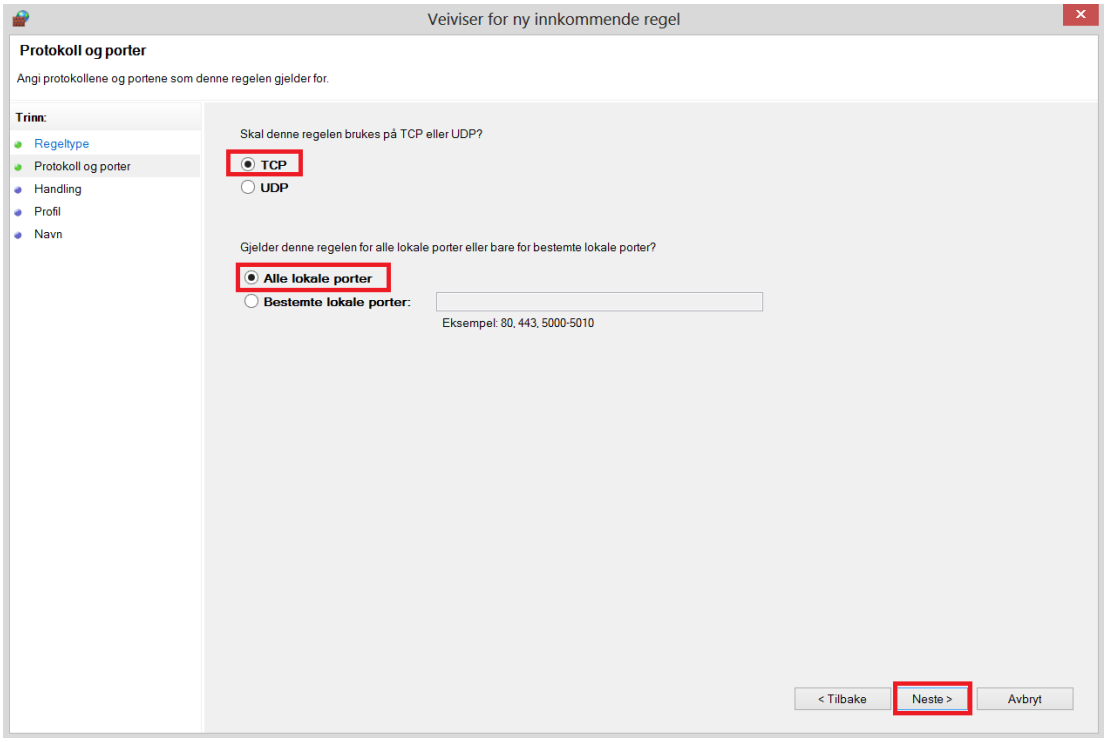
Deretter trykk “Avanserte innstillinger”.



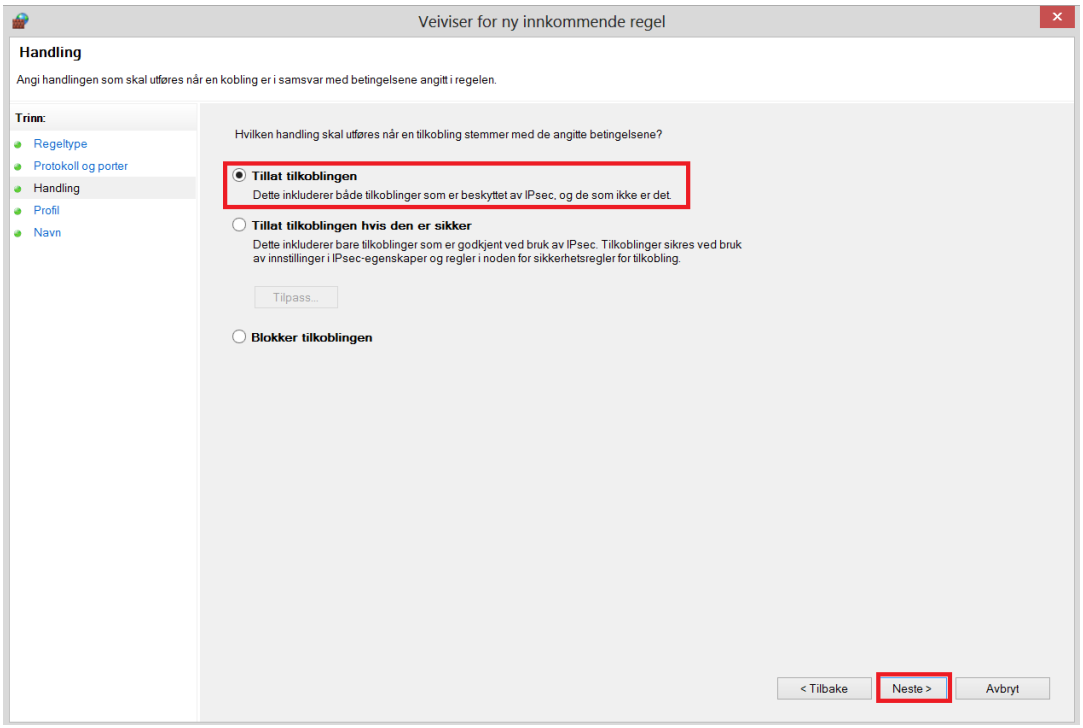
Trykk “Innkommende regler”.



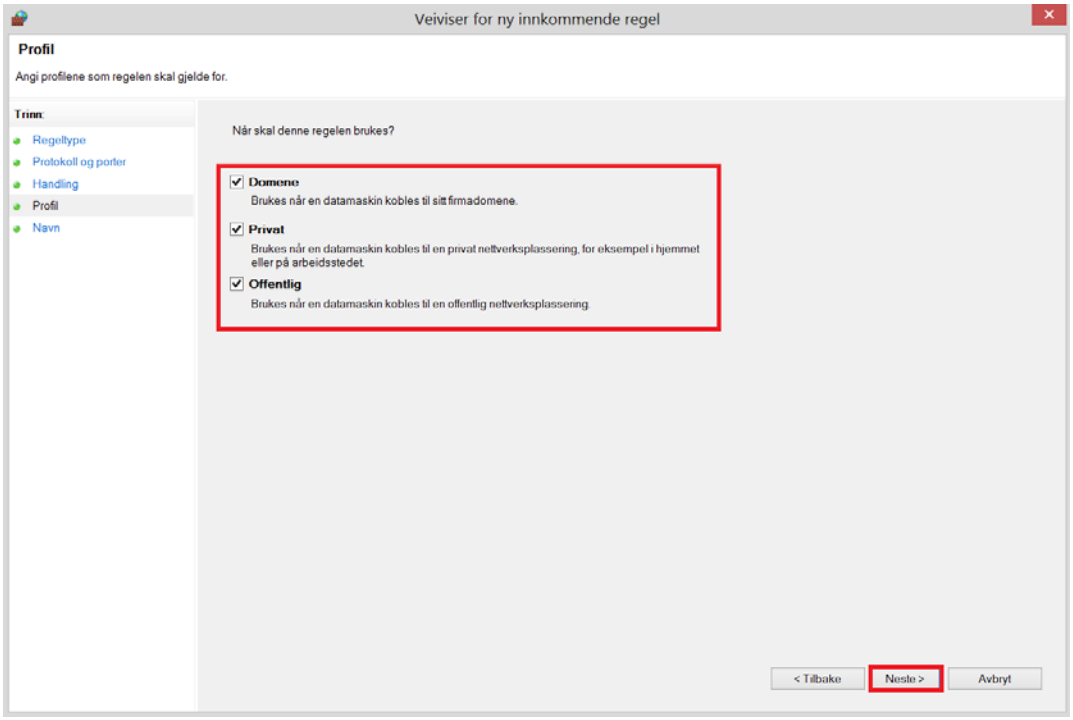
Velg “Port”.
Så trykk “Next”.



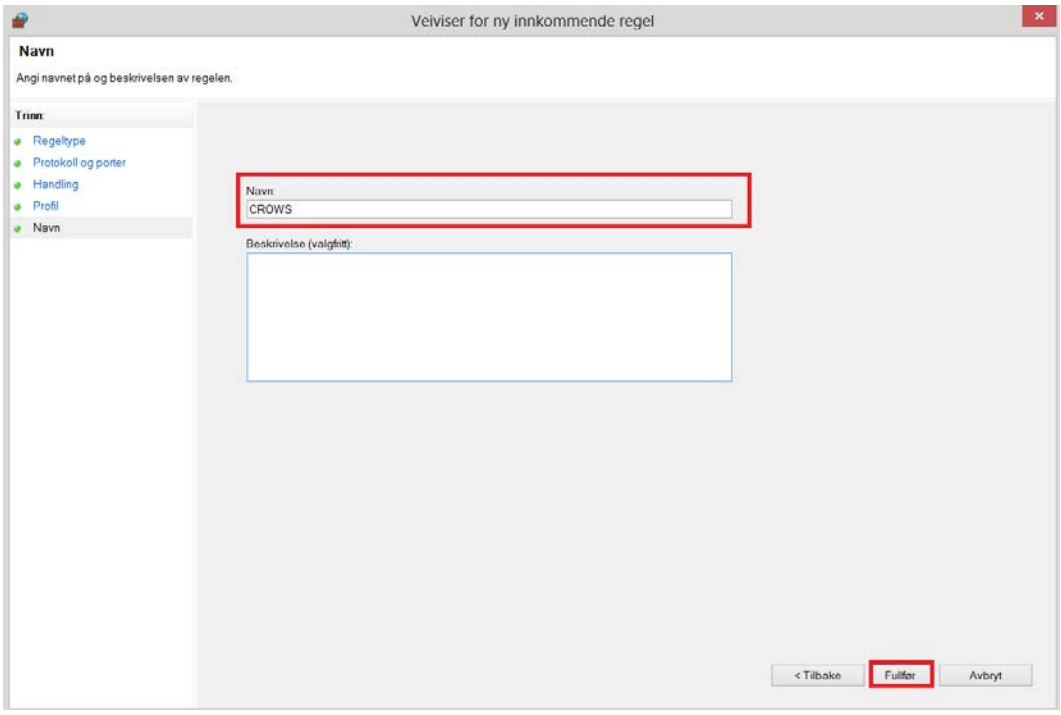
Velg “TCP”,
og “Alle locale porter”.
Så trykk “Next”.



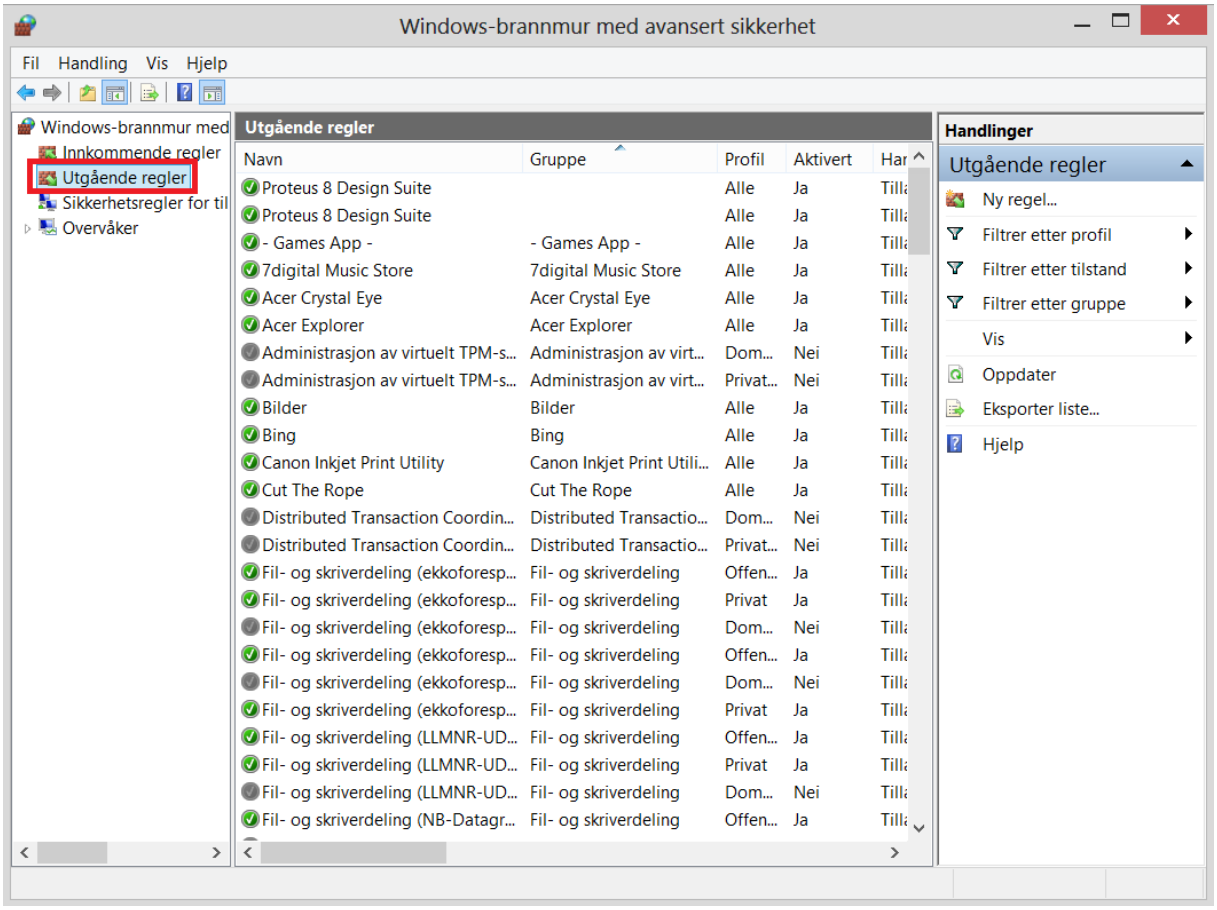
Velg “Tillat tilkoblingen”.
Trykk “Next”.



Sjekk at alle checkbox-ene er huket av.
Deretter trykk “Next”.



Navn: “CROWS”.
Trykk “Fullfør”.



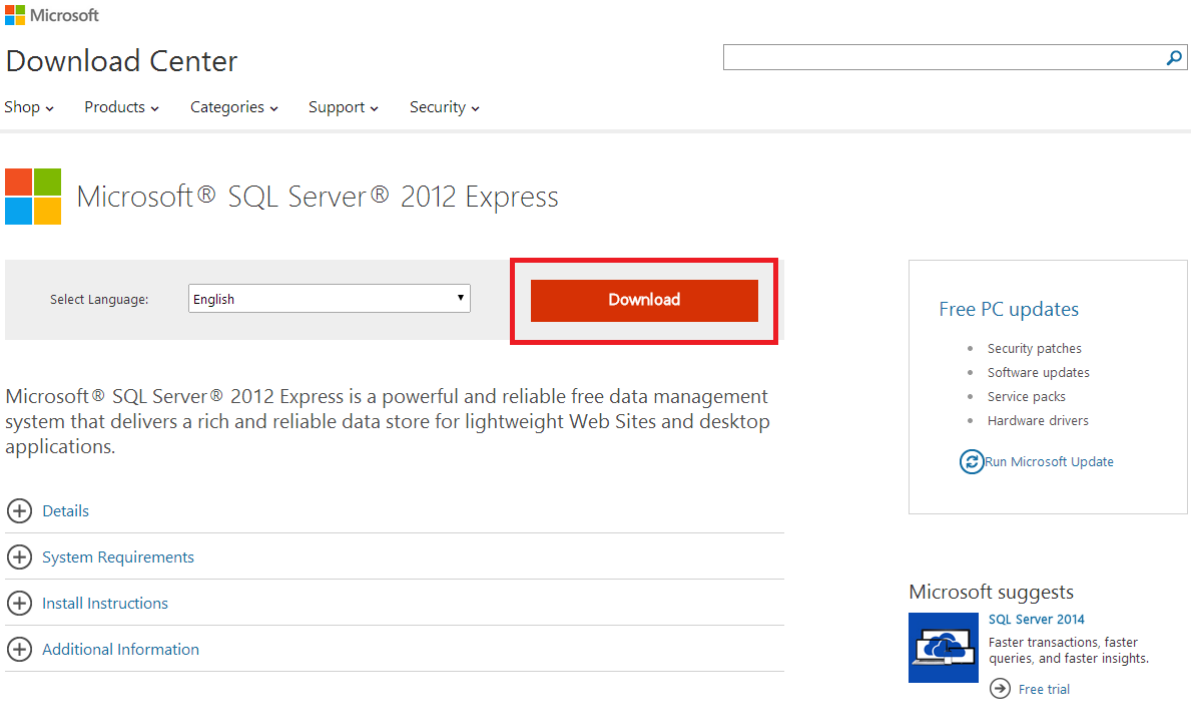
Trykk “Utgående regler” og lag den same regelen som du lagde for “Innkommende regler”.

Da har du reglene du trenger.

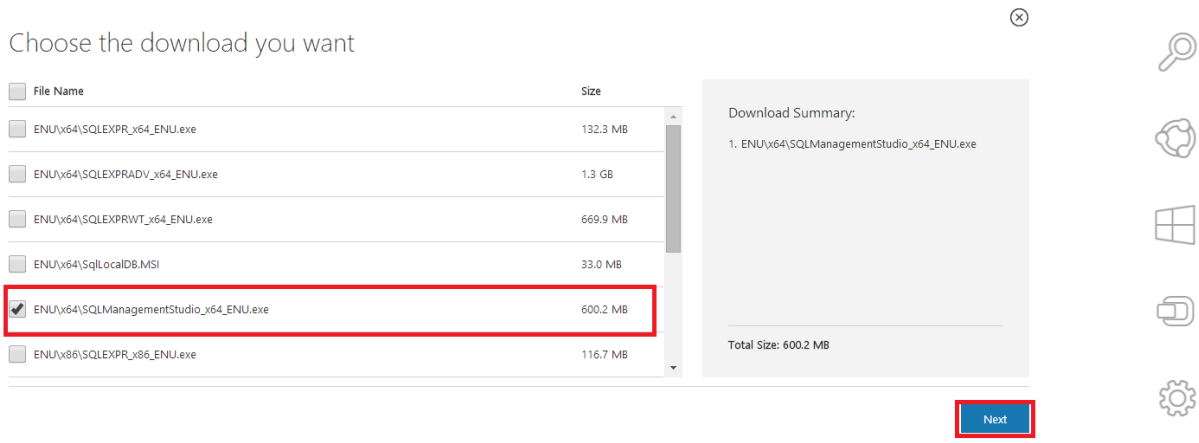
5 Microsoft SQL server

Følg disse instruksjonene for å installere Microsoft SQL server og legge inn databasen.

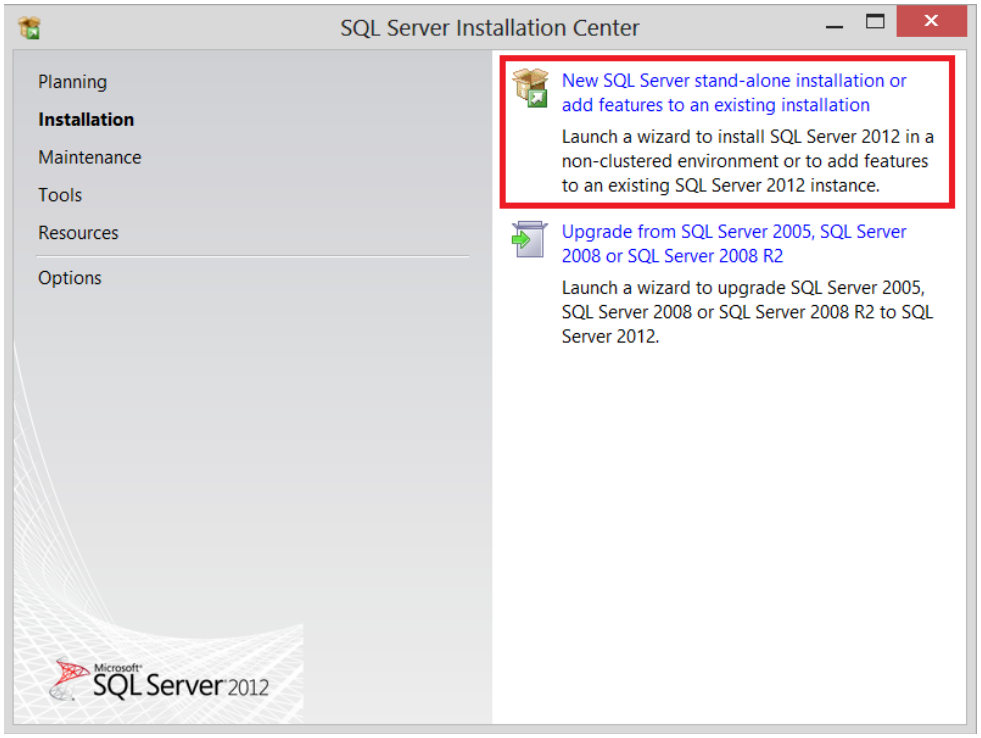
Gå til denne linken:
http://www.microsoft.com/en-us/download/details.aspx?id=29062



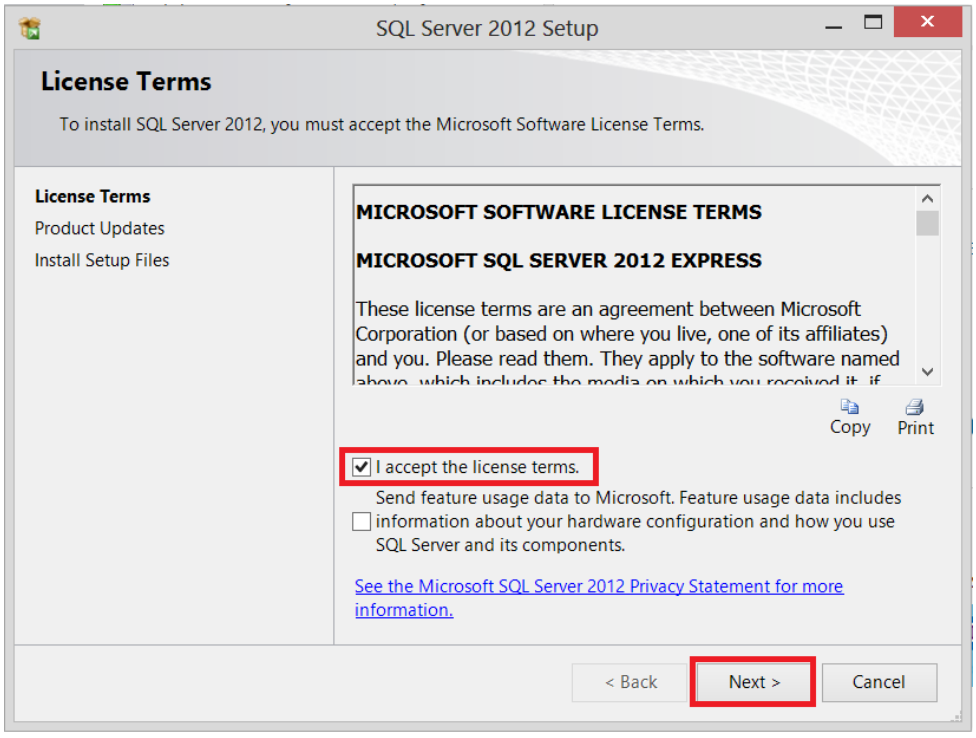
Trykk “Download”.



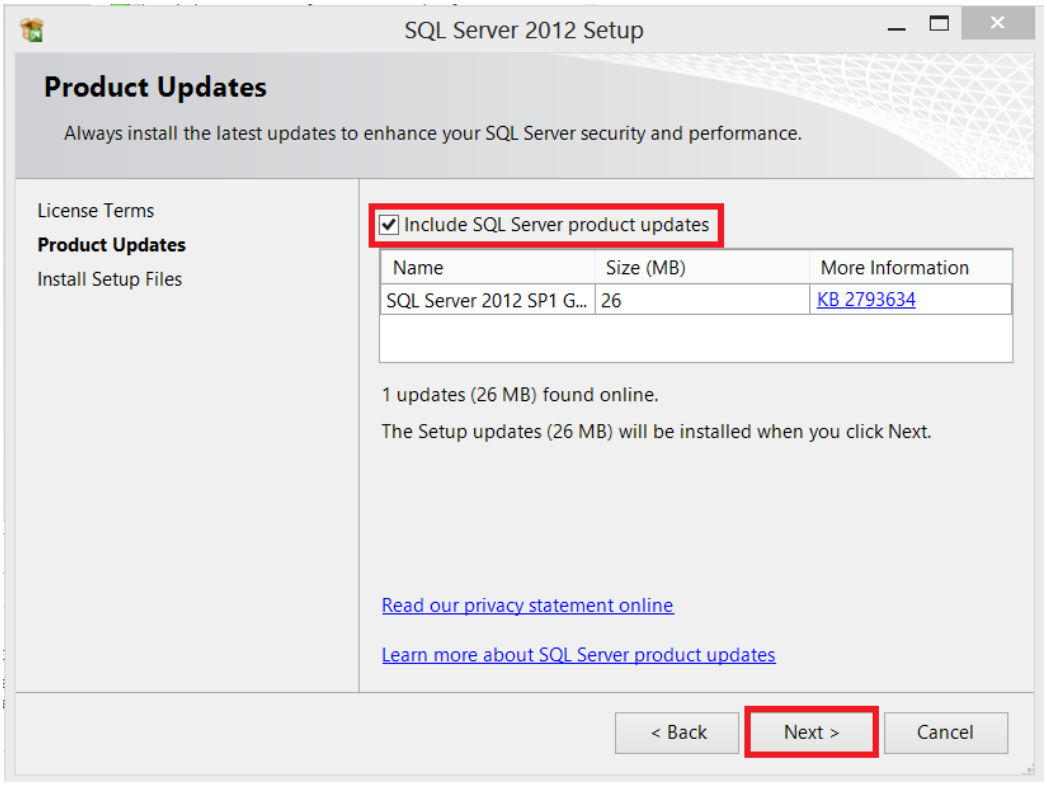
Velg “ENU\x64\SQLManagementStudio_x64_ENU.exe”
Trykk “Next”.
Kjør fila når den er ferdig nedlastet.



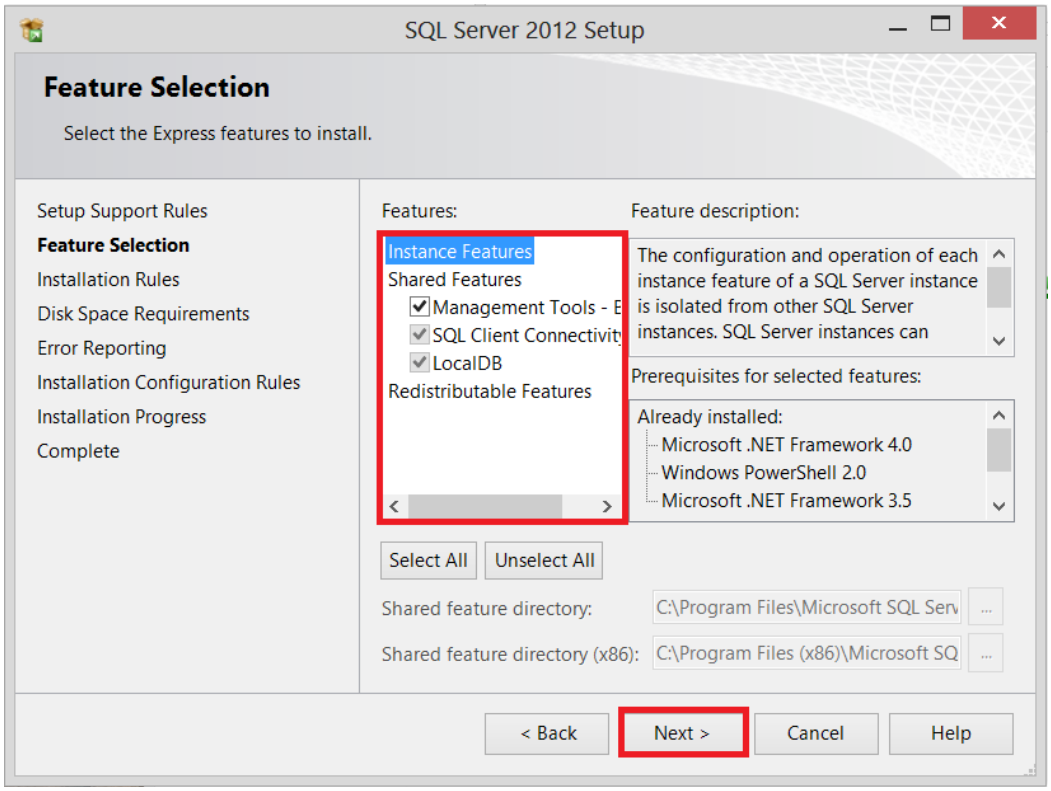
Trykk “New SQL Server stand-alone installation or add features to an existing installation”



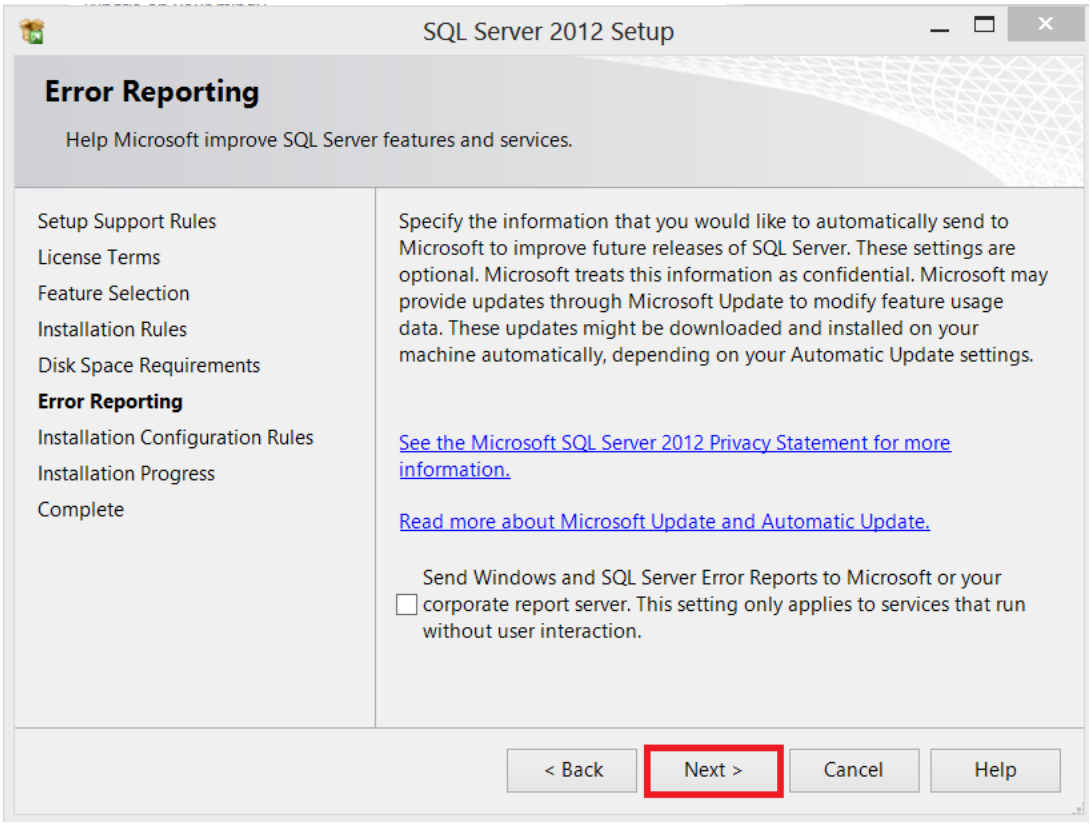
Huk av “I accept the license terms.”
Deretter trykk “Next”.



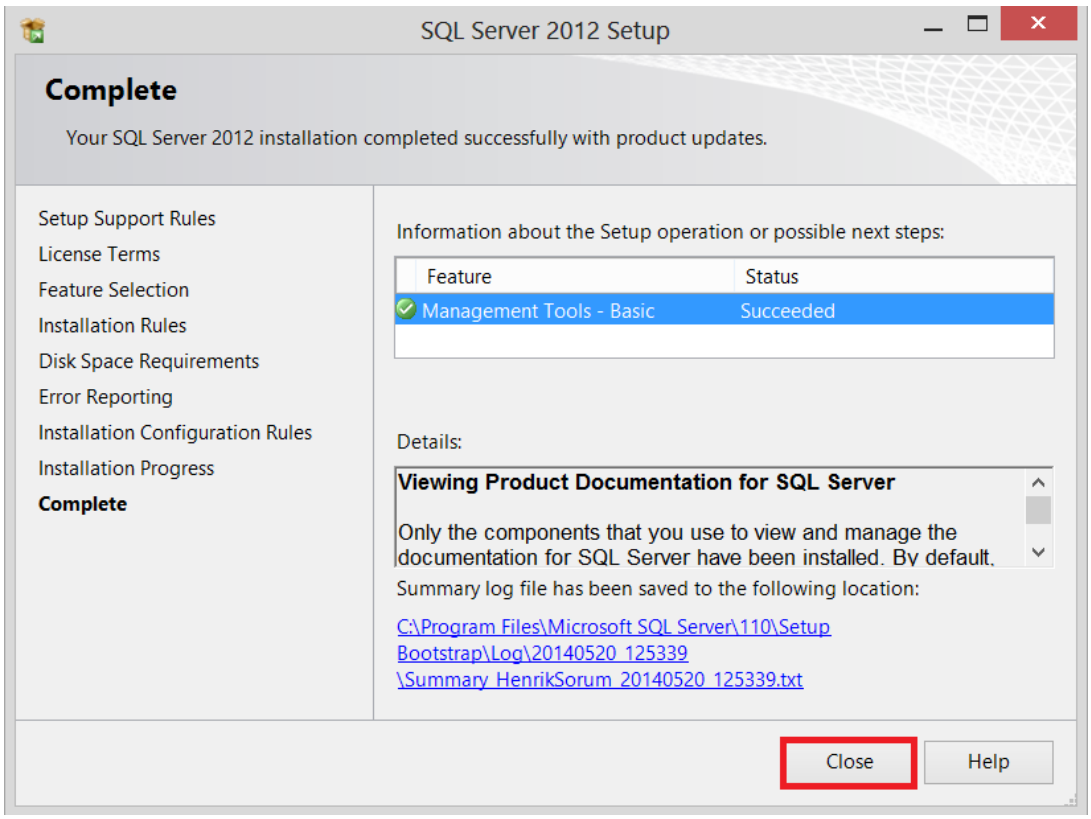
Huk av "Include the SQL Server product updates", så trykk "Next".



Features: Huk av alle checkbox-ene.
Deretter trykk "Next".

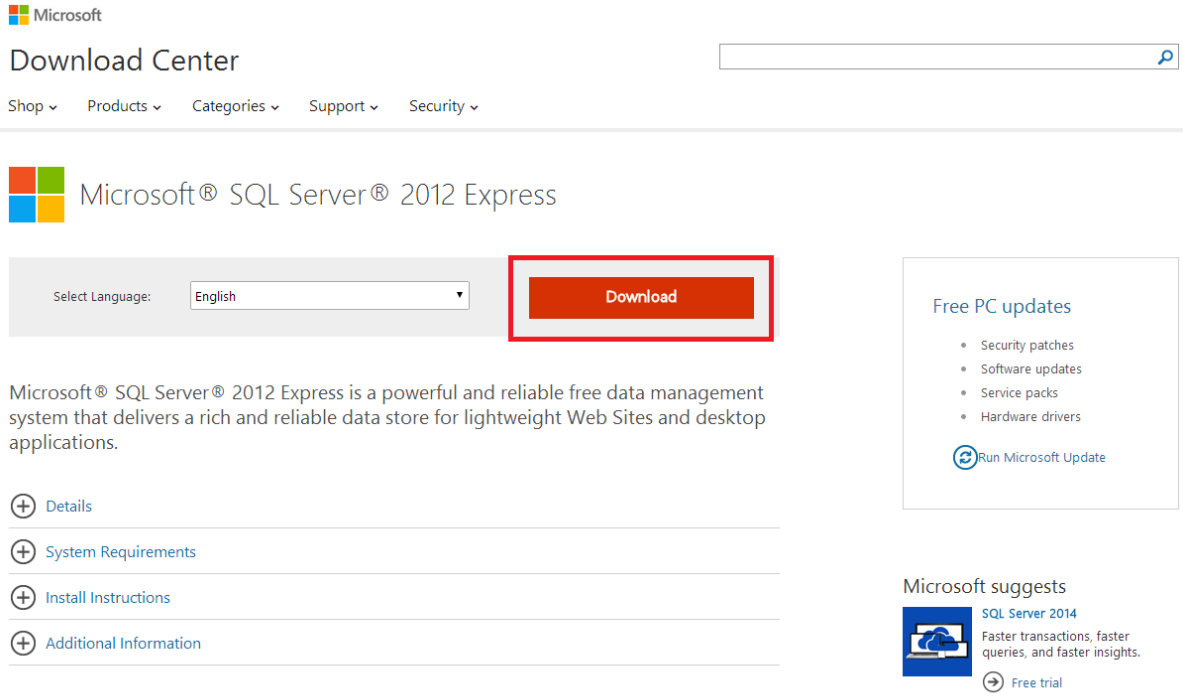


Trykk “Next”.

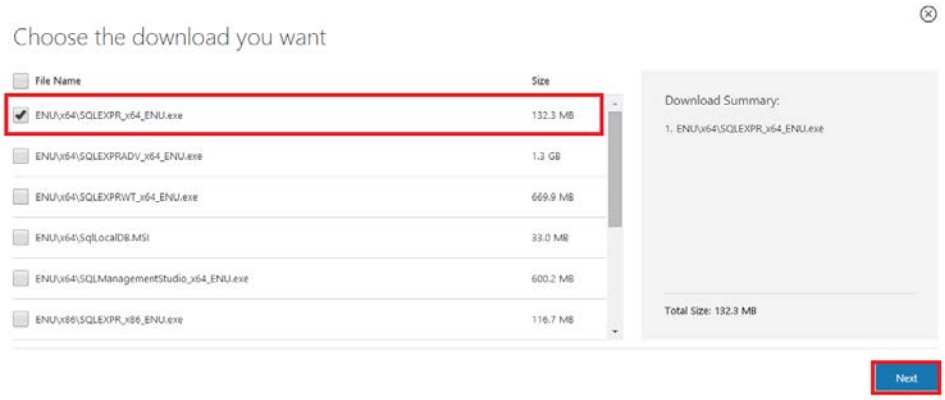


Trykk “Close”.

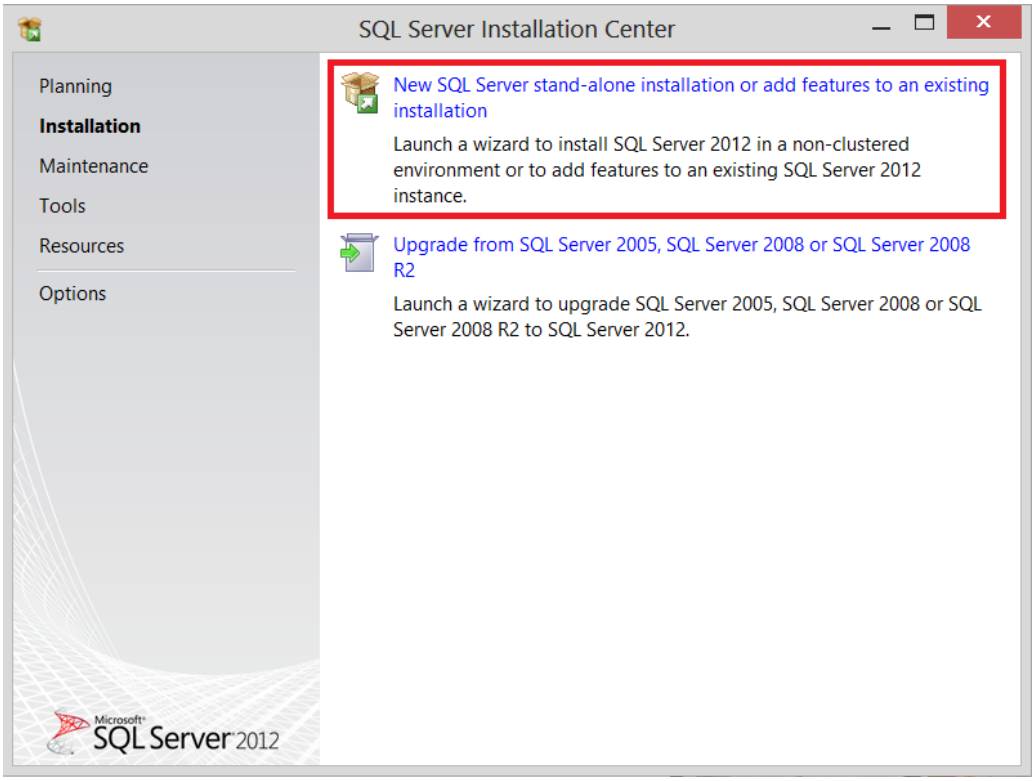
Open this link again:
http://www.microsoft.com/en-us/download/details.aspx?id=29062



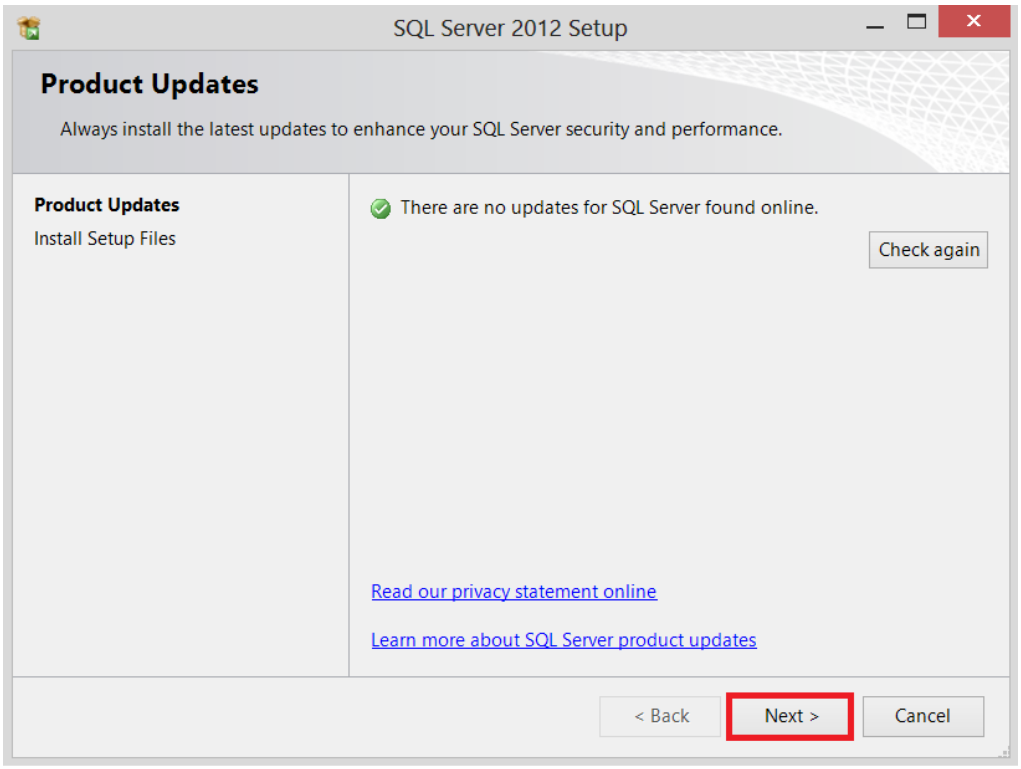
Trykk “Download” igjen.



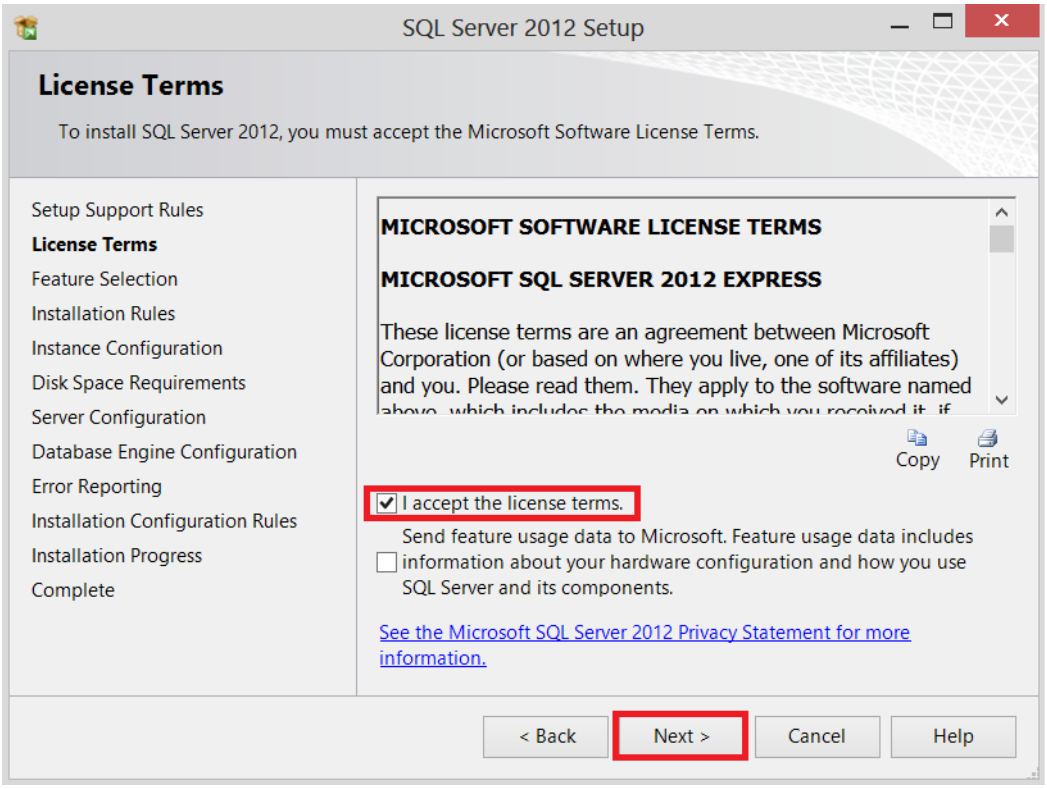
Velg “ENU\x64\SQLEXPR_x64_ENU.exe”.
Så trykk “Next”.
Kjør fila når den er ferdig nedlastet.



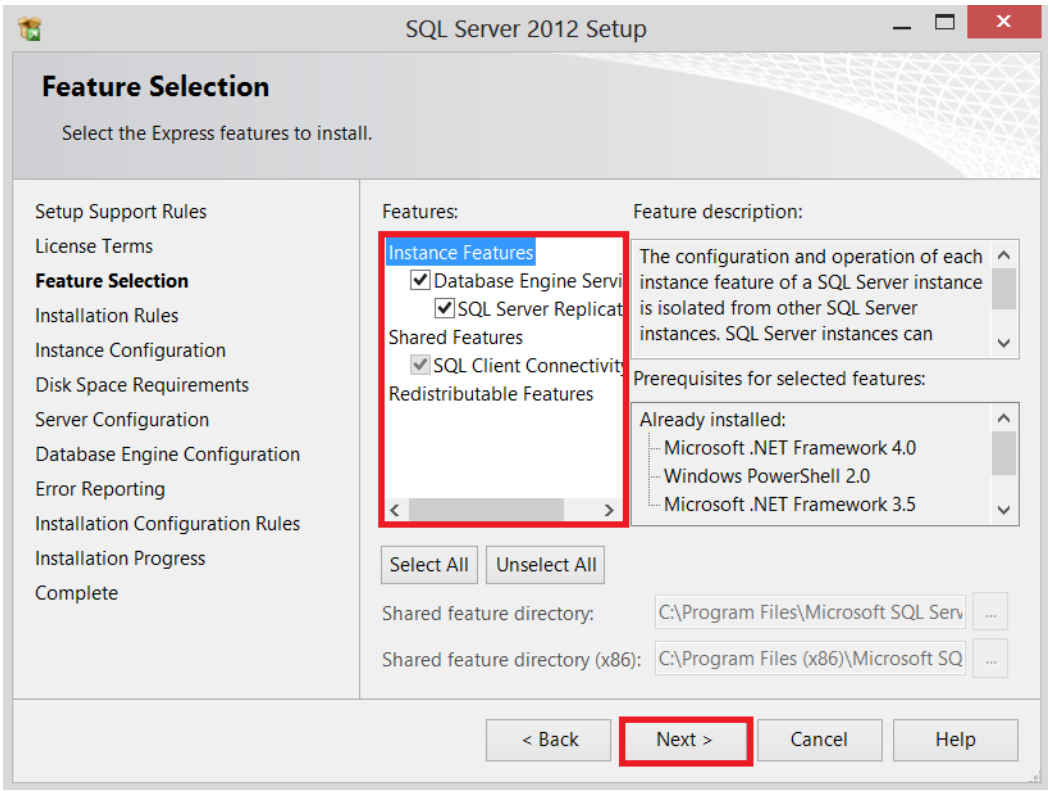
Trykk “New SQL Server stand-alone installation or add features to an existing installation”.



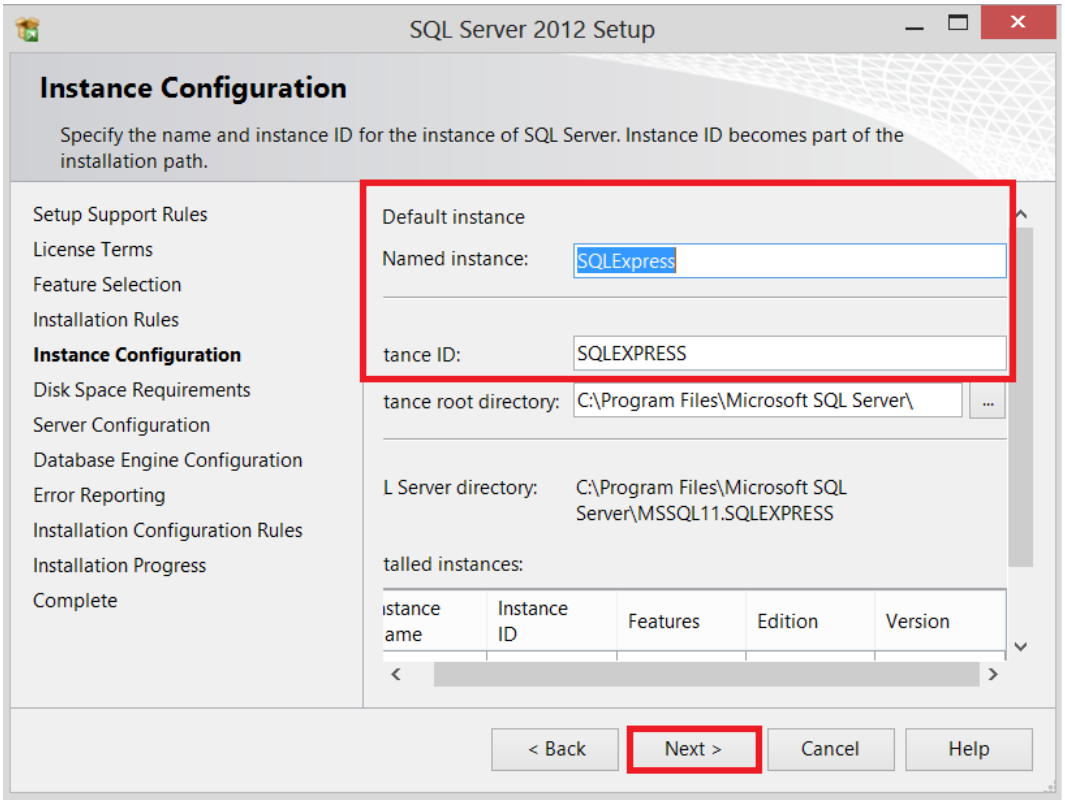
Trykk “Next”.



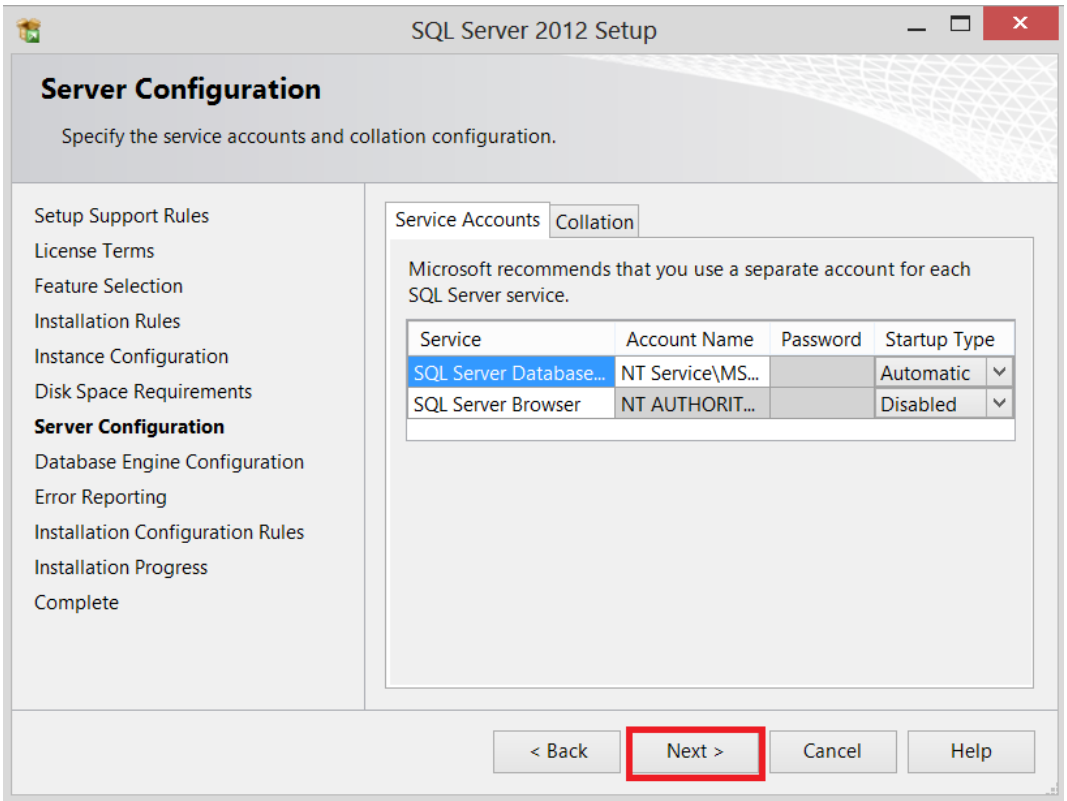
Huk av "I accept the license terms".
Deretter trykk "Next".



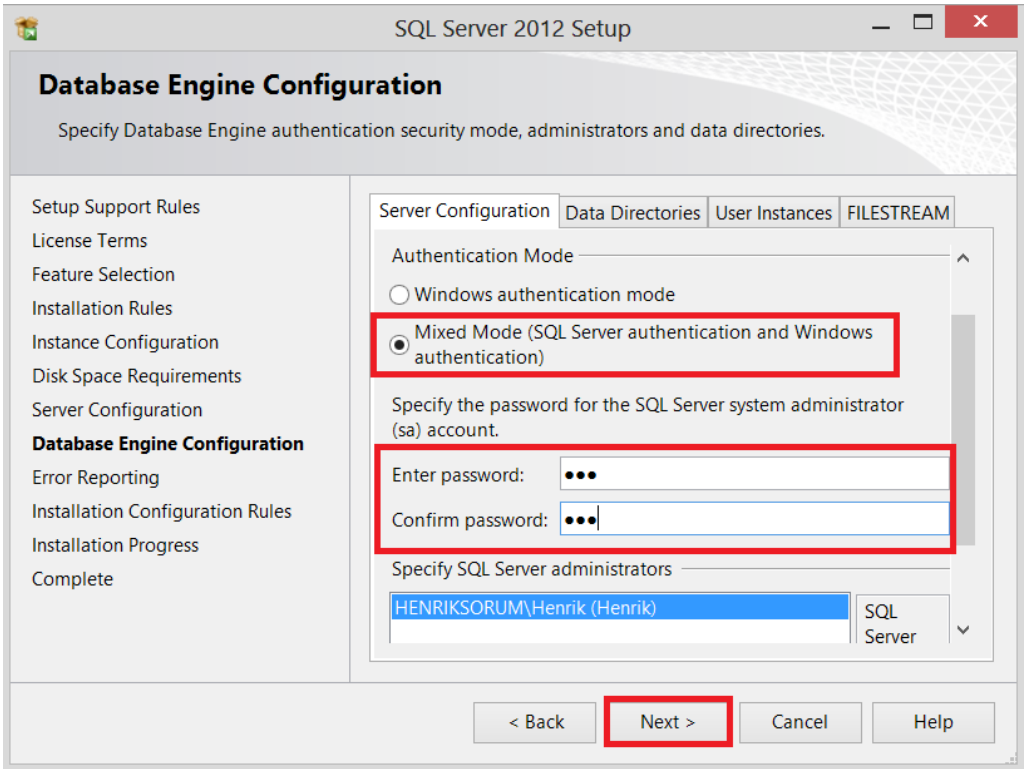
Features: Huk av alle checkbox-ene.
Trykk "Next".



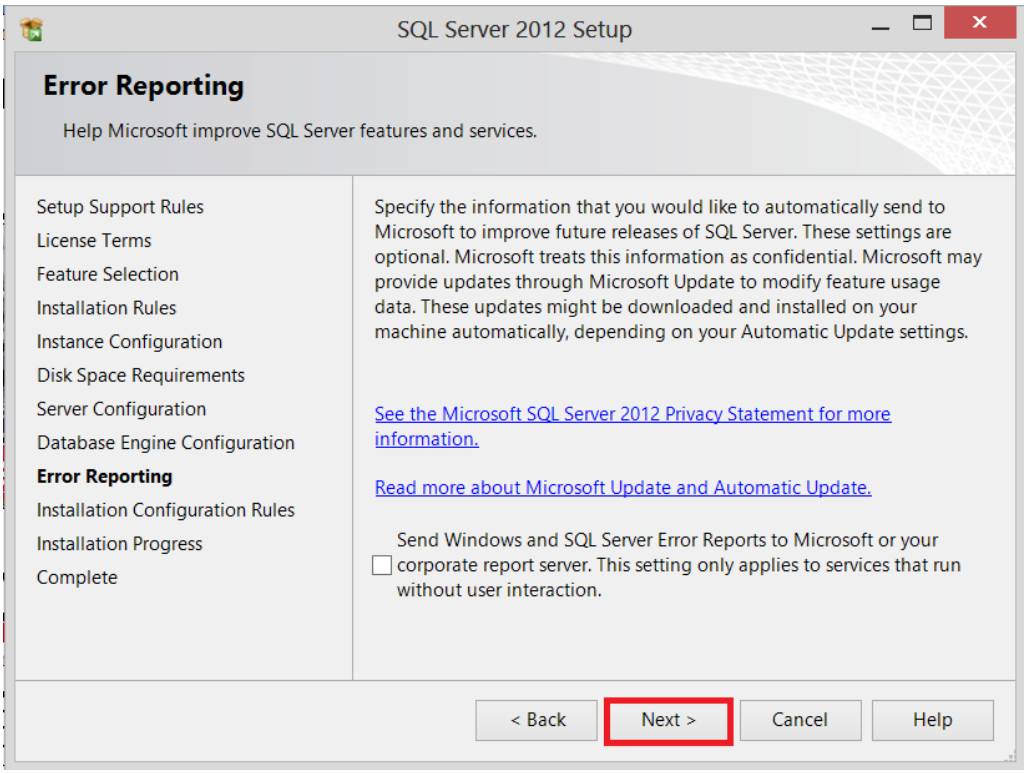
Sjekk at det står "SQLExpress".
Deretter trykk "Next".



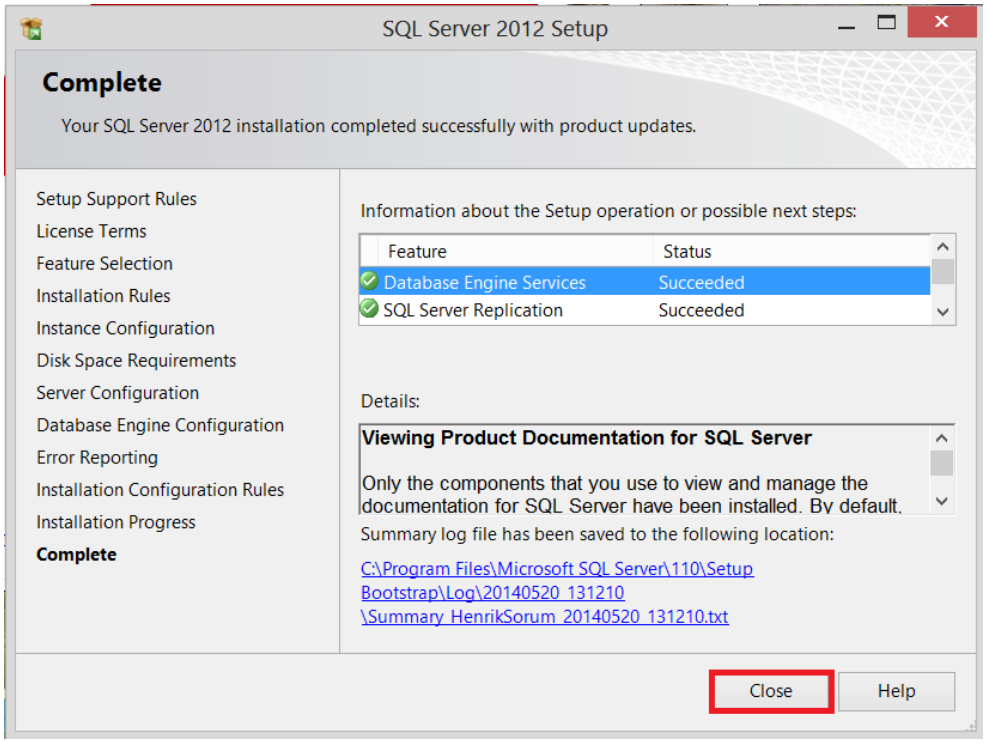
Trykk "Next".



Velg “Mixed Mode (SQL Server authentication and Windows authentication)”.
Passord: “act”.
Trykk “Next”.

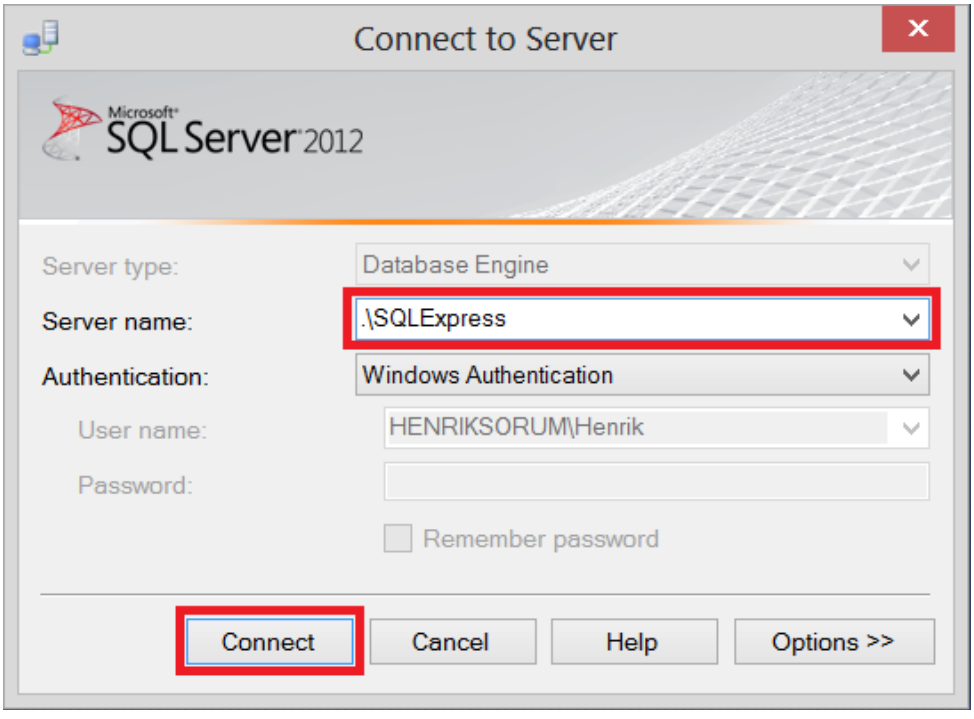


Trykk “Next”.

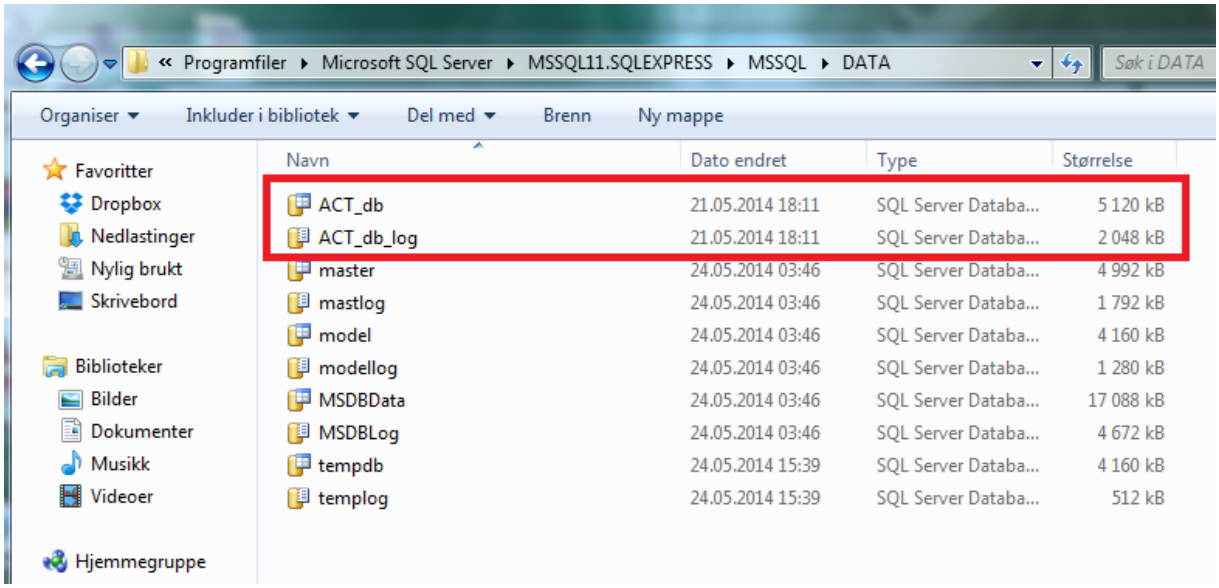


Trykk “Close”.

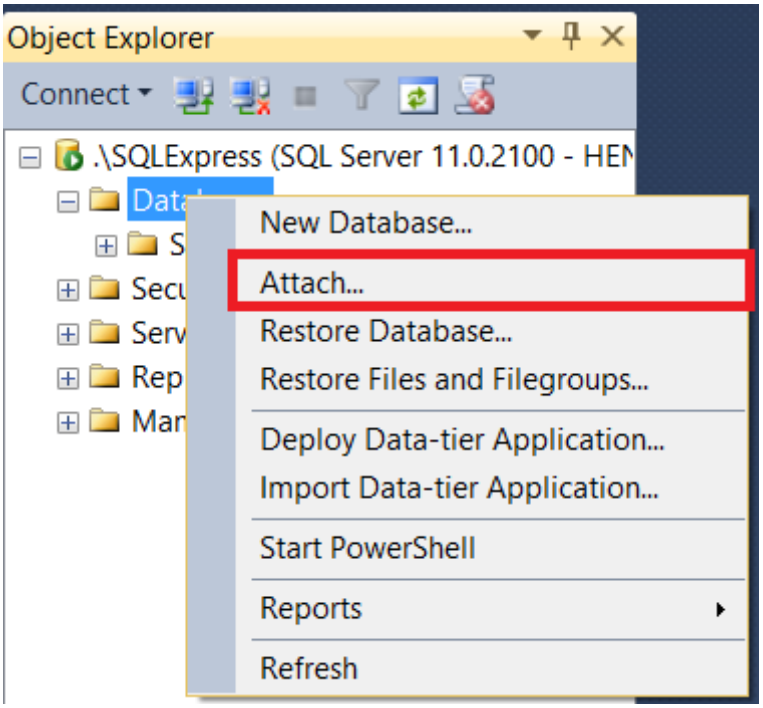
Åpne “Microsoft SQL Sever Management Studio 2012”.



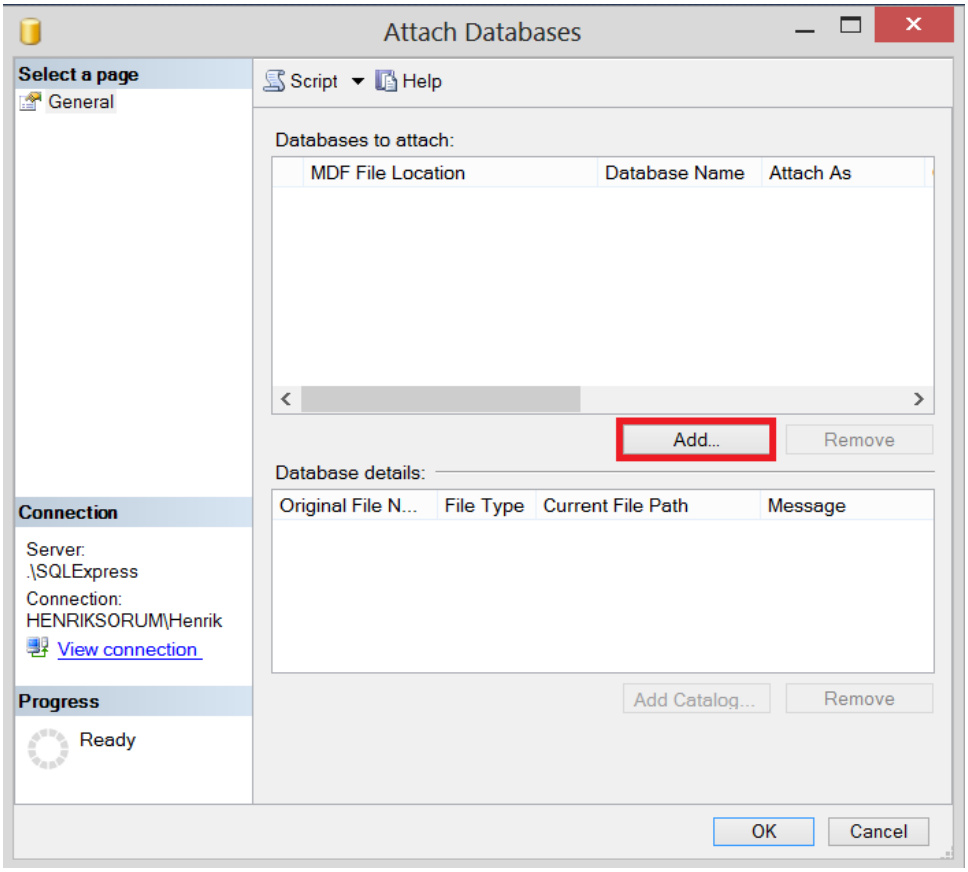
Server name: “.\SQLExpress”.
Deretter trykk “Connect”.



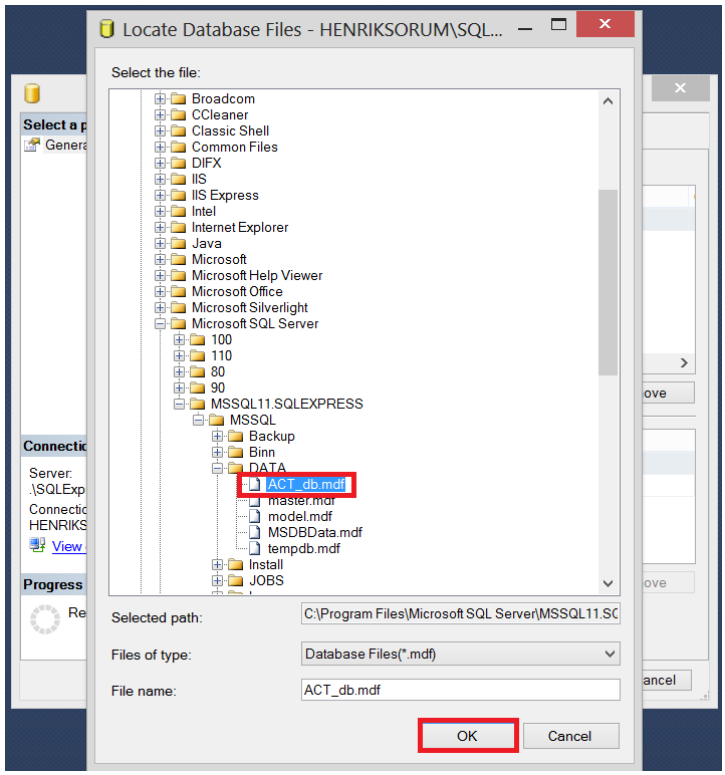
På cd'en finner du disse to filene. (De kan også lastes ned fra hjemmesiden vår etter den 27.05.14)
Åpne explorer og legg disse to filene til
C:\Program Files\Microsoft SQL Server\MSSQL11.SQLEXPRESS\MSSQL\DATA



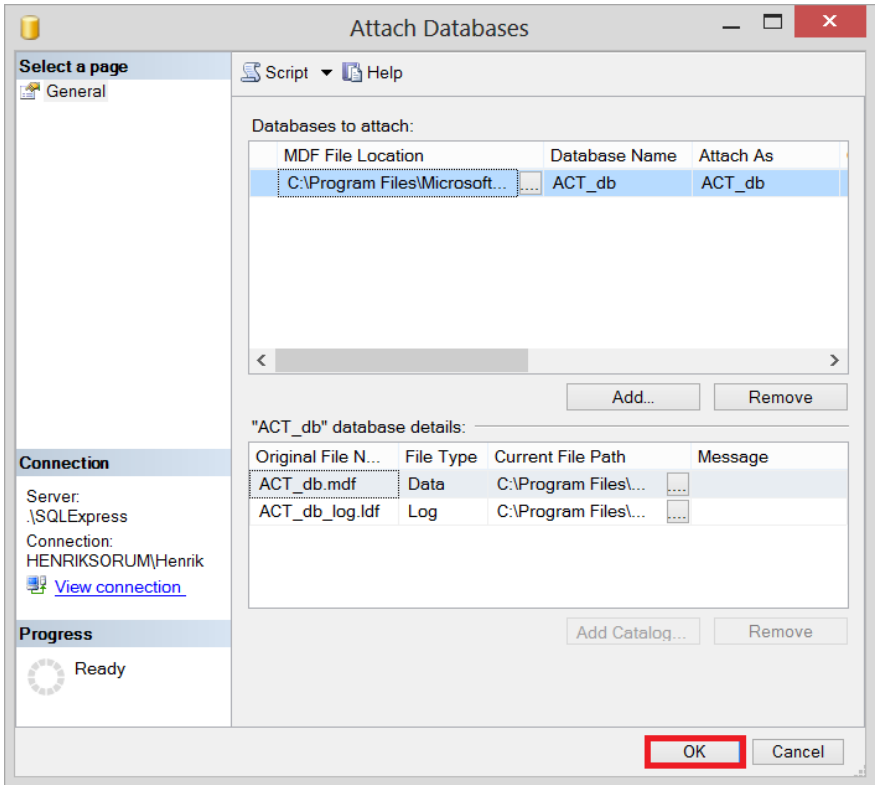
Når du er koblet til serveren høyreklikk på "Database" mappen.
Trykk "Attach…".



Trykk “Add…”.



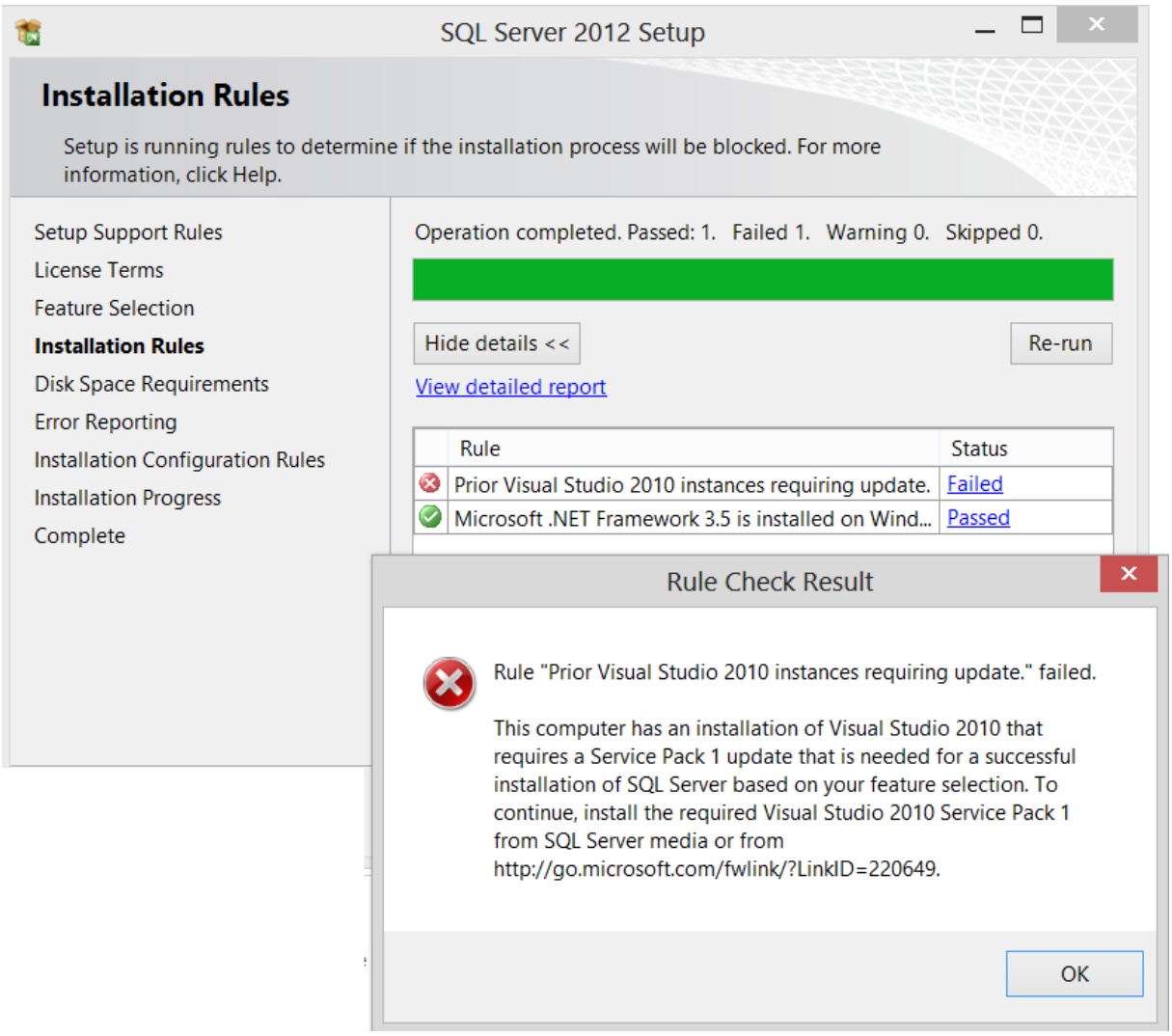
Velg “ACT_db.mdf”.
Trykk “OK”.



Trykk "OK".

Nå er serveren oppe og databasen er koblet til.

5.1 Error



Hvis du får denne error meldingen gå til denne linken:
http://go.microsoft.com/fwlink/?LinkID=220649
og installer Service Pack.

ACT AUTOMATED CROWS TESTING

ITERASJONSDOKUMENT

AUGUST KIND SVENDSEN
EIRIK LIEN ROA
HENRIK BERGE SØRUM
STÅLE RUDIN

KONGSBERG

HBV
HØGSKOLEN I BUSKERUD OG VESTFOLD
PROFESJONSHØGSKOLEN

Iterasjonsdokument			
PROSJEKT	ACT - Automated CROWS Testing		
OPPDRAGSGIVER	Kongsberg Protech Systems		
UTFØRT VED	HBV - Høgskolen i Buskerud og Vestfold		
GRUPPE	August Kind Svendsen, Eirik Lien Roa, Henrik Berge Sørum og Ståle Rudin		
VERSJON	2.0		
ANTALL SIDER	35		
DOKUMENTHISTORIE	VERSJON	UTGITT	BESKRIVELSE
	1.0	24.03.14	Første utgivelse
	2.0	26.05.14	Andre utgivelse

INNHOLDSFORTEGNELSE

LISTE OVER TABELLER

1 Om dokumentet

1.1 Dokumenthistorie

VERSJON	DATO	ENDRING	SIGNATUR
0.1	14.03.14	• Dokumentet ble opprettet	AKS, ELR, HBS, SR
1.0	21.03.14	• Første utgivelse	AKS, SR
1.1	16.04.14	• Lagt til rapport for konstruksjonsiterasjon #1	HBS
1.2	22.04.14	• Lagt til plan for konstruksjonsiterasjon #2	AKS
1.3	02.05.14	• Lagt til rapport for konstruksjonsiterasjon #2	SR
1.4	05.05.14	• Lagt til plan for konstruksjonsiterasjon #3	SR
1.5	16.05.14	• Lagt til rapport for konstruksjonsiterasjon #3	ELR
1.6	19.05.14	• Lagt til plan for konkluderende iterasjon	SR
2.0	25.05.14	• Lagt til rapport for konkluderende iterasjon • Lagt til konklusjon	SR

Tabell 1: Dokumenthistorie

1.2 Definisjoner og forkortelser

Liste som beskriver ord og uttrykk som kan være brukt i dette dokumentet:

UTRYKK	FORKLARING
HBV	Høgskolen i Buskerud og Vestfold
KPS	Kongsberg Protech Systems (oppdragsgiver)
ACT	Automated CROWS Testing
Industrirobot	Universal Robots 5
CROWS	Common Remotely Operated Weapon Station
RWS	Remote Weapon Station
DCP	Display Control Panel
MPU	Main Processing Unit
Konfigurasjon	Forskjellige CROWS oppsett

Tabell 2: Definisjoner og forkortelse

2 Innledning

Dette dokumentet er et iterasjonsdokument laget av prosjektgruppa ACT.

Dokumentet inneholder alle iterasjonene vi har delt prosjektet inn i. Hver iterasjon inneholder en iterasjonsplan og en iterasjonsrapport.

Vi har delt inn prosjektet i fire faser; Innledende, utdypende, konstruksjon og konkluderende fase. Her er det én iterasjon i innledende fase, to iterasjoner i utdypende fase, tre iterasjoner i konstruksjonsfasen og to iterasjoner i den konkluderende fasen.

En iterasjon er en del av en fase og er en periode der vi jobber med bestemte aktiviteter. Alle iterasjoner vil ha forskjellige aktiviteter i fokus. F.eks. så vil det være fokus på software-aktiviteter i konstruksjonsfasen, og dokumentasjon og planleggingsaktiviteter i den innledende og utdypende fase. Det skal også gjøres andre aktiviteter som er nødvendig gjennom iterasjonen, som dokumentasjon av hva vi skal gjøre, eller planlegger å gjøre er noe som vil oppdateres gjennom alle iterasjoner.

Planen skal skrives i starten av hver iterasjon og skal inneholde en plan over hva som skal gjøres i den spesifikke iterasjonen. Det skal skrives et mål for iterasjonen, nærmere sagt hva vi ønsker å oppnå med denne iterasjonen. Det skal skrives en aktivitetsliste med alle aktiviteter som skal jobbes med, hvem som er ansvarlig for aktiviteten, hvem som skal jobbe på aktiviteten og et forventet resultat. Det skal også skrives en mer detaljert oversikt over aktivitetene, der vi skal gå mer inn på spesifikke ting som skal gjøres innen en aktivitet.

Iterasjonsrapporten skal være en rapport som skrives etter hver iterasjon. Denne rapporten skal inneholde aktivitetslisten fra planen, som viser hvilket resultat man oppnåde i iterasjonen, og eventuelle ufordringer og tiltak. Samt en konklusjon.

Risikoanalyse og aktiviteter er hentet fra prosjektplan[1]. Fargekode i risikoanalysen og tabellene er henter fra prosjektplan. Alle aktiviteter og ID er hentet fra prosjektplan.

3 Iterasjoner

3.1 Innledende iterasjon

Periode: 13.01.14 – 20.02.14

3.1.1 Iterasjonsplan for innledende iterasjon

Mål

Målet med den iterasjonen vi har i den innledende fasen er å få på plass utstyr og dokumenter som skal hjelpe oss videre i prosjektet. Med utstyr mener vi hardware vi trenger for å kunne få satt opp et fungerende system. Hovedsakelig da industrirobot og våpenstasjon med tilhørende utstyr.

Det er veldig viktig at vi blir enige med arbeidsgiver om hva resultatet av prosjektet skal være i denne iterasjonen. Derfor må vi få satt på plass de forskjellige kravene som definerer systemets egenskaper. Til kravene må vi også sette noen tilhørende tester som gir oss muligheten til å evaluere status til kravet.

For å kunne sette spesifikke tester er vi nødt til å planlegge mer konkret hvordan vi har tenkt å løse utfordringene teknisk sett. Det innebærer å bli enige om hvordan kommunikasjon vi har tenkt å bruke mellom hardware, og hvilket språk og utviklings-software vi hovedsakelig skal arbeide i.

Vi må også finne ut om vi skal jobbe på eget rom på skolen eller om vi får et dedikert kontor hos arbeidsgiver.

Oppgaven krever at vi får på plass ekstern veileder og sensor. Dette er i først og fremst arbeidsgivers oppgave å velge, men vi må videreformidle dette og informere skolen.

Risikoanalyse

I denne iterasjonen vil vi prioritere kravspesifikasjonen og prosjektplanen vår høyest. Dette gjør vi på grunn av at kravspesifikasjonen vil danne grunnlaget for hva vi må jobbe ut ifra, og en god prosjektplan vil være viktig for at vi skal gjennomføre et godt prosjekt. Dersom vi ikke klarer å få på plass disse tidlig i prosjektet kan det ha store konsekvenser for hvordan vi jobber resten av prosjektet.

RISIKOFAKTORER	ÅRSAK	S	K	R	RISKOREDUSERENDE TILTAK
Tap av data	Feil på hardware og software.	1	5	5	Vi reduserer denne risikoen ved å ha siste versjon lagret på minst en minnebrikke, vi har et dokumenthåndteringssystem der vi regelmessig tar sikkerhetskopi av dokumenter og filer.
Sykdom eller fravær	N/A.	3	2	6	Ingen tiltak nødvendig under normale omstendigheter.
Feilestimering av tidsbruk	Ikke god nok planlegging	2	3	6	Bruke nok tid til planlegging, og ta med potensielle tidskrevende utfordringer i planleggingen.
Prosjektplan	Ikke ferdig	1	5	5	Sette av nok tid til prosjektplan. I tillegg har vi satt høy prioritet for å få denne ferdig.
Kravspesifikasjon	Ikke ferdig	1	5	5	Sette av nok tid. Den vil også være høy prioritet.

Tabell 3: Risikoanalyse for innledende iterasjon #1

Aktiviteter

AKTIVITET	ANSVARLIG	FRIST	RESSURSER	RESULTAT	ID
Prosjektplan	Henrik Berge Sørum	18.02.14	Alle	Prosjektplan	1200
Kravspesifikasjon	August Kind Svendsen	18.02.14	AKS, HS	Kravdokument	3000
Testspesifikasjon	Henrik Berge Sørum	18.02.14	AKS, HS	Testdokument	5100
Møter	Eirik Lien Roa		ELR		1000
Timelister	Ståle Rudin	01.02.24	SR	Timelistemal	1300
Presentasjon 1	Eirik Lien Roa	20.02.14	Alle	Powerpoint, presentasjon	2000

Tabell 4: Aktiviteter for innledende iterasjon #1

Aktiviteter detaljert

Til første presentasjon må vi ha skrevet en versjon av prosjektplan, kravspesifikasjon og testspesifikasjon. Til disse dokumentene må vi bli enige om en konsekvent måte å skrive dokumentene på. Det vil si at oppsett av overskrifter, font og layout er likt overalt. Det ønskes også at vi får en god forside som forteller hva dokumentet heter og hvem som står bak det.

Prosjektplanen må inneholde:

- Mal og rammer for prosjektet
- Avgrensninger og forutsetninger
- Hvordan skal vi gjennomføre prosjektet?
- Møter som skal gjennomføres
- Aktivitetsliste

Kravspesifikasjonen setter vi i samarbeid med arbeidsgiver. Planen der er å dele kravene inn i følgende kategorier:

Rammekrav, sikkerhetskrav, funksjonelle krav, utstyrskrav og andre krav.

Testspesifikasjonen tar for seg de kravene som har blitt satt og forteller hvordan vi har tenkt å gå frem for å evaluere tilstanden til kravet. Her må vi analysere kravet for å komme frem til best mulig testmetode. Det må også settes et godkjenningskriterium som vi kan sjekke opp mot.

Det å arrangere møter krever at møtearrangør får hørt med alle parter når det passer og deretter kalle inn til passende tidspunkt og lokasjon.

Vi skal lage en mal for timelistene slik at hvert gruppemedlem enkelt kan taste inn timer uten å måtte sette av mye tid til det. Her er det viktig at vi får et oversiktlig dokument som automatisk legger sammen totalsummen, slik at vi ikke trenger å regne om på dette hver gang det føres timer.

3.1.2 Iterasjonsrapport for innledende iterasjon

Aktiviteter fra iterasjonsplan

AKTIVITET	ANSVARLIG	FRIST	RESSURSER	RESULTAT	ID
Prosjektplan	Henrik Berge Sørum	18.02.14	Alle	Prosjektplan	1200
Kravspesifikasjon	August Kind Svendsen	18.02.14	AKS, HS	Kravdokument	3000
Testspesifikasjon	Henrik Berge Sørum	18.02.14	AKS, HS	Testdokument	5100
Møter	Eirik Lien Roa		ELR		1000
Timelister	Ståle Rudin	01.02.24	SR	Timelistemal	1300
Presentasjon 1	Eirik Lien Roa	20.02.14	Alle	Powerpoint, presentasjon	2000

Tabell 5: Aktiviteter fra innledende iterasjon #1

Overholdelse

Aktivitetene vi planla i iterasjonsplanen har blitt overholdt. Vi har produsert følgende resultater/artefakter:

- Prosjektplan
- Kravspesifikasjon
- Testspesifikasjon
- Timelistemal
- Presentasjon 1 med PowerPoint

Vi har fått tildelt følgende sensorer og veiledere:

- Intern veileder: Daniel Larsson
- Ekstern veileder: Dag Christian Nygaard
- Ekstern sensor: Hallvard Murberg

Utfordringer og tiltak

I oppstartsfasen har det vært mye å sette seg inn i. Det har for eksempel tatt oss tid å analysere og gjøre research på hvilken prosjektmodell som er lurt å jobbe ut i fra. Å evaluere hva prosjektplanen skal inneholde var en overraskende tidskrevende prosess. Når vi klarte å komme frem til hva vi skal ha med og bruke har skriveprosessen gått greit.

I kravspesifikasjonen har det vært litt vanskelig å presisere kravene til systemet da vi ikke helt vet de tekniske spesifikasjonene. Tiltaket her er å sette opp et nytt møte med arbeidsgiver sånn at vi kan gå over kravene sammen og presisere disse.

Konklusjon

Vi kan konkludere med at den første iterasjonen har gått som forventet. Siden prosjektarbeid med dokumentasjon på et så stort prosjekt som dette er noe nytt for alle på gruppa, har vi antatt at vi vil få tilbake litt konstruktiv kritikk etter første presentasjon. Sentralt står en vurdering av prosjektmodellen. Dette vil vi jobbe med deler av neste iterasjon.

3.2 Utdypende iterasjon #1

Periode: 21.02.14 – 07.03.14

3.2.1 Iterasjonsplan for utdypende iterasjon #1

Mål

I denne iterasjonen skal vi jobbe med alle tilbakemeldingene vi fikk under første presentasjon. Vi fikk tilbakemelding om at prosjektmodellen og gantt-diagrammet ikke stemte overens med hverandre. Derfor må vi finne en løsning på hvordan vi skal fikse dette. Eventuelt endre prosjektplan, eller endre gantt-diagrammet slik at det passer med våres prosjektmodell. Når vi gjør slike endringer må vi også endre tilhørende informasjon om disse diagrammene.

Noe vi måtte endre på var kravene våres, fordi det var for mange av dem som var satt av gruppa. Målet er da å få til et møte med KPS og diskutere disse.

Vi skal også starte og designe systemet våres. Vi vil starte med å lage et software design dokument. Fokuset her blir å lage diverse UML diagrammer. Blant annet har vi tenkt å ta med «use case», «klassediagram», «sekvensdiagram» og «deploymentdiagram».

Hvis det blir tid skal vi også begynne og gjøre research på TCP/IP, parsing og bildegjenkjenning.

Risikoanalyse

Hvis vi mot formodning ikke skulle bli ferdig med alle aktivitetene i denne iterasjonen vil det ikke ha en veldig stor betydning for prosjektet. For dette er den første av to iterasjoner i den utdypende fasen. Noen av aktivitetene som er i denne iterasjonen strekker seg over begge iterasjonene i den utdypende fasen.
Det som vi kommer til å prioritere i denne iterasjonen er først og fremst å få rettet opp i prosjektmodellen vår, for uten en ordentlig prosjektmodell vil ha store konsekvenser for resten av prosjektet vårt. I tillegg blir det viktig å få rettet opp og få på plass en god kravspesifikasjon så kjapt som mulig. En kravspesifikasjon er grunnlaget som vi må jobbe etter gjennom hele prosjektet, og vil være viktig for at prosjektet skal lykkes. Disse to aktivitetene vil ha størst fokus i denne iterasjonen.

RISIKOFAKTORER	ÅRSAK	S	K	R	RISKOREDUSERENDE TILTAK
Tap av data	Feil på hardware og software.	1	5	5	Vi reduserer denne risikoen ved å ta jevnlige sikkerhetskopier til lokal mappe på en PC og minnebrikke.
Sykdom eller fravær	N/A.	3	2	6	Ingen tiltak nødvendig under normale omstendigheter.
Feilestimering av tidsbruk	Ikke god nok planlegging	2	3	6	Bruke nok tid til planlegging, og ta med potensielle tidskrevende utfordringer i planleggingen.
Prosjektmodell	Ikke ferdig	1	5	5	Denne vil få høy prioritet og det vil bli satt av god tid til dette.
Kravspesifikasjon	Ikke ferdig	1	5	5	Denne vil bli prioritert høyt og vi har satt av møte med KPS for å få dette gjort.

Tabell 6: Risikoanalyse for utdypende iterasjon #1

Aktiviteter

AKTIVITET	ANSVARLIG	FRIST	RESSURSER	RESULTAT	ID
Prosjektplan	Henrik Berge Sørum	07.03.14	ELR, AKS, HBS, SR	Oppdatert prosjektplan	1200
Kravspesifikasjon	August Kind Svendsen	07.03.14	ELR, AKS, HBS, SR	Oppdaterte krav	3000
Software designdokument	August Kind Svendsen	07.03.14	ELR, AKS	Forklarende diagrammer	3200
Kommunikasjon	Ståle Rudin	07.03.14	HBS, SR	Kunnskap	4020
Bildebehandling	Eirik Lien Roa	07.03.14	ELR	Kunnskap	4030
Iterasjonsdokument	Ståle Rudin	07.03.14	SR	Iterasjonsplan og rapport for Utdypende iterasjon #1	1500

Tabell 7: Aktiviteter fra utdypende iterasjon #1

Aktiviteter detaljert

Prosjektplan:
I denne aktiviteten skal vi først bli enige om hva vi skal gjøre med de kommentarene vi fikk under første presentasjon. Her må vi gjøre et valg om vi enten skal endre hele prosjektplanen, eller om vi skal endre på gantt-diagrammet slik at det passer med vannfallsmodellen vi allerede har. Hvis dette er tilfellet må vi endre på all tilhørende tekst som har med planen og gjøre. I tillegg må vi da lage nye diagrammer som beskriver den nye planen. Hvis dette ikke er tilfellet og vi bestemmer oss for å endre på den nåværende modellen trenger vi bare og lage nytt eller endre på det gamle GANTT diagrammet.

Kravspesifikasjon:
Vi har tenkt å sette opp et møte med KPS og diskutere kravene våres. Dette er for å bli enige om hvilke krav som kan være som de er, hvilke vi må endre og hvilke som må fjernes.

Software design dokument:
Det er i denne iterasjon vi skal begynne å designe systemet. Vi har da valgt å vise dette ved hjelp av UML diagrammer. Dette er fordi det gir et visuelt inntrykk av hvordan systemet skal fungere og hvilke klasser som skal ha kommunisere med hverandre.

TCP/IP:
Gjøre research om hvordan vi skal få kommunikasjon mellom server til både våpenstasjon og robot. For å få til dette må vi i tillegg skaffe oss kunnskap om både «mutual exclusion» og «regular expressions». Dette er for å kunne pakke dataen, sende dataen og pakke den ut på mottaker sin side. Kort fortalt skal det være en toveis kommunikasjon mellom PC(klient) og våpenstasjon(server), og PC(server) og robot(klient).

Bildebehandling:
Gjøre research på å fange opp bilde fra DCP-en og lagre det. Deretter å hente ut tekst fra dette bildet som vi kan bruke for å sammenligne om teksten stemmer overens med det som skal stå der.

3.2.2 Iterasjonsrapport for utdypende iterasjon #1

Aktiviteter fra iterasjonsplan

AKTIVITET	ANSVARLIG	FRIST	RESSURSER	RESULTAT	ID
Prosjektplan	Henrik Berge Sørum	07.03.14	ELR, AKS, HBS, SR	Oppdatert prosjektplan	1200
Kravspesifikasjon	August Kind Svendsen	07.03.14	ELR, AKS, HBS, SR	Oppdaterte krav	3000
Software designdokument	August Kind Svendsen	07.03.14	ELR, AKS	Forklarende diagrammer	3200
Kommunikasjon	Ståle Rudin	07.03.14	HBS, SR	Kunnskap	4020
Bildebehandling	Eirik Lien Roa	07.03.14	ELR	Kunnskap	4030
Iterasjonsdokument	Ståle Rudin	07.03.14	SR	Iterasjonsplan og rapport for Utdypende iterasjon #1	1500

Tabell 8: Aktiviteter fra utdypende iterasjon #1

Overholdelse

Prosjektplan:
Vi valgte å skifte ut vannfallsmodellen vår med UP (Unified Process). Når vi gjorde dette måtte vi lage ny GANTT og endre all tilhørende tekst i prosjektplanen som hadde med prosjektmodellen og gjøre.

Kravspesifikasjon:
Kravene er uendret så langt, men skal ha møte med KPS i neste iterasjon.
Software design dokument:
Vi har produsert følgende: use case-, deployment-, klasse- og sekvensdiagram.
TCP/IP:
Startet å gjøre research. Vi satte opp et enkelt program som fikk kontakt med våpenstasjon.
Parsing:
Startet research rundt parsing, spesiell vekt på regular expressions. Dette er en måte å fange opp spesielle ord/setninger eller annen bestemt data, der dette kan være gjemt blant annen irrelevant data.
Bildebehandling:
Startet research. Uthentet bilde fra DCP, og fant bibliotek for å hente ut tekst fra dette bildet.

Utfordringer og tiltak

En utfordring i denne iterasjonen har vært å designe systemet. For å få til dette hadde vi et par møter med en ekspert på software design. Utenom det har research og dokumentasjon gått etter plan. Skiftet prosjektmodell og derfor laget dette dokumentet som forklarer hva som skal gjøres og har blitt gjort i hver iterasjon. I tillegg er det en risikoanalyse over hver iterasjon der vi bestemmer oss for hva som får prioritet etter en risikoanalyse er tatt.

Konklusjon

Vi er nå midt i den utdypende fasen og har allerede et godt inntrykk av hvordan systemet kommer til å se ut. Føler vi har et godt utgangspunkt når vi går over til siste iterasjon i den utdypende fasen.

3.3 Utdypende iterasjon #2

Periode: 10.03.14 – 14.03.14

3.3.1 Iterasjonsplan

Mål

Målene for denne iterasjonen er å gjøre oss klare til å starte på konstruksjonsfasen. Det vil si at planlaggende dokumentasjon skal være ferdig og alle skal ha en forståelse om hva som skal gjøres i neste iterasjon. Alle skal derfor gjøre research innenfor sitt eget emne i software biten. Dette vil gjøre det lettere å starte med programmeringen i neste iterasjon. Det skal også være laget en oppdatert versjon av krav og testspesifikasjonen som er laget i samarbeid med bedrift. I tillegg skal vi fortsette med software design-dokumentet som ble startet i forrige iterasjon.

Risikoanalyse

Det er viktig at alle aktivitetene i denne iterasjonen blir ferdig før vi går inn i konstruksjonsfasen. Vi får ikke startet å lage hovedprogrammet til systemet vårt før planleggingen er ferdig. Hvis vi starter uten at planleggingen er ferdig vil det være vanskelig å integrere de forskjellige delene hvis man ikke vet hvordan de skal kommunisere med hverandre. Derfor er det mye fokus på design-dokumentet i denne iterasjonen.

RISIKOFAKTORER	ÅRSAK	S	K	R	RISKOREDUSERENDE TILTAK
Tap av data	Feil på hardware og software.	1	5	5	Vi reduserer denne risikoen ved å ha siste versjon lagret på minst en minnebrikke, vi har et dokumenthåndteringssystem der vi regelmessig tar sikkerhetskopi av dokumenter og filer.
Sykdom eller fravær	N/A.	3	2	6	Ingen tiltak nødvendig under normale omstendigheter.
Feilestimering av tidsbruk	Ikke god nok planlegging	2	3	6	Bruke nok tid til planlegging, og ta med potensielle tidskrevende utfordringer i planleggingen .
Software designdokument	Ikke ferdig	3	3	9	Høy prioritering og legge inn mer tid hvis det er nødvendig.

Tabell 9: Risikoanalyse for utdypende iterasjon #2

Aktiviteter

AKTIVITET	ANSVARLIG	FRIST	RESSURSER	RESULTAT	ID
Software designdokument	August Kind Svendsen	14.03.14	AKS, ELR	Planleggende diagrammer	3200
Prosjektplan	Henrik Berge Sørum	14.03.14	AKS, ELR, HBS, SR	Oppdatert prosjektplan	1200
Kravspesifikasjon	August Kind Svendsen	14.03.14	Alle	Oppdatere kravspesifikasjon	3000
Testspesifikasjon	Henrik Berge Sørum	14.03.14	ELR	Oppdatere testspesifikasjon	5100
Iterasjonsdokument	Ståle Rudin	14.03.14	AKS, ELR, HBS, SR	Iterasjonsplan og rapport for Utdypende iterasjon #2	1500
Kommunikasjon	Ståle Rudin	14.03.14	AKS, HBS, SR	Satt seg inn i TCP/IP	4020
Bildebehandling	Eirik Lien Roa	14.03.14	ELR	Satt seg inn i bildegjenkjenning	4030
Møter	Eirik Lien Roa		Alle		1000

Tabell 10: Aktiviteter fra utdypende iterasjon #2

Aktiviteter detaljert

Software design dokument:
Alle planleggende UML-diagrammer skal gjøres ferdig i denne iterasjonen. Disse diagrammene beskriver hvordan systemet skal se ut og hvordan det skal utføres. Vi skal også skrive om de forskjellige klassene, brukergrensesnittet og generelt om systemet.

Prosjektplan:
Prosjektplanen må oppdateres siden vi har valgt å endre prosjektmodell. Vi må derfor gjøre de siste oppdateringene som ikke ble gjort forrige iterasjon. Gantt-diagrammet må gjøres ferdig og limes inn i dokumentet.

Kravspesifikasjon:
Kravspesifikasjonen var en av tingene som fikk kommentarer fra sensorer under første presentasjon. Derfor ble vi enige om at vi skulle prate med bedriften og oppdatere kravspesifikasjonen slik at det er kunden som skal komme med de fleste kravene. Vi skulle også gjøre om en del av kravene så de blir mer spisset inn og lettere å teste. Det vi skal gjøre i denne iterasjonen er å ha et møte med KPS der vi diskuterer kravene og endre, slette eller legge til nye krav. Når møtet er ferdig skal vi da også bruke noe tid til å gjøre kravspesifikasjonen mer utfyllende.

Testspesifikasjon:
Testspesifikasjonen fikk også noen kommentarer på første presentasjon. Siden kravspesifikasjon skal endres må også testspesifikasjonen endres i henhold til det nye kravspesifikasjonsdokumentet. Vi må se på de nye kravene og komme opp med måter å teste disse på.

Iterasjonsdokument:
Iterasjonsdokument må opprettes. Dette dokumentet skal inneholde planer og rapporter for hver enkelt iterasjon. Det skal skrives en plan før hver iterasjon, og en rapport når en iterasjon er ferdig. Rapporten oppsumerer iterasjonen og beskriver hva som kunne vært gjort annerledes hvis det trengs.

Siden prosjektmodellen er endret skal det skrives plan og rapport for de tre første iterasjonene i denne iterasjonen. Iterasjonsdokumentet vil være et eget dokument.

Denne uken skal det også fortsettes på research. Dette er en viktig for å kunne planlegge de neste iterasjonene i konstruksjonsfasen. Mer vi vet om de forskjellige protokollene og delene av systemet før vi starter og programmere desto lettere blir det å programmere. Vi skal derfor se på de oppgavene som vi skal starte på i konstruksjons iterasjon #1. Disse oppgavene er TCP/IP, der vi også må se på «mutual exclution» og «regular expressions». Vi må også se på bildegjenkjenning, her må vi finne ut hvordan vi kan hente ut tekst fra bildet som er hentet fra DCP-en.

3.3.2 Iterasjonsrapport for utdypende iterasjon #2

Aktiviteter fra iterasjonsplan

AKTIVITET	ANSVARLIG	FRIST	RESSURSER	RESULTAT	ID
Software designdokument	August Kind Svendsen	14.03.14	AKS, ELR	Planleggende diagrammer	3200
Prosjektplan	Henrik Berge Sørum	14.03.14	AKS, ELR, HBS, SR	Oppdatert prosjektplan	1200
Kravspesifikasjon	August Kind Svendsen	14.03.14	Alle	Oppdatere kravspesifikasjon	3000
Testspesifikasjon	Henrik Berge Sørum	14.03.14	ELR	Oppdatere testspesifikasjon	5100
Iterasjonsdokument	Ståle Rudin	14.03.14	AKS, ELR, HBS, SR	Iterasjonsplan og rapport for Utdypende iterasjon #2	1500
Kommunikasjon	Ståle Rudin	14.03.14	AKS, HBS, SR	Satt seg inn i TCP/IP	4020
Bildebehandling	Eirik Lien Roa	14.03.14	ELR	Satt seg inn i bildegjenkjenning	4030
Møter	Eirik Lien Roa		Alle		1000

Tabell 11: Aktiviteter fra utdypende iterasjon #2

Overholdelse

Alle planlagte resultater til aktivitetene har blitt overholdt.
Her er det en liste over resultatene vi har oppnådd:

- Software designdokument : Use case, klasse, deployment, og sekvensdiagram er opprettet.
- Prosjektplanen er oppdatert med ny prosjektmodell, nytt gantt-diagram, fremdriftsplan er endret, prosjektfasene er endret, generell beskrivelse av iterasjonene er opprettet og aktivitetslisten er oppdatert.
- Kravspesifikasjonen er oppdatert etter møte vi hadde med KPS, vi gikk gjennom hele kravspesifikasjonen på møtet og diskuterte alle krav og endret, slettet og lagde nye ettersom hva vi ble enige om. KPS ville at vi skulle gjøre dokumentet ferdig etter møte og sende den nye kravspesifikasjonen. KPS så over dokumentet på nytt og tilkalte til enda et møte for en siste finpuss på kravspesifikasjonen.
- Testspesifikasjonen er oppdatert i henhold til den oppdaterte kravspesifikasjonen. KPS vil at vi sender denne sammen med kravspesifikasjonen så de kan se over begge dokumentene.
- Iterasjonsdokumentet er opprettet, og iterasjonsplan og iterasjonsrapport for de tre første iterasjonene er skrevet.
- Research for TCP/IP og bildebehandling er gjennomført, så alle gruppemedlemmer har en viss oversikt over hva som skal gjøres når neste iterasjon starter.

Utfordringer og tiltak

Utdypende iterasjon #2 har vært en kort iterasjon der siste del av planlegging og tilrettelegging av dokumentasjon har vært fokus. Iterasjonen har gått bra og det har vært få utfordringer.

Nå som iterasjonen er ferdig har vi endelig fått ferdig planleggingen og er klare til å starte på konstruksjonsfasen. Vi hadde litt problemer med bookinga til andre presentasjon. Sensoren vår kunne ikke dagen vi hadde satt opp. Derfor måtte vi bytte til et mye tidligere tidspunkt. Dette vil påvirke hvor mye vi får vist frem på andre presentasjonen, men vi planlegger å sette av ekstra mye tid og få ferdig designdokument og et lite eksempel på hva oppgaven går ut på.
Vi ble ferdig med et førsteutkast av designdokumentet, og kan derfor i teorien begynne å programmere. Siden presentasjon 2 ble så mye tidligere enn forventet har vi valgt å fokusere dokumentasjon og planlegging i starten av neste iterasjon. Dette vil påvirke tida vi kommer til å ha på programmering, men vil ikke ha noe stor påvirkning på sluttresultatet. Planleggingen ville foregått mens vi programmerer, så om vi gjør det før eller mens vi programmerer vil ikke ha så stor betydning.

Konklusjon

Vi kan konkludere med at utdypende iterasjon #2 har gått som forventet. Prosjektplan, krav- og testspesifikasjon er ferdig, og software designdokumentet er et godt utgangspunkt å jobbe fra. Starter neste iterasjon med siste oppdatering av planleggingsdokumentasjon som skal være ferdig til andre presentasjon.

3.4 Konstruksjonsiterasjon #1

Periode: 17.03.14 – 16.04.14

3.4.1 Iterasjonsplan konstruksjonsiterasjon #1

Mål

I første iterasjon for konstruksjonsfasen har vi som mål å få satt opp følgende:

- Presentasjon 2
- Rammeverk for software
- TCP/IP kommunikasjon mellom hardware
- Bildeuthenting med dataekstraksjon
- Lage noen enkle scripts til industriroboten

Midt i denne iterasjonen har vi vår andre presentasjon. Dette er en olbigatorisk presentasjon som vil påvirke sluttkarakter. Derfor ønskervi å få på plass noen av målene vi har satt ovenfor, slik at vi får vist frem noe mer enn bare teori.

Risikoanalyse

I starten av denne fasen er det dokumentasjon som skal være ferdig til andre presentasjon som har hovedfokus. Det vil si, prosjektplan, kravspesifikasjon, testspesifikasjon, iterasjonsdokument og software designdokument. Software Designdokumentet må ha en første versjon klar til andre presentasjon og det er kritisk at denne er klar. Den bygger mye på andre presentasjon og videre hvor lett det blir for oss å programmere. Hvis ikke denne er ferdig vil vi ha en uklar presentasjon, der vi ikke klarer å få fram hvordan vi har planlagt oppbygningen av systemet.
Senere i iterasjonen er det programmering som skal gjøres. Bildegjenkjenning og kommunikasjon med parsing er hovedprioritet og må ha et fungerende utkast til neste iterasjon. Det er ikke så mye igjen av semesteret og derfor er det viktig at vi legger mye tid i å få til denne programmeringen så det er klart til iterasjon #2.

RISIKOFAKTORER	ÅRSAK	S	K	R	RISKOREDUSERENDE TILTAK
Tap av data	Feil på hardware og software.	1	5	5	Vi reduserer denne risikoen ved å ha siste versjon lagret på minst en minnebrikke, vi har et dokumenthåndteringssystem der vi regelmessig tar sikkerhetskopi av dokumenter og filer.
Sykdom eller fravær	N/A.	3	2	6	Ingen tiltak nødvendig under normale omstendigheter.
Feilestimering av tidsbruk	Ikke god nok planlegging.	2	3	6	Bruke nok tid til planlegging, og ta med potensielle tidskrevende utfordringer i planleggingen.
Skade på komponenter	Miskalkulert robotscript resulterer i skade på DCP.	2	4	8	Scripte robotbevegelsene i små steg, og alltid være beredt på å trykke «avbryt» knappen.
Dysfunksjonell hardware	Hardware slår seg vrang.	2	3	6	Sørge for å behandle hardware på en god og forsvarlig måte. Ikke tukle med knapper/kabler/bevegelige deler unødig.
Software utkast ferdig	Ikke bli ferdig før neste iterasjon	3	4	12	Det må legges ekstra timer inn i denne etter andre presentasjon. Det er tiden vi har igjen i iterasjonen som er farlig, og derfor reduserer vi denne risikoen hvis vi jobber fler timer enn planlagt.

Tabell 12: Risikoanalyse for konstruksjonsiterasjon #1

Aktiviteter

AKTIVITET	ANSVARLIG	FRIST	RESSURSER	RESULTAT	ID
Iterasjonsdokument	Ståle Rudin	14.03.14	AKS, SR	Itr.plan og rapport	1500
Kommunikasjon	Ståle Rudin	04.04.14	HBS, ELR	Kommunikasjonsvei mellom hardware	4020
Robotprogrammering	Henrik Berge Sørum	04.04.14	SR, HBS	Bevegelsesmønstre til robot	4200
Verifikasjon og sammenligning av resultater	Eirik Lien Roa	04.04.14	ELR, HBS	Uthenting av bilde, og ekstraktere data	4040
GUI	Ståle Rudin	21.03.14	SR, AKS	GUI	4010
Presentasjon 2	Eirik Lien Roa	26.03.14	Alle	Presentasjon 2	2100
Software designdokument	August Kind Svendsen	26.03.14	AKS, HBS	Software designdokument	3200

Tabell 13: Aktiviteter fra konstruksjonsiterasjon #1

Aktiviteter detaljert

Kommunikasjon:
En viktig ting å få på plass i denne iterasjonen er TCP/IP kommunikasjonsveien med en parser. Dette legger grunnlaget for all datatrafikk mellom hardware. Vi har allerede en tilkobling til våpenstasjonen, men for å kunne sende beskjeder trenger vi en parser. På grunn av hvor grunnleggende dette er for systemet kommer vi til å legge fokus på dette i denne iterasjonen.

Ingen på gruppa har erfaring med parsing og trenger derfor å lese seg opp på dette. Kort sagt er dette et program, vanligvis en del av en kompilator, som deler data inn i mindre elementer for enkel oversetting til et annet språk.

Robotprogrammering:
Begynne å programmere bevegelsene til roboten slik at den kan trykke på de forskjellige knappene på DCP-en, deretter sette dette sammen til en sekvens som utgjør en testsekvens. Dette har en på gruppa erfaring med fra tidligere arbeid, noe som resulterer i at TCP/IP med parsing vil bli prioritert høyere fordi vi forventer at dette vil bli vanskeligere.

Verifikasjon:
På verifikasjonsbiten har vi allerede klart å hente ut bilder fra DCP-en. Det som gjelder nå er å kunne lage en algoritme som ser på bildet og henter ut den relevante informasjonen, ofte består dette av tekst. Denne teksten skal gjøres om til en string som kan benyttes andre steder i systemet. Dette er en viktig del å få på plass, men som ikke trenger å være ferdig like tidlig som nettverkskommunikasjonen og vil vike dersom vi trenger fler ressurser på den biten.

GUI:
Vi forventer å få et førsteutkast på hvordan GUI-en kommer til å se ut, som vi kan bruke til å integrere de andre delene som blir produsert i denne iterasjonen. Dette er en både enkel og lite tidkrevende oppgave som vi kommer til å gjøre etter det mer viktige er på plass.

Software designdokument:
Første uka i denne iterasjonen så vil vi ha ekstra fokus på å få ferdig en førsteversjon av software designdokumentet. På grunn av dette må noen av de andre aktivitetene vike i denne perioden, men vi føler det er viktigere å få med dette dokumentet på presentasjonen, og det vil gjøre arbeidet med de andre aktivitetene enklere senere i iterasjonen.

Tillegg- Endring av tidsforbruk i denne iterasjonen

På grunn av eksamen i annet fag, der halve pensum ble utdelt en uke før eksamensdato, har vi måtte gjøre endringer på lengen av iterasjonen. Vi har blitt nødt til å forlenge denne iterasjonen med en og en halv uke (16.04.14). Dette forårsaker at vi må forkorte noe annet og i dette tilfelle har vi valgt å korte ned på den ene konkluderende fasen. Dette er på grunn av at arbeidet i denne iterasjonen er viktigere enn det som skal gjøres i konklusjonsfasen, med tanke på sluttprodukt.

3.4.2 Iterasjonsrapport

Aktiviteter fra iterasjonsplan

AKTIVITET	ANSVARLIG	FRIST	RESSURSER	RESULTAT	ID
Iterasjonsdokument	Ståle Rudin	14.03.14	AKS, SR	Itr.plan og rapport	1500
Kommunikasjon	Ståle Rudin	04.04.14	HBS, ELR	Kommunikasjonsvei mellom hardware	4020
Robotprogrammering	Henrik Berge Sørum	04.04.14	SR, HBS	Bevegelsesmønstre til robot	4200
Verifikasjon og sammenligning av resultater	Eirik Lien Roa	04.04.14	ELR, HBS	Uthenting av bilde, og ekstraktere data	4040
GUI	Ståle Rudin	21.03.14	SR, AKS	GUI	4010
Presentasjon 2	Eirik Lien Roa	26.03.14	Alle	Presentasjon 2	2100
Software designdokument	August Kind Svendsen	26.03.14	AKS, HBS	Software designdokument	3200

Tabell 14: Aktiviteter fra konstruksjonsiterasjon #1

Overholdelse

Iterasjonsdokument:
Vi har fått laget en iterasjonsplan for neste iterasjon, konstruksjonsiterasjon #2.

Kommunikasjon:
TCP/IP kommunikasjon mellom software, våpenstasjon og robot er oppe. Vi har også et fungerende utkast for parseren. Vi blir nødt til å se litt mer på denne i neste iterasjon.

Robotprogrammering:
Robotprogrammeringa har gått helt som planlagt. Programmet som ligger inne på datamaskinen som styrer roboten gjør det veldig enkelt å lage egne scripts og bevegelser. Vi kan også enkelt sette opp beskjeder som robotdatamaskinen kan ta inn for å sette i gang et script.

Verifikasjon:
På verifikasjonsbiten har vi fått til å hente ut et skjermbilde fra DCP-en, som vi deretter kan sjekke opp mot et annet bilde for å finne ut hvor like bildene er. Her blir piksel for piksel sammenlignet ved hjelp av en avansert algoritme, som vil tilslutt fortelle oss likheten i prosent. Dette kan vi bruke for å verifisere resultater underveis i testingen, men vi ønsker også å kunne få ekstraktere selve teksten fra bildet, om dette viser seg å være oppnåelig.

Presentasjon 2:
Etter litt trøbbel med å finne et riktig tidspunkt for denne presentasjonen, fant vi tilslutt noe som passet for alle. Det endte med å bli litt tidligere enn forventet, men dette hadde lite til ingen betydning for hvordan presentasjonen gikk, vi måtte bare omplassere litt på planene. Presentasjonen endte opp som en god presentasjon, der både vi og sensorer/veiledere ble fornøyd med resultatet.

GUI:
Vi har nå et fungerende utkast av GUI. GUI vil mest sansynlig forandre seg ettersom vi videreutvikler programmet vårt. Dette er arbeid som krever lite tid og ressurser, og vil bli satt mer fokus på når viktigere arbeid er unnagjort.

Software design:
Vi har hatt stort fokus på å få på plass et godt software design dokument denne iterasjonen. På grunn av dette har enkelte andre aktiviteter (parsing og verifikasjon) mistet litt tid, men det er på grunn av software designdokumentet legger grunnlaget for hvordan komponentene som jobbe sammen, noe vi føler er viktigere å få klargjort slik at alle arbeider med den samme visjonen.

Utfordringer og tiltak

På grunn av en eksamen som ble utvidet med ekstra pensum på kort varsel, følte gruppa at vi måtte kutte av litt tid fra hovedprosjektet for å kunne øve på denne eksamen. Vi ble derfor litt hengende etter den originale planen, men hentet inn dette ved å utvide denne iterasjonen med 1 uke og jobbe litt ekstra i påskeferien.

Dette forårsaker at vi må forkorte noe annet og i dette tilfelle har vi valgt å korte ned på den ene konkluderende fasen. Dette er på grunn av at arbeidet i denne iterasjonen er viktigere enn det som skal gjøres i konklusjonsfasen.

Konklusjon

Denne iterasjonen har bydd på enkelte utfordringer som vi har taklet så godt vi kan. Vi har de viktigste komponentene oppe og går, der noen trenger litt ekstra arbeid for å fungere slik vi ønsker. Nå som det andre faget er over blir det lettere å fokusere fult og helt på dette prosjektet fremover, og håper at dette vil hjelpe oss å komme i mål.

3.5 Konstruksjonsiterasjon #2

Periode: 22.04.14 – 02.05.14

3.5.1 Iterasjonsplan

Mål

Nå er vi allerede kommet godt i gang med implementasjonen av de forskjellige delene til systemet. Målet for denne iterasjonen er å fortsette med implementasjonen og starte på integreringen av alle delene i systemet. Delene vi skal fokusere mest på i denne iterasjonen er verefisering, kontrollerklassen, lagring av rapport og robotscripting.

I tillegg skal vi starte på et teknologidokument. Dette dokumentet skal inneholde fakta om de forskjellige delene til systemet. Målet vårt er å skrive om hver del mens vi holder på med den, slik at det skal være lettere å skrive om de forskjellige emnene, og at det vil være mindre risiko for at vi glemmer noe.

Risikoanalyse

I denne iterasjonen er det medium risiko, for det er her vi setter opp grunnmuren til programmet. Hvis vi ikke har gjort god nok research på at delene i systemet vil fungere når de skal samarbeide må vi programmere om store deler av programmet. For å unngå denne risikoen har vi hatt noen møter med de ansatte på KPS anngående de valgene vi har tatt eller hva de anbefaler.

RISIKOFAKTORER	ÅRSAK	S	K	R	RISKOREDUSERENDE TILTAK
Tap av data	Feil på hardware og software.	1	5	5	Vi reduserer denne risikoen ved å ha siste versjon lagret på minst en minnebrikke, vi har et dokumenthåndteringssystem der vi regelmessig tar sikkerhetskopi av dokumenter og filer.
Sykdom eller fravær	N/A.	3	2	6	Ingen tiltak nødvendig under normale omstendigheter.
Feilestimering av tidsbruk	Ikke god nok planlegging	2	3	6	Bruke nok tid til planlegging, og ta med potensielle tidskrevende utfordringer i planleggingen.

Tabell 15: Risikoanalyse for konstruksjonsiterasjon #2

3.5.1.1 Aktiviteter

AKTIVITET	ANSVARLIG	FRIST	RESSURSER	RESULTAT	ID
Iterasjonsdokument	Ståle Rudin	02.05.14	SR, AKS	Itr.plan og rapport	1500
Verifikasjon og sammenligning av resultater	Eirik Lien Roa	02.05.14	ELR, HBS	Hente ut informasjon fra WS	4040
Bildebehandling	Eirik Lien Roa	02.05.14	ELR	Sammenligne to bilder	4030
Robotprogrammering	Henrik Berge Sørum	02.05.14	HBS	Bevegelsesmønstre til robot	4200
Kontroller-klassen	Ståle Rudin	02.05.14	SR, AKS	Hjernen til systemet	4070
Resultatdatabase med rapport	Henrik Berge Sørum	02.05.14	AKS, ELR	Lagre rapport i egen fil	4050
Teknologidokument	August Kind Svendsen	02.05.14	SR, AKS, HBS, ELR	Fakta om de forskjellige delene	

Tabell 16: Aktiviteter fra konstruksjonsiterasjon #2

Aktiviteter detaljert

Verifikasjon:
Vi skal hente ut informasjon fra våpenstasjonen og sammenligne disse dataene opp mot de riktige dataene som er lagret på forhånd.

Bildebehandling:
Her skal vi hente ut bilde fra våpenstasjonen og sammenligne dette med et bilde som allerede er lagret på pc-en.

Robotprogrammering:
Alle robotbevelgelsene vi trenger skal være ferdig i denne iterasjonen.

Kontroller-klassen:
Denne klassen skal kunne ta inn et valg om hvilken test det er fra GUI klassen. Deretter skal den kunne gå gjennom testen sekvens for sekvens helt til den er ferdig. Denne klassen skal også kunne håndtere flere tråder med hjelp av mutex.

Testrapport:
Når en test blir kjørt skal alt loggføres i en fil som blir lagt på pc-en. Her skal det stå hvilke resultater vi får ut, med bilder og tekst. I tillegg skal det stå hvilke komandoer som ble kjørt med datostempling på når de ble kjørt.

Teknologidokument:
I dette dokumentet skal det stå hva slags teknologi eller hvilke metoder vi har brukt for å fullføre de forskjellige delene til systemet.

3.5.2 Iterasjonsrapport

3.5.2.1 Aktiviteter fra iterasjonsplan

AKTIVITET	ANSVARLIG	FRIST	RESSURSER	RESULTAT	ID
Iterasjonsdokument	Ståle Rudin	02.05.14	SR, AKS	Itr.plan og rapport	1500
Verifikasjon og sammenligning av resultater	Eirik Lien Roa	02.05.14	ELR, HBS	Hente ut informasjon fra WS	4040
Bildebehandling	Eirik Lien Roa	02.05.14	ELR	Sammenligne to bilder	4030
Robotprogrammering	Henrik Berge Sørum	02.05.14	HBS	Bevegelsesmønstre til robot	4200
Kontroller-klassen	Ståle Rudin	02.05.14	SR, AKS	Hjernen til systemet	4070
Resultatdatabase med rapport	Henrik Berge Sørum	02.05.14	AKS, ELR	Lagre rapport i egen fil	4050
Teknologidokument	August Kind Svendsen	02.05.14	SR, AKS, HBS, ELR	Fakta om de forskjellige delene	

Tabell 17: Aktiviteter fra konstruksjonsiterasjon #2

Overholdelse

Nå begynner rammeverket å ta form. Vi er ferdig med et førsteutkast av alle delene til systemet og startet å integrere disse inn i et stort program. Det som ikke er ferdig integrert er verifikasjonen. Dette er fordi det er er en stor del av programmet vårt, og tok lenger tid enn forventet. Vi ville ha et fungerende utkast av det før vi integrerte det inn i programmet. Dette er noe vi fikk til i slutten av denne iterasjonen så integreringen av verifikasjonen vil skje i starten av neste iterasjon. Teknologidokumentet er også i gang. Her har vi startet på noen av dokumentene. Det er ikke alle som kan startes på før vi vet mer om hvordan det endelige programmet blir. Vi valgte å dele dette dokumentet opp i flere mindre dokumenter.

Her er en liste over hva som er gjort i de forskjellige delene til systemet:

- XML avlesning: Nå kan man laste inn et XML dokument som programmet leser av og lagrer de forskjellige sekvensene og kommandoene i et array. Dette er for at programmet skal vite hvilke metoder den skal bruke.
- Kontroller-klassen: Denne klassen henter arrayet som blir hentet ut ifra XML-dokumentet og går sekvensielt gjennom disse, og utfører de riktige metodene til hvilke kommando som kjører.
- Logg: For hver kommando som blir kjørt blir det lagret en beskrivende linje med datostempling på hva som har skjedd i en tekstfil (notisblokk).
- Robotprogrammering: Nå er alle de forskjellige bevegelsene vi trenger ferdige.
- Bildebehandling: Henter ut bilde fra DCP-en og sammenligner det opp mot en fil på pc-en og skjekker om alle pixlene er like.

Utfordringer og tiltak

Den største utfordringen i denne iterasjonen har vært tida. Programmering krever mye testing og feiling. Dette er noe vi ikke har tid til slik at vi måtte nedprioritere diverse ekstrafunksjoner.

Et problem når vi skal sammenligne to bilder fant vi ut at det var en liten bit av bildet som kan variere for hver gang vi tar et bilde av DCP-en. Dette løste vi ved å klippe ut den biten av bildet vi trenger å sammenligne. På denne måten vil bilde kun bestå av det vi trenger.

Konklusjon

Denne iterasjonen har gikk nesten som forventet. Vi har et godt utgangspunkt når vi går over til den siste iterasjonen i konstruksjonsiterasjonen. Vi har satt opp hele systemet slik at det er lett å sette inn verifikasjonen når den tid kommer.

3.6 Konstruksjon iterasjon #3

Periode: 05.05.14 – 16.05.14

3.6.1 Iterasjonsplan

Mål

Nå er vi ferdige med det meste av implementasjonen av de forskjellige delene, med unntak av verifiseringen. Dette kommer til å bli hovedfokuset i denne perioden. I tillegg kommer vi til å starte og teste mye, for å se om vi lykkes med å utføre tester automatisk, med verifisering av resultater.

Når vi har fått inn verifikasjonen vil vi se på forskjellige måter å håndtere feilsituasjoner i systemet. Vi vil her sette opp en egen klasse som skal kunne detektere uønskede feil under test, da med fokus av fatale feil, som for eksempel hvis våpenstasjonen fyrer eller beveger seg når den ikke skal.

Vi skal også kunne skrive ut en testrapport automatisk etter at testen er ferdig med resultater fra kjøringen og verifikasjonen. Dette skal vi gjøre via en database og rapport verktøy som er innebygd i visual studio.

I denne iterasjonen er målet å ha en ferdig software integrert og samtidig gjennomført en del tester.

Vi vil også ha en del fokus på dokumentasjon av de forskjellige delene av systemet vårt i form av teknologi dokumenter og rapporter.

Risikoanalyse

I denne iterasjonen er det høy risiko å få integrert verifikasjonen ettersom dette er det KPS legger størst vekt på i systemet vårt. Dette skal vi løse ved å sette av nok tid og ressurser og gjøre dette først i iterasjonen. En risiko som begynner å bli stor nå er feilhåndteringen til systemet. De andre komponentene av systemer må fungere godt før vi kan lage en god feilhåndtering. Dette blir da noe vi håper å rekke hvis vi klarer å få et fungerende system i løpet av iterasjonen.

RISIKOFAKTORER	ÅRSAK	S	K	R	RISKOREDUSERENDE TILTAK
Tap av data	Feil på hardware og software.	1	5	5	Vi reduserer denne risikoen ved å ha siste versjon lagret på minst en minnebrikke, vi har et dokumenthåndteringssystem der vi regelmessig tar sikkerhetskopi av dokumenter og filer.
Sykdom eller fravær	N/A.	3	2	6	Ingen tiltak nødvendig under normale omstendigheter.
Feilestimering av tidsbruk	Ikke god nok planlegging	2	3	6	Bruke nok tid til planlegging, og ta med potensielle tidskrevende utfordringer i planleggingen.
Verifikasjonen ikke blir ferdig	Mer omfattende en planlagt	3	4	12	Jobbe flere timer en planlagt denne uken slik at risikoen for at vi ikke rekker å bli ferdig reduseres
Feilhåndtering ikke blir ferdig	Systemet mangler funksjonalitet.	4	4	16	Jobbe flere timer med systemet slik at risikoen for at vi ikke rekker å starte på feilhåndtering reduseres.

Tabell 18: Risikoanalyse for konstruksjonsiterasjon #3

3.6.1.1 Aktiviteter

AKTIVITET	ANSVARLIG	FRIST	RESSURSER	RESULTAT	ID
Iterasjonsdokument	Ståle Rudin	16.05.14	SR	Itr.plan og rapport	1500
Verifikasjon og sammenligning av resultater	Eirik Lien Roa	16.05.14	ELR	Hente ut informasjon fra WS, og verifisere dette. Integrere dette inn i software	4040
Feilhåndtering	Eirik Lien Roa	16.05.14	ELR	Klare å detektere feil under kjøring	4090
Robotprogrammering	Henrik Berge Sørum	16.05.14	HBS	Bevegelsesmønstre til robot	4200
Integrasjon	Ståle Rudin	16.05.14	SR	Integrere de ulike delene sammen, og forbedringer i systemet.	4300
Resultatdatabase med rapport	Henrik Berge Sørum	16.05.14	HBS, SR	Lagre rapport i egen fil	4050
Teknologidokument	August Kind Svendsen	16.05.14	SR, AKS, HBS, ELR	Fakta om de forskjellige delene	
Software design dokument	August Kind Svendsen	16.05.14	AKS	Design av system	3200
Test av software	Ståle Rudin	16.05.14	ELR, SR	Teste systemet.	5000

Tabell 19: Aktiviteter fra konstruksjonsiterasjon #3

Aktiviteter detaljert

Verifikasjon:
Vi skal hente ut informasjon fra våpenstasjonen og sammenligne disse dataene opp mot de riktige dataene som er lagret på forhånd. I tillegg vil vi verifisere at bevegelser og tilstander er riktige under kjøring.

Feilhåndteringer i software:
Her skal vi programmere en klasse som skal håndtere feilsituasjoner på våpenstasjonen. Fokuset her blir å sjekke fyring, fatale feil, og uønsket bevegelse.

Robotprogrammering:
Alle robotbevelgelsene vi trenger skal være ferdig i denne iterasjonen. I tilegg til justeringer underveis.

Integrasjon/system:
Her skal vi integrere de ulike delene av systemet, se om dette fungerer og komme med justeringer som får dette til å fungere. I tillegg vil vi se på løsninger som kan gjøre systemet bedre.

Testrapport:
Når en test kjører skal en testrapport bli automatisk generert av systemet vårt og vist når testen er ferdig kjørt. Den skal inneholde informasjon om hver sekvens blir godkjent eller ikke.

Teknologidokument:
I dette dokumentet skal det stå hva slags teknologi eller hvilke metoder vi har brukt for å fullføre de forskjellige delene til systemet.

Software design dokument:
Her skal vi oppdatere software design dokumentet til å bli lik som det endelige resultatet av softwaren vår.

Testing:
Her skal de ulike delene testes og verifiseres at de opprettholder krav og fungerer.

3.6.2 Iterasjonsrapport

3.6.2.1 Aktiviteter fra iterasjonsplan

AKTIVITET	ANSVARLIG	FRIST	RESSURSER	RESULTAT	ID
Iterasjonsdokument	Ståle Rudin	16.05.14	SR	Itr.plan og rapport	1500
Verifikasjon og sammenligning av resultater	Eirik Lien Roa	16.05.14	ELR	Hente ut informasjon fra WS, og verifisere dette. Integrere dette inn i software	4040
Feilhåndtering	Eirik Lien Roa	16.05.14	ELR	Klare å detektere feil under kjøring	4090
Robotprogrammering	Henrik Berge Sørum	16.05.14	HBS	Bevegelsesmønstre til robot	4200
Integrasjon	Ståle Rudin	16.05.14	SR	Integrere de ulike delene sammen, og forbedringer i systemet.	4300
Resultatdatabase med rapport	Henrik Berge Sørum	16.05.14	HBS, SR	Lagre rapport i egen fil	4050
Teknologidokument	August Kind Svendsen	16.05.14	SR, AKS, HBS, ELR	Fakta om de forskjellige delene	
Software design dokument	August Kind Svendsen	16.05.14	AKS	Design av system	3200
Test av software	Ståle Rudin	16.05.14	ELR, SR	Teste systemet.	5000

Tabell 20: Aktiviteter fra konstruksjonsiterasjon #3

Overholdelse

Verifikasjon og sammenligning av resultater:
Vi har nå integrert all verifikasjon vi har lagd.

Feilhåndtering:
Feilhåndteringen ble ferdig implementert og vi trodde vi hadde noe som fungerte. Ved nærmene testing viste det seg at denne funksjonaliteten ikke fungerte som forventet. Det er på god vei, men mangler fungerer ikke per dags dato.

Robotprogrammering:
Vi lagde nye «fingre» som vi bruker til å trykke på knapper eller brytere. Dette gjorde vi fordi det vil være mindre sannsynlighet for at «fingrene» beveger på seg når vi bruker dem til å trykke på CG. Når vi gjorde dette måtte vi også endre på bevegelsene slik at de stemmer overens med de nye «fingrene».

Integrasjon:
Nå er alt integrert og klar for testing.

Resultatdatabase med rapport:
Vi brukte en lokal microsoft SQL server som vi opprettet en database i. Denne databasen blir koblet automatisk opp mot programmet så lenge den er innstallert på riktig måte. Når vi starter en test tømmes denne databasen slik at vi kan skrive ny informasjon til den. Når en sekvens i en test er fullført blir det fylt inn en ny rad i en tabell som ligger i databasen med id på hvilken sekvens det er og om den ble godkjent eller ikke godkjent. Når testen er fullført vises det et nytt vindu med en illustrasjon på hvordan rapporten ble og blir automatisk lagret som PDF på pc-en.

Teknologidokument:
Vi valgte å dele opp teknologidokumentet opp i flere deler.
Software: Denne skal inneholde hvordan selve programmet fungerer.
XML: Hva vi bruker XML til og hvordan vi bruker det.
Verifikasjon: Hva slags verifikasjon vi har og hvordan vi har løst det.
Robot: Hva slags robot vi har brukt og hvordan vi har brukt den.

Software designdokument:
Programmet vårt har forandret seg betraktelig i forhold til hvordan vi planla det skulle være. Derfor må vi oppdatere system designdokumentet til hvordan programmet er nå.

Test av software:
Vi har fullført en rekke tester av systemet slik at vi fikk rettet opp i de feilene vi fant.

Utfordringer og tiltak

Denne iterasjonen har gått som forventet med et par unntak. Programmet trengte en hel del testenr som tok lenger tid enn forventet slik at det ble mindre tid til dokumentasjon. Men vi valgte å bruke ressursene som var nødvendig for å få et fungerende program slik at vi bare trenger å tenke på dokumentasjon og testing av krav i neste iterasjon. Det vil også være lettere å skrive dokumentasjonen når programmet er slik vi ønsker at det skal være.

Feilhåndteringen er en stor risiko den siste iterasjonen. Siden denne ikke ble ferdig må vi se mer på denne i neste iterasjon og forhåpentligvis er denne på plass før prosjektet er ferdig.

Konklusjon

Vi føler at vi har laget et godt rammeverk av systemet. Det som mangler er feilhåndtering, testing av krav og resten av dokumentasjonen før prosjektet skal leveres. Dette prosjektet har vært for å lage et godt rammeverk som kan videreutvikles og forhåpentligvis brukes av KPS og dette er noe vi føler vi har fått til.

3.7 Konkluderende iterasjon

Periode: 19.05.14 – 23.05.14

3.7.1 Iterasjonsplan

Mål

Dette er den siste iterasjonen hvor målet er for det meste dokumentasjon, se mer på feilhåndtering og testing av krav. Når dette testes vil vi finpusse funksjoner og metoder sammtidig. Her er en liste over hvilke dokumenter som skal være ferdig innen denne iterasjonen er ferdig:

- Prosjektplan
- Kravspesifikasjon
- Testspesifikasjon
- Testrapport
- Etteranalyse
- Iterasjonsdokument
- Komponentdokument
- Software designdokument
- Fremtidsdokument
- Sluttrapport
- Teknologidokument – Software, XML, robot og Verifikasjon.

De fleste av disse dokumentene er nesten ferdige, men vi må oppdatere en del av disse eller legge til ny relevant informasjon. Når det kommer til programmet mangler vi feilhåndtering og finpuss, men denne finpussen er viktig når vi skal teste systemet slik at vi får oppfylt de kravene vi har mulighet til å oppfylle. Dette vil også si at vi skal sette programmet på prøve og gjøre alle testene vi har satt opp for å få kravene godkjent.

Risikoanalyse

Dette er den iterasjonen med høyest risiko. Her er det viktig at alt er på plass. Hvis noe skulle gå galt slik at vi ikke blir ferdig med en aktivitet vil det påvirke hva vi får levert inn. Så her er det viktig å være produktiv og fokusert. For å redusere risikoen på at noe skal gå galt har vi satt en dato et par dager før vi skal levere hvor programmet og all testingen skal være ferdig. Slik at vi har tid til å gå over dokumentasjonen vår og se over feil eller legge til ny informasjon.

RISIKOFAKTORER	ÅRSAK	S	K	R	RISKOREDUSERENDE TILTAK
Tap av data	Feil på hardware og software.	1	5	5	Vi reduserer denne risikoen ved å ha siste versjon lagret på minst en minnebrikke, vi har et dokumenthåndteringssystem der vi regelmessig tar sikkerhetskopi av dokumenter og filer.
Sykdom eller fravær	N/A.	3	2	6	Ingen tiltak nødvendig under normale omstendigheter.
Feilestimering av tidsbruk	Ikke god nok planlegging	2	3	6	Bruke nok tid til planlegging, og ta med potensielle tidskrevende utfordringer i planleggingen.
Krav ikke godkjennes	System som ikke klarer å teste kravene eller andre bugs som medfører at krav ikke kan testes.	3	4	12	Starte iterasjonen med testing av krav, oppdatere programmet og fikse bugs hvis vi finner dette underveis.
Feilhåndtering ikke blir ferdig	Ikke nok tid til å få ferdig feilhåndtering.	4	4	16	Teste alle krav som ikke innebærer feilhåndtering tidlig så det kan vi kan se om vi rekker å ordne feilhåndteringen mot slutten av uken.

Tabell 21: Risikoanalyse for konkluderende iterasjon #1

3.7.1.1 Aktiviteter

AKTIVITET	ANSVARLIG	FRIST	RESSURSER	RESULTAT	ID
Iterasjonsdokument	Ståle Rudin	25.05.14	SR	Bli ferdig med dokumentet	1500
Prosjektplan	Henrik Berge Sørum	25.05.14	HBS, SR	Bli ferdig med dokumentet	1200
Kravspesifikasjon	August Kind Svendsen	25.05.14	AKS	Bli ferdig med dokumentet	3000
Testspesifikasjon	Henrik Berge Sørum	25.05.14	HBS	Bli ferdig med dokumentet	5100
Testrapport	Henrik Berge Sørum	25.05.14	HBS, SR, ELR, AKS	Bli ferdig med dokumentet	5200
Etteranalyse	August Kind Svendsen	25.05.14	AKS	Bli ferdig med dokumentet	6100
Komponentdokument	August Kind Svendsen	25.05.14	AKS	Bli ferdig med dokumentet	3700
Software designdokuent	August Kind Svendsen	25.05.14	AKS	Bli ferdig med dokumentet	3200
Tek. Dok – Software	Ståle Rudin	25.05.14	SR	Bli ferdig med dokumentet	3300
Tek. Dok – XML	August Kind Svendsen	25.05.14	AKS	Bli ferdig med dokumentet	3400
Tek. Dok – Robot	Henrik Berge Sørum	25.05.14	HBS	Bli ferdig med dokumentet	3500
Tek. Dok – Verifikasjon	Eirik Lien Roa	25.05.14	ELR	Bli ferdig med dokumentet	3600
Sluttrapport	Eirik Lien Roa	25.05.14	ELR	Bli ferdig med dokumentet	6000
Fremtidsdokument	August Kind Svendsen	25.05.14	AKS	Bli ferdig med dokumentet	3800
Feilhåndtering	ELR	25.05.14	ELR, SR	Bli ferdig med feilhåndtering	4090

Tabell 22: Aktiviteter fra konkluderende iterasjon #1

Aktiviteter detaljert

Dokumentasjon:
All dokumentasjon skal skrives ferdig.

Testrapport:
Her skal vi gå gjennom alle testene som skal til for å få kravene godkjent. Om kravene blir godkjent eller ikke skal dokumenteres i testrapporten med en begrunnelse på hvorfor den ble godkjent eller ikke.

Feilhåndtering:
Feilhåndteringen er noe som burde komme på plass. Vi har noe som vi er ganske sikre på er i nærheten av å være ferdig, og derfor satser vi på at dette kan bli gjort før prosjektet skal leveres. Siden vi skal lage et rammeverk til automatisering av funksjonstester, så er det ingen katastrofe om dette ikke fungerer helt optimalt. KPS vil ikke bruke systemet før den er videreutviklet og derfor er det ingen krise om feilhåndteringen ikke blir helt ferdig.

3.7.2 Iterasjonsrapport

AKTIVITET	ANSVARLIG	FRIST	RESSURSER	RESULTAT	ID
Iterasjonsdokument	Ståle Rudin	25.05.14	SR	Bli ferdig med dokumentet	1500
Prosjektplan	Henrik Berge Sørum	25.05.14	HBS, SR	Bli ferdig med dokumentet	1200
Kravspesifikasjon	August Kind Svendsen	25.05.14	AKS	Bli ferdig med dokumentet	3000
Testspesifikasjon	Henrik Berge Sørum	25.05.14	HBS	Bli ferdig med dokumentet	5100
Testrapport	Henrik Berge Sørum	25.05.14	HBS, SR, ELR, AKS	Bli ferdig med dokumentet	5200
Etteranalyse	August Kind Svendsen	25.05.14	AKS	Bli ferdig med dokumentet	6100
Komponentdokument	August Kind Svendsen	25.05.14	AKS	Bli ferdig med dokumentet	3700
Software designdokuent	August Kind Svendsen	25.05.14	AKS	Bli ferdig med dokumentet	3200
Tek. Dok – Software	Ståle Rudin	25.05.14	SR	Bli ferdig med dokumentet	3300
Tek. Dok – XML	August Kind Svendsen	25.05.14	AKS	Bli ferdig med dokumentet	3400
Tek. Dok – Robot	Henrik Berge Sørum	25.05.14	HBS	Bli ferdig med dokumentet	3500
Tek. Dok – Verifikasjon	Eirik Lien Roa	25.05.14	ELR	Bli ferdig med dokumentet	3600
Sluttrapport	Eirik Lien Roa	25.05.14	ELR	Bli ferdig med dokumentet	6000
Fremtidsdokument	August Kind Svendsen	25.05.14	AKS	Bli ferdig med dokumentet	3800
Feilhåndtering	ELR	25.05.14	ELR, SR	Bli ferdig med feilhåndtering	4090

Tabell 23: Aktiviteter fra konkluderende iterasjon #1

Overholdelse

Dokumenter:

Alle dokumentene er ferdig og gjort klart til innlevering. Vi har også ryddet opp i dropbox mappen og brent ut nødvendige CDer til innlevering.

Testrapport:

Kravene er testet og rapporten er ferdig. Det er tre krav som feilet, dette er rammekrav 2 som er krav om feilhåndtering, sikkerhetskrav 1 som er krav om feilhåndtering og funksjonskrav 3 som er krav om eksternt måleutstyr.

Utfordringer og tiltak

Denne iterasjonen har vært utfordrende. Testing av krav medførte en del mer finpussing av programmet en forventet så det ble dårligere tid til alt annet. Kravene måtte testes ferdig denne uken og derfor var det fullt fokus på å finpusse ferdig programmet i starten av uken så alle krav kunne testes. Det er tre av kravene som feilet testingen. To av disse er feilhåndtering som er

vanskelig å få på plass før de andre komponentene av systemet fungerer godt. Derfor har det ikke vært mulig å utvikle dette grundig nok. Det siste kravet som feilet er det eksterne måleutstyret, dette er et B krav og har derfor ingen innvirkning på prosjektet. Dette er en ekstra funksjonalitet som systemet burde ha hvis det skal brukes til testing, noe som gjør verifikasjonen til systemet sikrere, men er ingen must for at systemet skal verifisere på en god måte.
Dokumentasjonen har gått greit for seg, det har blitt mye skriving de siste dagene og det ble en full helg med jobbing, men dette var noe vi var klar over og alle dokumenter som trengs for innlevering er på plass.

Konklusjon
Denne iterasjonen har vært krevende. Det har vært mye jobbing med dårlig tid, men alt i alt så har iterasjonen gått greit for seg. Alt som måtte bli ferdig til innlevering av bachelorprosjektet er ferdig og vi er fornøyde. Det var litt synd at vi ikke fikk tid til å se litt mer på feilhåndtering så vi kunne bruke denne i programmet. Feilhåndteringen er en funksjonalitet som burde være på plass hvis KPS skal kunne bruke ACT systemet. Det er også en funksjonalitet som ikke er helt nødvendig til innlevering på grunn av at feilsituasjoner som kan oppstå nå, medfører bare at testsekvenser feiler og testen må kjøres på nytt. Eneste som er litt kritisk er hvis det skulle oppstå en kritisk feilsituasjon, da burde systemet stoppe slik at det ikke kan påføres skade på utstyr.

Dokumenter er lagt inn i pdf format og skal printes i morgen. Det har også blitt brent ut 3 CDer med nødvendig informasjon. 1 CD som inneholder alt som skal leveres og 2 CDer som er helt like og bare inneholder dokumentasjonen. Det som gjenstår nå er å ta tre kopier av prosjektet mandag den 26.05.14 og levere prosjektet.

4 Referanser

REFERANSE	DOKUMENTTITTEL	VERSJON
[1]	Prosjektplan	3.0

Tabell 24: Referanser