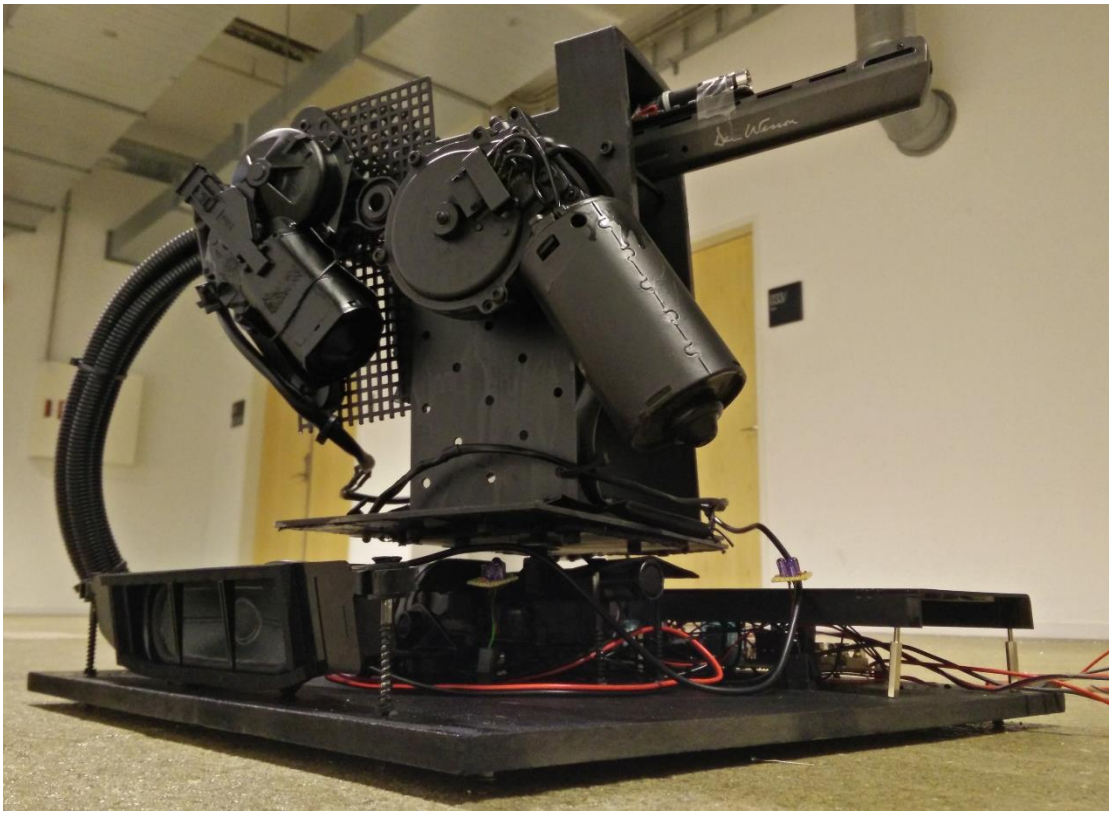
HOME DEFENSE TURRET

3. Semesterprojekt, Gruppe 7



Andreas Engelbrecht Larsen [AEL]	201370065	_____
Árni Þór Þorsteinsson [ATT]	201370318	_____
Dennis Tychsen [DT]	201311503	_____
Joachim Dam Andersen [JDA]	201370031	_____
Kasper Behrendt [KB]	20071526	_____
Lars Rudbæk Andersen [LRA]	201370796	_____
Steffen Fog [SF]	201370497	_____

Vejleder: Michael Alrøe

Dato: 28/05-2015

1. Resumé [JDA]

Denne projektrapport indeholder beskrivelse af semesterprojektet for 3. semester på Elektro-IKT og Stærkstrøm. Projektets formål, er at udvikle en Home Defense Turret prototype. Det endelige produkt vil bestå af et motoriseret skydevåben, der kan kontrolleres fysisk gennem et *brugerinterface*.

Produktet er udarbejdet med et indlejret Linux system, Raspberry Pi, samt en PSoC4. *Brugerinterfacet* samt underliggende funktionalitet er implementeret på det indlejrede Linux system, imens motor- og sensorstyring implementeres blandt andet på PSoC4. Med denne projektrapport følger en CD-ROM med relevante bilag, samt en projektdokumentation, hvor der findes dybdegående beskrivelser af produktets elementer.

2. Abstract [JDA]

This project contains a description of the semester project for 3rd semester students from Elekto IKT and stærkstrøm. The projects objective is to develop a Home Defense Turret prototype. The final product will consist of a motorized weapon, which can be physically controlled through a user interface. This project was developed with an embedded Linux system, Raspberry Pi, and a PSoC4. The user interface and the underlying functionalities are implemented on the embedded Linux system, while motor and sensor control, among other things are implemented on the PSoC4. With this project report follows a CD-ROM with supporting documentation, as well as a project documentation where there is in-depth descriptions of every product feature.

3. Indholdsfortegnelse

4. Forord [KB]

Denne rapport er resultatet af et projektarbejde udført på IKT-, Elektro- og Stærkstrømsuddannelserne på Ingeniørhøjskolen – Aarhus Universitet. Rapporten er udarbejdet af gruppe 7 i forbindelse med kurset E3PRJ3, som er sat til 5 ECTS-points.

Til udarbejdelse af en prototype samt dertilhørende projektrapport/projektdokumentation forventes der at hver studerende ligger minimum 142,5 timer i projektarbejdet.

5. Opgaveformulering [LRA]

Opgaven, i dette projekt, er at lave en HDT (Home Defense Turret) der hjælper en landmand med at sikre sin ejendom mod rovdyr. Denne skal kunne implementeres der hvor landmanden har brug for det. Landmanden skal selv kunne styre HDT'en, horisontalt og vertikalt, fra et *brugerinterface*, der kan stå langt væk fra selve HDT'en. Landmanden vil være i stand til at se hvad han/hun skyder efter, igennem et kamera og han/hun sigter ved hjælp af en laser-pointer.

HDTen advarer landmanden, om at der er fremmede iform af en alarm. Landmanden kan slå denne alarm til og fra i *brugerinterfacet.* Så han ikke behøver at høre på alarmen mens det er han opererer HDTen.

Selve systemet vil være beskyttet af en adgangskode som forhindrer fremmede eller børn i at benytte sig af det(1905).

Der skal også være en log der holder styr på antal skud og bevægelsesaktivitet. Så man ved hvornår der er blevet skudt eller gået en alarm.

6. Projektafgrænsning [LRA]

Fra IHA's side er der på forhånd defineret nogle krav til projektets indhold, hvilket indebærer at:

- Der skal indgå minimum én sensor og én aktuator.
- Der skal indgå brugerinteraktion.
- Der skal bruges et Indlejret Linux System og en PSoC4.
- Kommunikationen mellem indlejret Linux system og PSoC4 skal være en seriel kommunikation såsom SPI, I2C, RS232 eller lignende.

Ud fra opgaveformuleringen ønsker vi at lave et system der kan hjælpe en landmand, med at beskytte sine ejendom.

Til at styre HDT'en vil der være et joystik der taler direkte med det Indlejrede Linux System.

Da IHA ikke tillader at eleverne arbejder med 230V og der skal findes en strømforsyning der går ind under kategorien svagstrøm.

Der vil blive arbejdet hårdt for at få lavet en fuldt funktionel prototype.

Pistolen vil være af en ikke dræbende kaliber.

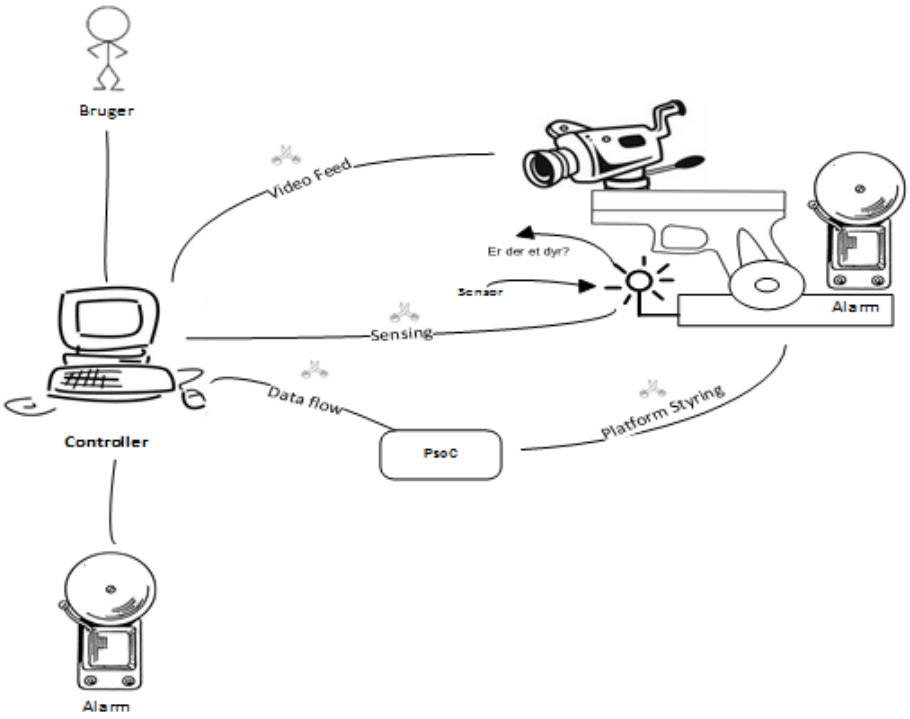
Systembeskrivelse [SF]

"Home Defense Turret" består først og fremmest af en motorstyret pistol, som kan styres ved hjælp af et joystick. På et eksternt Indlejret Linux System kan man logge ind i et *brugerinterface*. Brugeren logger ind med et matrixkeyboard og dette bruges til at navigere rundt i *brugerinterfacet*. Når man er logget ind på *brugerinterfacet*, har man fuld kontrol over den motorstyrede pistol. I *brugerinterfacet* er der en menu, der indeholder forskellige knapper. Der er adgang til at kunne afspille lyd ud af to højtalere. Der er koblet en højtaler til det Indlejrede Linux System. Højtaleren afspiller en lyd, hvis der er bevægelse ude foran pistolen. Her sidder der en ekstern PIR-sensor[1], der holder øje med, om der er uønsket aktivitet på ens område. PIR-sensoren vil opfange bevægelse og herfra vil Højtaleren begynde at afspille en lyd, hvis den er aktiveret. Der sidder også en højtaler på selve den platform, som pistolen står på. Denne kan brugeren af systemet også vælge at aktivere. Denne højtaler afspiller en advarselslyd, der ikke er til at tage fejl af.

Brugeren af systemet kan også vælge at genlade sit våben. På *brugerinterfacet* kan man se, hvor mange skud der er tilbage i pistolen. Når der er 0 skud tilbage, er det ikke muligt at affyre pistolen. Trykker man på knappen *genlad*, deaktiveres systemet, da brugeren selv er nødt til at gå ud til pistolen, for at fylde dens magasin op. Når brugeren har ladt pistolen op skal der huskes at indskrives hvor mange skud, der er fyldt på pistolen og herefter er pistolen klar til at affyres igen.

[1] Passive Infrared Sensor

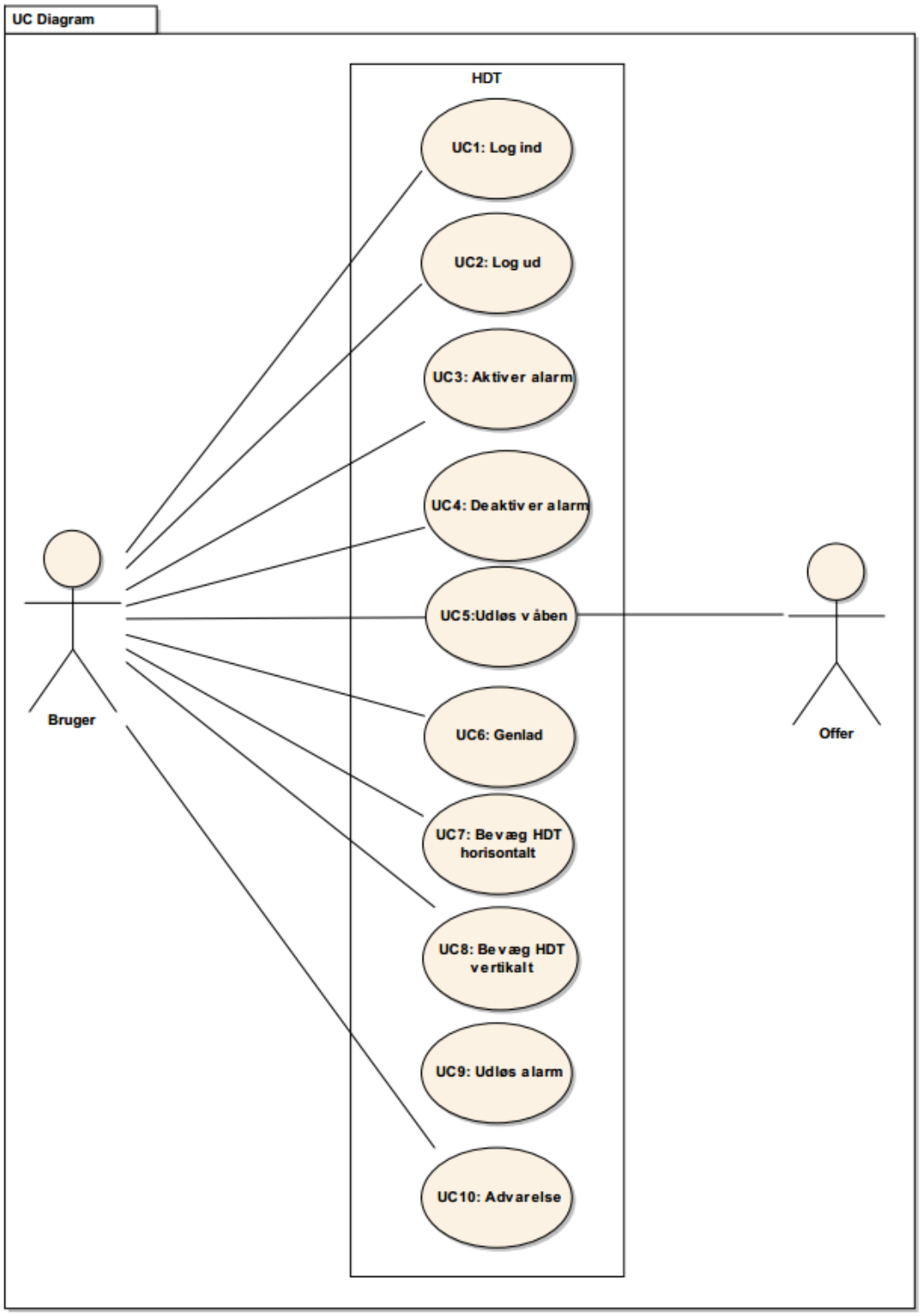
Brugeren kan se hvor pistolen peger hen, via et kamera, der sidder på platformen sammen med pistolen. Denne er tilsluttet det Indlejrede Linux System, hvor der også er tilsluttet en skærm. Herudover er tilsluttet en laser på pistolen for at brugeren nemmere kan sigte. På Figur 1 ses en illustration af projektet.



Figur 1 - Rigt billede af systemet

7. Krav [ATT]

Systemet skal tillade brugeren at styre en Home Defense Turret (HDT) med et Joystick og en Trigger. Derudover skal systemet tillade adgang til systemet med en adgangskode, som åbner for et *brugerinterface* og dermed aktivere systemet. HDT'en skal vise et kamerafeed gennem hovedmenuen. HDT'en skal kunne registrere et offer og give en alarm til systemets bruger, i lydform og som tekst i menuen. Brugeren kan herudover, gennem hovedmenuen afgive en advarsel til et evt. offer. På Figur 2 nedenfor ses et Use case diagram over systemet.



Figur 2 – Use Case Diagram

7.1. Use Case 1: Log Ind

Brugeren ønsker at låse systemet op. Dette gøres ved at indtaste en firecifret pinkode i Log Ind menuen. Det afsluttes med firkant, hvorefter kodelåsen verificerer om koden er korrekt. Hvis koden er korrekt, vil hovedmenuen derefter låses op og brugeren vil herefter have adgang til systemet. Hvis koden er forkert vil log ind menuen udskrive, at en forkert kode er indtastet.

7.2. Use Case 2: Log Ud

For at låse systemet skal brugeren trykke på ”Log Ud” knappen i hovedmenuen. Derefter låses systemet og Log Ind menuen vises igen.

7.3. Use Case 3: Aktiver Alarm

Use Case 1 skal være gennemført. Brugeren ønsker at aktivere alarmen som informere brugeren af systemet, om et offers indtrængen. Brugeren trykker på ”Aktiver” knappen i hovedmenuen, som herefter udskriver at alarmen er aktiveret. Hvis brugeren logger ud forbliver alarmen stadig aktiv.

7.4. Use Case 4: Deaktiver Alarm

Use Case 3 skal være gennemført. Brugeren ønsker at deaktivere alarmen som registrerer offeret. Brugeren trykker på ”Deaktiver” knappen på hovedmenuen. Hovedmenuen udskriver at alarmen er deaktiveret.

7.5. Use Case 5: Udløs Våben

Use Case 1 skal være gennemført. Brugeren ønsker at affyre et skud og trykker på ”Trigger” knappen, hvor HDT herefter vil affyres. Dette resulterer i at ammunitionstælleren dekrementeres. Hvis ammunitionstælleren viser ’0’, kan der ikke affyres et skud.

7.6. Use Case 6: Genlad

Use Case 1 skal være gennemført. Brugeren ønsker at genlade våbnet. Brugeren trykker på ”Genlad” knappen i hovedmenuen. Systemet deaktiveres og ”Genlad” menuen vises i brugerinterfacet. Her har brugeren mulighed for at indtaste antal skud, som er indsat i våbnet. Brugeren trykker på firkant og systemet aktiveres igen.

7.7. Use Case 7: Bevæg HDT Horisontalt

Use Case 1 skal være gennemført. Brugeren ønsker at bevæge HDT’en horisontalt. Brugeren bevæger Joysticket mod højre eller venstre. HDT’en bevæger sig nu i den valgte retning.

7.8. Use Case 8: Bevæg HDT Vertikalt

Use Case 1 skal være gennemført. Brugeren ønsker at bevæge HDT'en vertikalt. Brugeren bevæger Joysticket op eller ned. HDT'en bevæger sig nu i den valgte retning.

7.9. Use Case 9: Udløs Alarm

Use Case 3 skal være gennemført. PIR sensoren registrerer et offer og højtaleren ved brugeren lyder. Samtidig udskrives på brugerinterfacet, at et offer er registreret.

7.10. Use Case 10: Advarsel

Use Case 1 skal være gennemført. Brugeren ønsker at give en advarsel til et evt. offer. Brugeren trykker på "Advarsel" knappen i hovedmenuen. Højtaleren ved HDT afgiver da en advarselslyd.

8. Projektbeskrivelse

I dette afsnit er de teknikker og arbejdsmetoder, der er brugt til udarbejdelsen af projektet, beskrevet. Projektstyring og rollefordelingen er beskrevet, samt hvordan projektets struktureringen er foregået, og hvilke teknikker der er brugt til dette.

8.1. Projektgennemførsel og metoder [JDA, DT]

I dette afsnit beskrives styringen og gennemførelsen af dette semesterprojekt. Der er brugt en række værktøjer og principper, der havde til mål at definere, hvordan projektet skulle udarbejdes, samt struktureringen af projektet. På denne måde var der hele tiden et overblik over, hvad der skulle laves, og hvad der var færdigt.

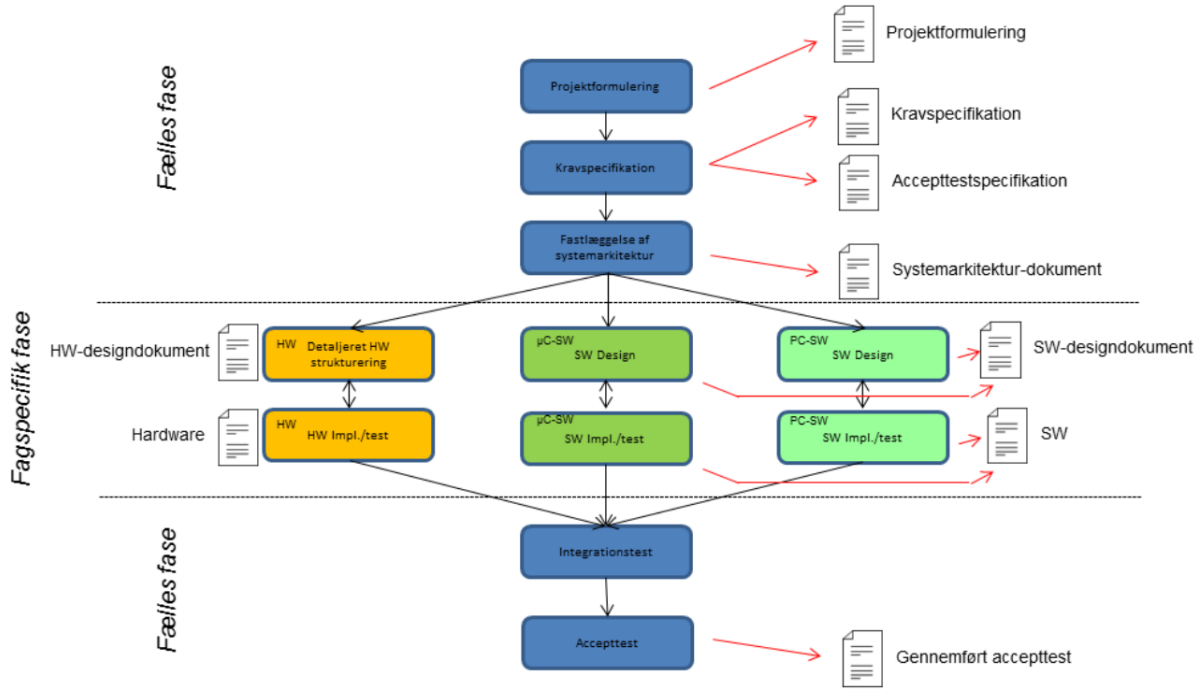
Dette afsnit indeholder følgende elementer:

- Projektmodellen – ASE modellen
- Projektgennemførsel med SCRUM
- Projektstyring
- SysML
- Brug af Git

Projektet er grundlæggende udarbejdet efter den velkendte ASE-model, som blev brugt første gang på 1. semester. Denne model definerer en god proces til udarbejdelse af semesterprojekterne, samtidig med at der kører sideløbende kurser, som skal bruges i projektet.

Udover ASE-modellen er nogle af grundprincipperne fra den agile styringsform SCRUM blevet brugt. Til projektet er der brugt en nedtrappet og meget grundlæggende version af SCRUM, hvor kun de mest grundlæggende SCRUM-terminologier er taget i brug. Dette vil være beskrevet i større detalje lidt senere i dette afsnit.

8.1.1. ASE modellen



Figur 3 - ASE modellen

ASE-modellen er god at bruge, til semesterprojekter som dette. Dette skyldes, at den viden og teori der anvendes i semesterprojektet er viden der bliver undervist i sideløbende med projektet, igennem semestret. ASE modellen starter ud med, at der udarbejdes en projektformulering. I denne defineres der, hvad projektet går ud på, hvad der ønskes fremstillet og en meget overordnet forklaring af, hvordan produktets funktionalitet skal være. Når projektformuleringen er fremstillet og godkendt, så begyndes der på en kravspecifikation. I denne defineres der specifikt, hvad produktet skal, hvordan og hvornår. Dette gøres som regel ud fra ”Use cases”, som er tabeller med gennemgange af alle de funktionaliteter, der ønskes implementeret. Under udviklingen af kravspecifikationen udvikles også en accepttestspecifikation. Denne beskriver de tests systemet skal gennemføre og på hvilken måde, før de pågældende krav er opfyldt og implementeret tilfredsstillende.

Herefter nås det sidste skridt i den indledende fælles fase, nemlig fastlæggelsen af systemarkitekturen. Systemarkitekturen er den del, hvor der besluttes, hvilke moduler systemet skal indeholde og hvordan grænsefladerne mellem disse moduler skal være. Arkitekturen bekrives som regel ved hjælp af SysML diagrammer, som hurtigt giver et overblik over produktets struktur og indhold. Alt dette er en fælles fase, hvor hele teamet skal være enige om, hvad produktet indeholder og hvordan alting snakker sammen.

Når alle disse ting er på plads skal der laves detaljeret design af alle komponenter. Hardwaren skal designes, såvel som softwaren. Denne fase laves oftest opdelt, hvilket er muligt, da den overordnede systemarkitektur skal være fastlagt fra før, så udviklerne af hardware er klar over hvordan der skal kommunikeres med software og vice versa.

Efter den detaljerede designfase skal det hele implementeres. Det er ofte her de reelle problemer opstår. Jo bedre produktet er designet i den detaljerede designfase, og jo mere der er tænkt over alle de problemstillinger der måtte komme, jo færre fejl skulle forhåbentlig forekomme under implementeringen. Under implementeringen enhedstestes alle moduler der fremstilles, og efterfølgende nås integrationstestene.

Integrationstestene er det punkt, hvor modulerne kobles sammen med de moduler de skal snakke med. Her testes om modulerne også virker sammen med andre reelle dele af produktet, og ikke blot i et enhedstest-miljø, hvor inputs er kontrollerede. Jo bedre modulerne er enhedstestet, jo færre problemer opstår der under integrationstesten. Gode enhedstest tester også for uhensigtsmæssige eller forkerte input, for at se, hvordan det pågældende modul reagerer på dette.

Til sidst afholdes en accepttest, hvor hele systemet gennemgår en samlet test, som blev defineret under fremstillingen af kravspecifikationen. Accepttestens tests skal kunne udføres med det korrekte resultat, før produktet kan markeres som færdigt. Det er ud fra accepttesten at kunden kan se, om det produkt der er blevet fremstillet, stemmer overens med det lovede produkt.

Måden, hvorpå projektets produkt udvikles, når ASE modellen følges er især god til semesterprojekter, hvor der hele tiden sideløbende bliver tilegnet ny viden, der skal indgå i projektet. Først forholdsvis sent i processen begyndes der på det detaljerede design og implementering af produktet. Dette betyder at man som studerende kan nå at tilegne sig en del viden, som kan udnyttes til fremstillingen af produktet.

8.1.2. SCRUM i projektet

SCRUM er en agil udviklingsmetode udviklet i 1990'erne. SCRUM bruges primært til softwareudvikling, og bruges blandt andet af store softwarehuse som Systematic[2]. Et af hovedpunkterne i SCRUM er, at man skal "gøre det der giver mening ".

[2] https://systematic.com/how-we-work/approach/scrum-solutions-that-fit-real-needs/ - Agile Development Based On Scrum

Til dette projekt var der enkelte ting fra SCRUM der gav mening at bruge. En modificeret udgave af sprints, SCRUM boardet og backlog.

Sprints er små intervaller, hvor der defineres nogle opgaver, der skal nås og hvornår deadline er. I starten blev disse brugt meget, men tonede ud, efterhånden som der ofte kom opgaver op i semestrets kurser, der skubbede de forskellige deadlines.

SCRUM boardet er et board der giver overblik over, hvilke opgaver der er tilgængelige, hvilke er under udvikling og hvilke der er færdige eller skal kigges igennem af gruppens andre medlemmer.

Backloggen er blot en liste over de opgaver der skal udføres. Backloggen er der hvor opgaver tages fra, og startes udvikling af, hvorefter de bliver markeret som færdige, når opgaven er udført.

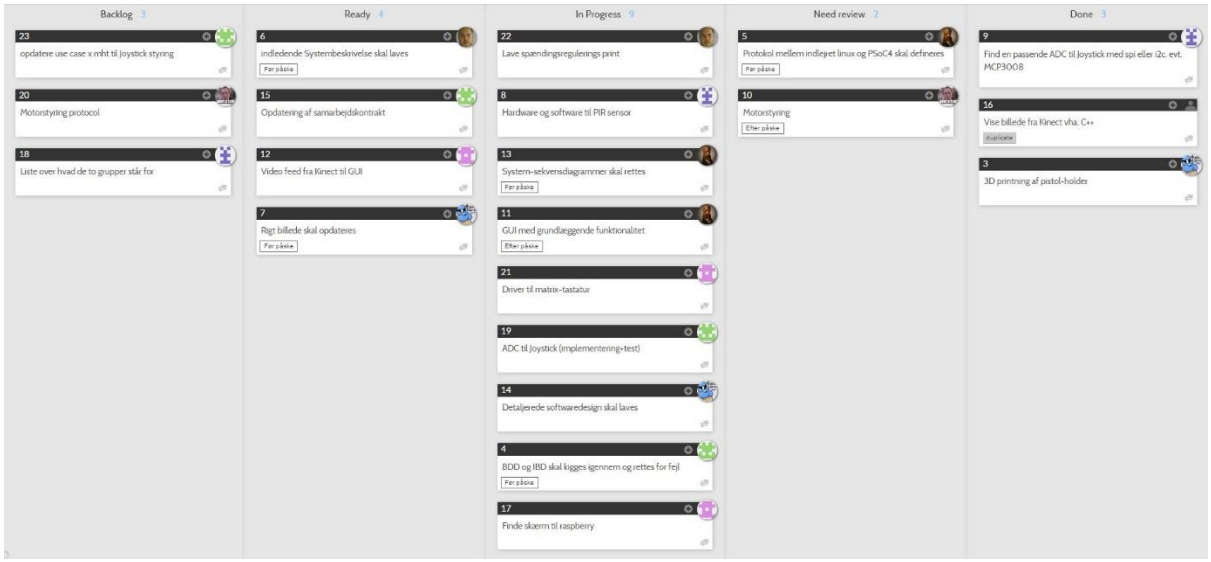
Til projektet er anvendt en webbaseret løsning til SCRUM boardet, som fungerer i samarbejde med Github[3]. Github bruges til versionstyring af softwaren. SCRUM boardet og Git forklares mere om i næste afsnit.

[3] https://github.com/

8.1.3. Github i projektet

Til at organisere projektet er der brugt et webbaseret SCRUM board. Dette SCRUM board fungerer ved at kategorisere Github issues. Github er en hjemmeside til hosting af Git repositories. Git er et program til versionstyring af software. Et *issue* kan bruges til at beskrive en opgave der skal laves, og kan markeres med et label. Dette label bliver brugt på SCRUM boardet. På SCRUM boardet er der fem kategorier. Backlog, Ready, In Progres, Need review og Done. Alle nye opgaver bliver automatisk lagt i backloggen. Når gruppen har aftalt at opgaven er klar til at blive udført, så flyttes den til "Ready". Herefter kan en eller flere gruppemedlemmer begynde at udføre opgaven, og den flyttes derfor til "In progress". Når opgaven er udført flyttes den til "Need review", så gruppens resterende medlemmer kan tjekke arbejdet igennem, og godkende det som tilfredsstillende. Når arbejdet er reviewet og godkendt flyttes opgaven til "Done", og er derefter færdig. Således er det nemt at skabe sig et overblik over nye opgaver, ting der er i gang og hvad der er blevet gjort færdigt. Dette skaber overblik og man kan nemt finde en opgave at give sig til.

SCRUM boardet ser således ud.



Figur 4 - SCRUM board i en tidlig fase af projektet

På Figur 4 ses en tidlig udgave af SCRUM boardet. Der er mange opgaver i gang med at blive færdiggjort, et par stykker i "Need review" og nogle i Done ligeså. Backloggen har efterfølgende fået mange flere opgaver, og flere nye er kommet til hele tiden.

8.1.4. SysML i projektet

Til at udarbejde projektets mange faser, er der anvendt SysML. SysML står for System Modeling Language, og er en måde at beskrive og visualisere systemers opbygning.

Til udarbejdelsen af systemarkitekturen er der anvendt Block Definition Diagrams (BDD) og Internal Block Diagrams (IBD), samt overordnede sekvensdiagrammer og en domænemodel. BDD bruges til at beskrive de fysiske komponenter i systemet. IBD beskriver de interne grænseflader mellem de fysiske blokke fra BDD'et. Domænemodel og sekvensdiagrammer bruges til at beskrive den overordnede programmæssige funktionalitet i systemet.

I de detaljerede design og implementeringsdokumenter er der igen anvendt BDD og IBD, samt detaljerede sekvensdiagrammer, state machines og aktivitetsdiagrammer. De sidste tre bruges til at beskrive det detaljerede design af softwaren, som efterfølgende bruges, når softwaren skal implementeres.

8.1.5. Rollefordeling

Da projektgruppen består af studerende fra de tre forskellige retninger, IKT, Elektro og Stærkstrøm, var det oplagt at inddele gruppen I to mindre grupper. Grupperne havde da hvert deres ansvar inden for hhv. software og hardware. Dog med introduktionen af PSoC4 og dens brug i projektet, tilfaldt opgaven med programmering heraf, hardware-gruppen. Software udviklet til Raspberry Pi herunder brugerinterfacet samt al underliggende funktionalitet, på nær SPI[4] interfacing med Joysticket blev udviklet af softwaregruppen. Dog er det ikke alt arbejdet i projektet der deles op mellem grupperne.

Som ASE-modellen viser, se Figur 3, ses det hvorledes den første og den sidste fase indebærer samarbejde mellem de to grupper. Dette samarbejde koncentrerede sig om udarbejdelse af projektformuleringen, kravspecifikationen og deraf accepttestspecifikationen. Derefter udarbejdedes systemarkitekturdokumentet i fællesskab, hvorefter grupperne gik hver til sit, og arbejdede på deres fagspecifikke opgaver. Da design og implementering stod klart hos grupperne, blev der afholdt integrationstest, og derefter accepttest. Det skal dog pointeres at der mellem de fælles og fagspecifikke faser stadig blev kommunikeret, både under og uden for de fælles gruppemøder.

8.1.6. Gruppestruktur og samarbejdskontrakt

Der er i projektgruppen ikke valgt en specifik leder. Problemstillinger og beslutninger blev derimod taget hånd om ved fælles gruppemøder på demokratisk vis. Dog var opgaver som at tage referat ved møderne, samt skrive dagsorden og sende mødeindkaldelser ud, altid bemandet. Hvem der gjorde hvad blev besluttet

[4] Serial Peripheral Interface

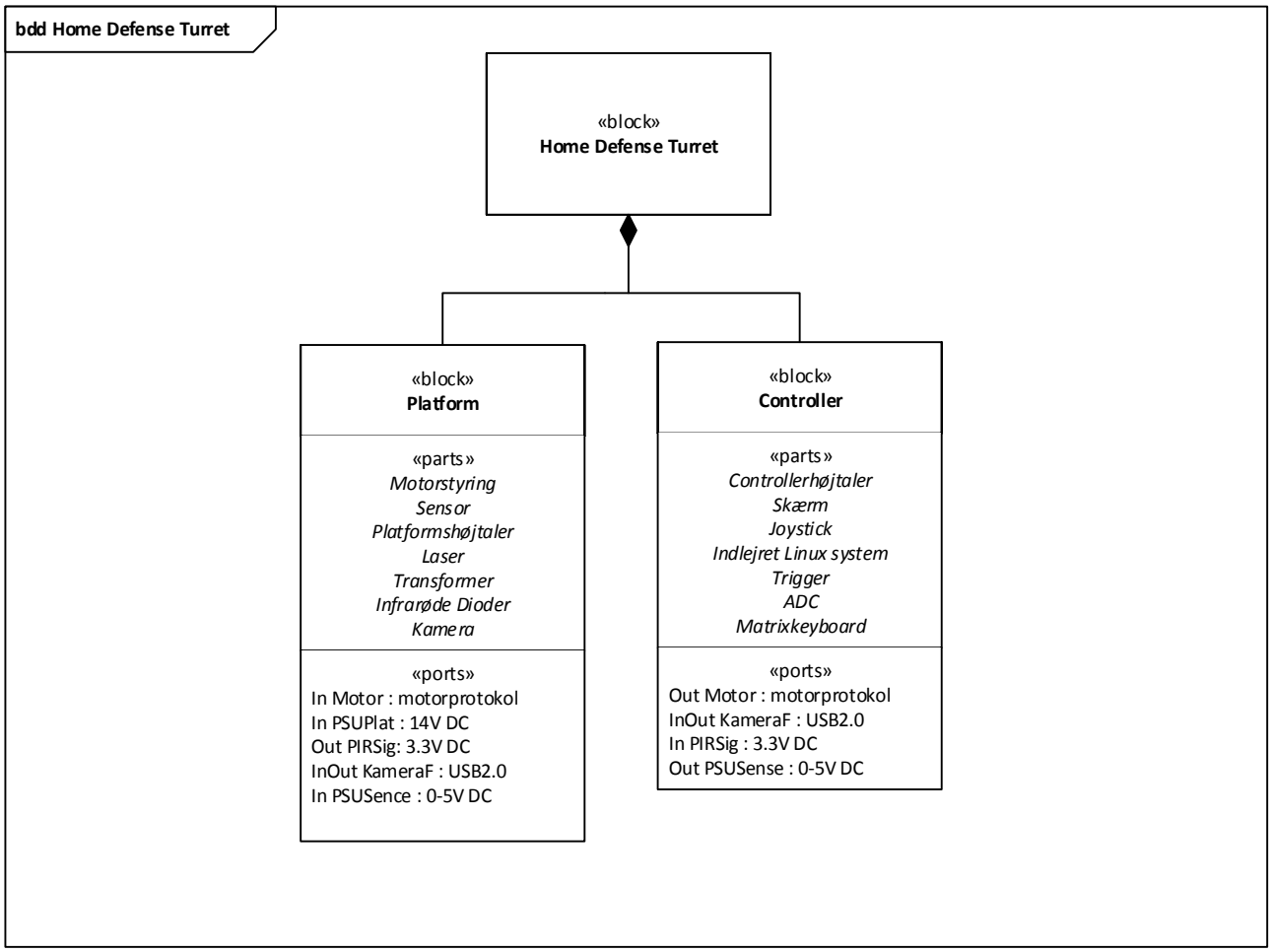
ved mødet før. Som udgangspunkt blev der afholdt gruppemøde en gang om ugen, på en fast defineret dag. På møderne blev SCRUM-boardet[5] opdateret, og der blev talt om, hvilke opgaver der var løst, samt evt. forhindringer for dette. I begyndelsen af projektet blev der udarbejdet en samarbejdskontrakt, der skulle sørge for en fælles vision for projektet, en mødeprotokol og indeholde visse grundregler for samarbejdet samt konsekvenserne hvis reglerne skulle blive brudt.

[5] Se Figur 4 - SCRUM board i en tidlig fase af projektet

9. Systemarkitektur [SF, KB]

I dette afsnit beskrives arkitekturen for Home Defense Turet. Systemarkitekturen er blevet brugt som udviklingsramme for design og implementering. Herudfra kan systemets komponenter bestemmes. Her er de enkelte blokke beskrevet med hensyn til funktionalitet, der er beskrevet i systembeskrivelsen og kravspecifikationen, som kan nedbrydes til mere overordnede moduler.

Home Defense Turret er opdelt i to overordnede blokke, Platform og Controller.



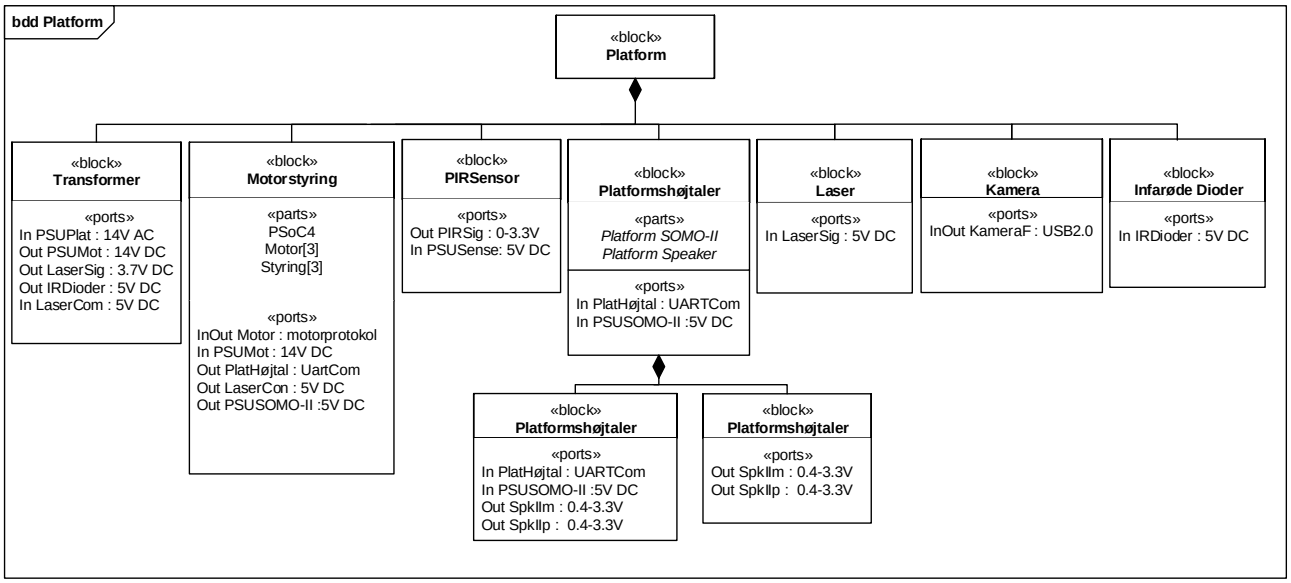
Figur 5 - BDD af Home Defense Turret

På selve platformen sidder pistolen og motorerne der styrer HDT vertikalt, horisontalt samt pistolens aftræk.

Controllerblokken indeholder de komponenter, som brugeren benytter til at styre platformskomponenterne.

9.1. Blokidentifikation af Platform

For at brugeren kan udføre de specificerede Use Cases, er der en række blokke, som platformen er nødt til at indeholde. Platformen indeholder moduler der er nødvendige for at brugeren kan interagere fra Controllereren til Platformen. De identificerede moduler der ønskes for at opfylde platformens funktionaliteter er illustreret i det følgende BDD.



Figur 6 - BDD af platform

9.1.1. Blokbeskrivelse af Platform

Transformer sørger for Powersupply til Motorstyring, Laser og infrarøde dioder.

Motorstyring sørger for selve kommunikationen mellem Platform og Controller-blokken, samt styring af motorer. Blokken Motorstyring indeholder en PSoC4, der kommunikerer med det Indlejrede Linux System fra Controller blokken. Når brugeren fx bevæger Joysticket op, modtager Motorstyring et signal om dette, og sørger derefter for, at den ønskede handling forekommer.

Når der forekommer bevægelse, registrerer **PIRsensor** dette. PIRsensor sender sit signal om, at der foregår en bevægelse, videre til Controller blokken.

Platformshøjtaler blokken afspiller en lyd, hvis den modtager en besked om dette. Dette sker via PSoC4 i Motorstyring, hvis brugeren har anmodet om dette fra Controller blokken.

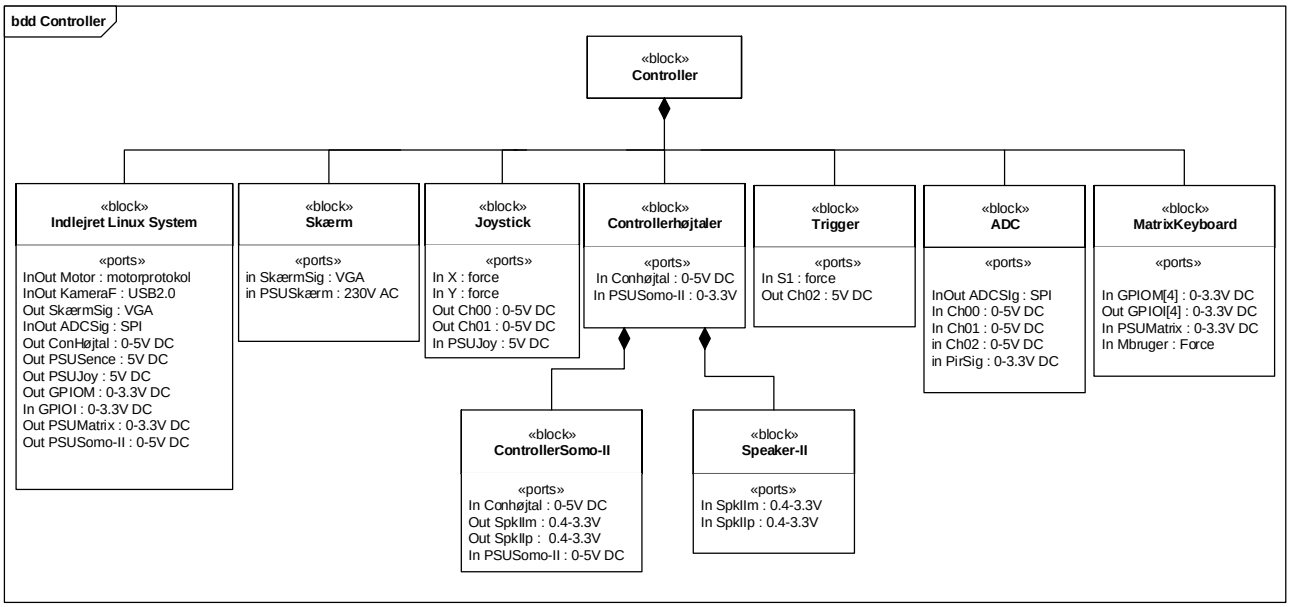
Laser tændes, når systemet er blevet aktiveret af Controller blokken.

Brugeren ønsker at se, hvad pistolen peger imod. Der er tilføjet et **Kamera,** der sørger for et kamerafeed, der er tilsluttet det Indlejrede Linux System på Controller-blokken.

For at **Platform** blokken ikke bevæger sig for langt til siderne og for langt op og ned, skal motorerne stoppe. Dette sørger **Infrared Diode** blokken for**.** Der skal tilsluttes tre enheder med hver tre dioder for at kunne opfylde kravene.

9.2. Blokidentifikation af Controller

Controller blokken er blevet identificeret ud fra, hvordan brugeren skal anvende systemet og hvilke komponenter, der skal spille sammen for at muliggøre Use Casene samt systembeskrivelsen. Controller blokken skal indeholde blokke der gør, at brugeren kan bevæge motorerne, affyre pistolen og se, hvad brugeren skyder imod.



Figur 7 - BDD af Controller

9.2.1. Blokbeskrivelse af Controller

Indlejret Linux System er kernen i Controller blokken. Stort set al kommunikation mellem komponenterne i Controller blokken, går igennem Indlejret Linux System blokken. Indlejret Linux System er også grænsefladen imellem Controller og Platform. Indlejret Linux System indeholder alt software for at brugeren kan opfylde Use Casene.

For at brugeren kan se, hvor pistolen peger hen, er der tilsluttet en **Skærm.**

Brugeren kan styre motorerne op og ned, samt fra side til side. Dette kan brugeren gøre ved hjælp af et **Joystick**. **ADC** blokken skal bruges til at fortolke joystickket og triggeren. **Trigger** kan brugeren trykke på, for at affyre pistolen, da der er koblet en motor på pistolens aftrækker, som triggeren aktiverer.

Matrixkeyboard er tilsluttet, så brugeren kan logge ind og ud af det Indlejrede Linux System og trykke på tasterne i *brugerinterfacet*.

Der skal afspilles en alarm, når PIRsensoren opfanger bevægelse ved HDT. Derfor er der tilsluttet en **Controllerhøjtaler**.

10. Hardware Design og Implementering

10.1. Platform Hardware

10.1.1. Motorstyring Hardware [ATT]

Idéen med motorstyringen er at bevæge motorer i horisontale og vertikale retninger samt at styre hastigheden. Der var erhvervet viskemotorer til denne opgave. Disse motorer skal kunne bevæge sig i forskellige retninger. H-bro var den første idé som blev overvejet. Den kan skifte retning med styrelse af relæer. De relæer kan så blive styret af en MOSFET, som er forbundet til en microkontroller. Den samme microkonroller vil så styre hastigheden på motorene med et PWM signal, som bliver sent til en anden MOSFET. Den samme ide blev brugt til trigger motor styring. Den bliver også styret af relæ'er, som åbner for belastning til motoren.

Den metode som blev brugt til at udvikle motorstyringen var i princippet en "Learn by doing" metoden. Det blev implementeret en idé, som blev bygget op på et fumlebræt. Derefter blev det testet til at se, om det virkede som det skal. Til sidst bev det så testet med softwaren til at tjekke, om det fungere også med forbindelse til microcontrolleren.

10.1.2. Dødswich hardware[ATT]

Idéen med dødswicherne er at horisontal og vertical motorene kan ikke bevæge sig for meget til en bestemt retning som brugeren vælger. Der er valgt at bruge infrarøde LED'er og fotodioder. Grunden til at der ikke bruges en almindelig LED er at HDT'en skal kunne klare at være udenfor, hvor det kan være meget lyst. Dette lys kan drille fotodioden og systemet vil være ustabilt. Når der bruges infrarøde dioder vil noget i den retning ikke ske. Den infarøde fotodioder vil så give en logiske high til microkonrolleren som vil aflæse det og stoppe motoren som kører.

Udviklings procsessen blev den samme og i motorstyring processen, somsagt ”Learn by doing” metoden. Eneste forskelle var at der opståede en fejl, når systemet blev forbundet til microkontrolleren,som så senere blev fixet (læs Projektdokumentation[6]).

10.1.3. SOMO-II Platform Hardware [ATT]

Til at afspille en advarelse blev det besluttet at bruge en SOMO-II. SOMO-II’en er valgt på grund af de smarte funktioner, som kan bruges til at styre den. Den kan også aflæse direkte af et SD-kort således, at man kan kontrollere hvilken lydfil, der bliver afspillet og hvornår. Hardwaren som blev brugt til SOMO-II’en er ikke noget som blev udviklet I dette project. SOMO-II’en har allerede en bestemt måde som den skal forbindes på, som blev blot aflæst i databladet. Den måde som den blev forbundet på kan aflæses i Projektdokumentation, afsnit 9.1.4.

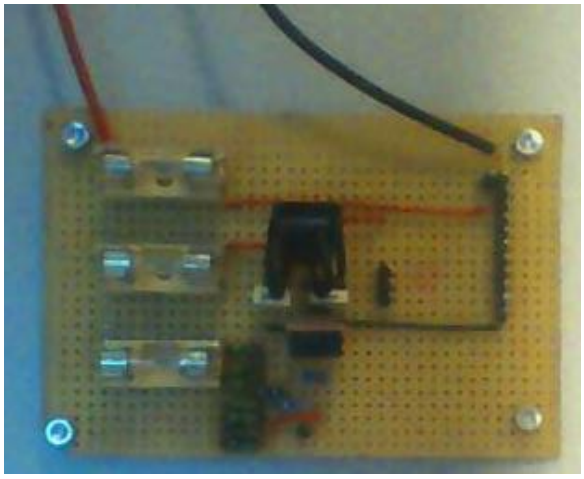
10.1.4. Spændingsforsyningsprint [LRA]

Til at forsyne platforrmen med strøm skal der bruges et spændingsreguleringsprint, da det ikke er alle ting på platforrmen der skal have 14V. Motorerne på platformen er det eneste der skal have 14V, laseren skal have 3,7V og dioderne skal have 5V.

For at opnå spændingen på 5V blev der benyttet et en LM7805 som er en standart 5V spændingsregulator. For at ungå at LM7805 brænder sammen er der sat en køleplade på.

Til regulering af 3,7V forsyningen blev der overvejet flere muligheder, som et specielt kredsløb med en LM7805, en formodstand eller en LM7805 sammen med en formodstand. Der blev valgt at bruge en formodstand, da det virkede mest simpelt, men der kom et uventet problem, da modstanden brændte over, så det blev udregnet, at syv modstande burde være nok, men de kunne ikke klare at blive udsat for den mængde stress i en længere periode. Til sidst blev der påsat to 5Ws modstande for at sikre at det ikke ville brænde sammen.

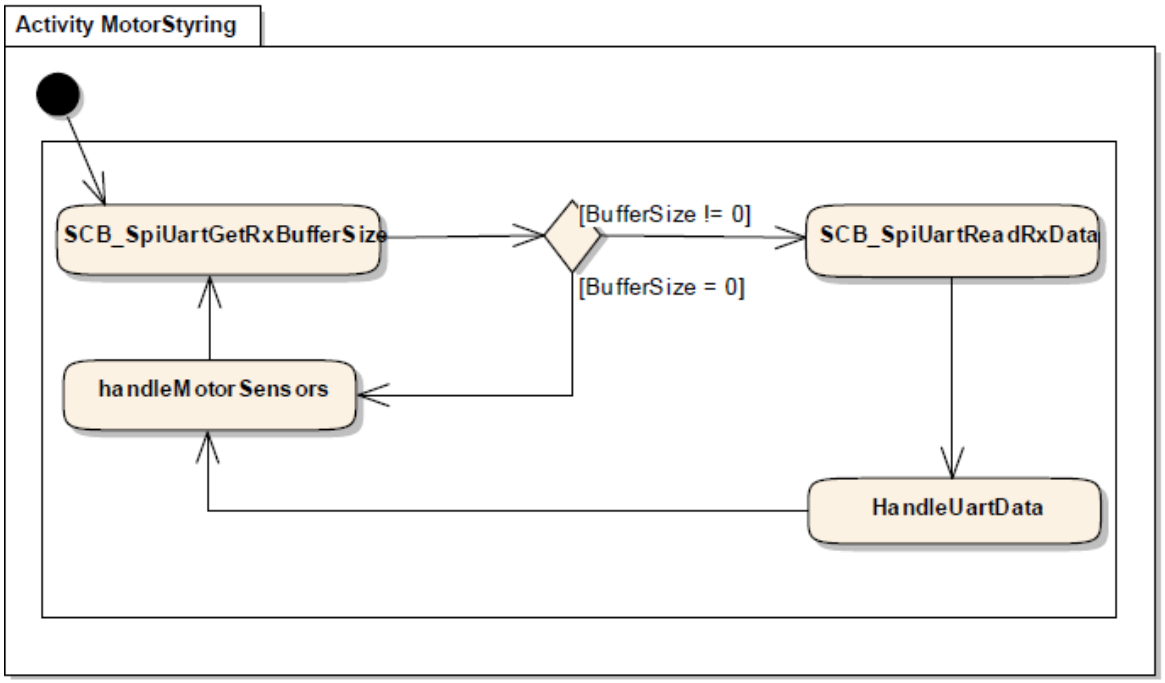
[6] Se afsnit 9.1.3 i projektdokumentationen.



Figur 8 Billede af spændingsreguleringsprint

10.2. Plarform Software [ATT]

Idéen med softwareren er, at den kan få UART kommandoer igennem seriel forbindelse til det Indlejerede Linux System. Når softwaren ikke får nogen kommandoer fra det Indlejerede Linux System vil den tjekke statussen på Dødswitchen. Således vil softwaren stoppe den motor som kører. Hvis motoren kører samtidigt med at den infrarøde fotodiode modtager infrarødt lys fra LED'erne, vil den stoppe. Når softwaren modtager en kommando vil den ordne det med nogen sammenligninger mellem den kommando som softwareren modtager, via UART, og den ønskede kommando. Hvis softwaren modtager start byte og tjeksummen passer vil den udføre den ønskede kommando som blev sent (læs projektdokumentation, afsnit 9.3).



Figur 9 -Activity Diagram af Motor Styring software

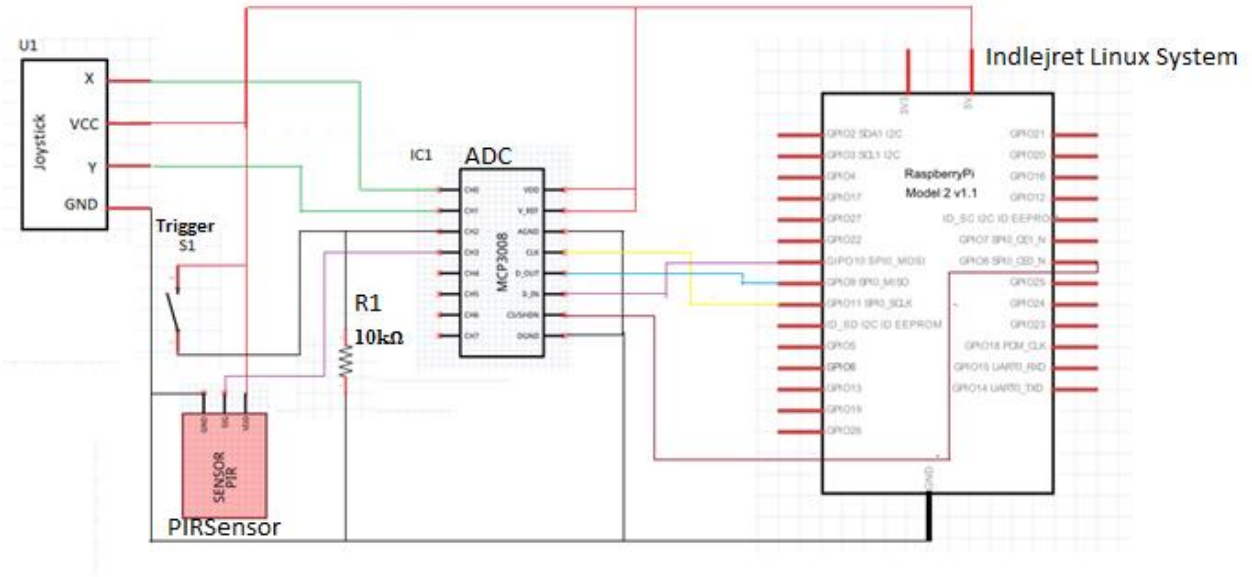
SOMO-II'en på platformen skal styres af den samme software. Dette sker også i gennem UART fordi SOMO-II'en allerede har et inbygget bibbliotek til at ordne UART kommandoer. Det er meget passende for dette projekt da der bruges en microkontroller til at styre hardwaren på platformen. Der blev nogen problemer med at få de to til at kommunikere sammen, i starten, men det blev løst (læs projektdokumentationen, afsnit 9.3.2).

Under softwaretesten blev det hovedsageligt gjort igennem et scope på grund af at microkontrolleren skal sende en fysisk spændning ud, når den ønskede funktion har blevet udført. Når det lykkedes, blev microkontrolleren forbundet til selve platformens hardwaren og testet på denne måde. Så det blev udført en enheds test for hver del og så til sidst testet det hele samlet.

10.3. Controller Hardware [SF og KB]

Følgende afsnit er en beskrivelse af det samlede design og implementering af systemets Joystick, ADC, Trigger, PIRSensor og Controllerhøjtaler

Hardwaren er simpel, da det er færdige komponenter, der er sat sammen. En opstilling af Joystick, ADC, Trigger og PIRsesor kan ses på nedenstående Figur 10. Joystick består af to poteniometre, og der var her brug for en ADC, der kunne tolke dens signaler, og sende dem videre til Indlejret Linux System. Der er tilføjet en modstand på Trigger's indgangssignal, da der ellers aflæses tilfældige værdier, når der ikke er trykket på Trigger. Ved tryk på Trigger sendes der 5V ind på indgangen af ADC'en. Der er valgt en ADC, med et SPI til denne opgave. For at SPI skal fungere, er det vigtigt at Indlejret Linux System og ADC'en er forbundet korrekt. Forbindelserne kan ses på Figur 10.



Figur 10 - Opstiling af Joystick, Trigger, PIRsensor, ADC og Indlejret Linux System

Der er oprettet en klasse, SensorsSPI, hvis formål er at håndtere softwaren imellem ADC og Indlejret Linux System, da eksterne komponenter er tilsluttet ADC. Dette gælder Trigger, PIRSensor og Joystick. Klassen indeholder også en funktion, der anvendes til at skrive til ControllerHøjtaler, da ControllerHøjtaler skal afspille en lyd, afhængigt af om PIRSensoren opfanger bevægelse.

Klassen består af en funktion for hver channel, som der er aktiv på ADC'en. Det vil sige fire, da der er to outputs fra Joystick, et output fra Pirsensor og et output fra Trigger. Disse funktioner returnerer alle sammen en integer, for at de er lettere at implementere med andet software. Herudover består klassen også af en init og exit funktion samt en transmission funktion

spi_init står for initieringen af spidev device på Indlejret Linux System. Dette gøres med en file-descriptor, der opsætter interfacet med spidev device. Her er det vigtigt at være opmærksom på, at de rigtige ioctl() funktioner kaldes. Se projektdokumentationen, afsnit 9.6.

Funktionerne for PIRSensor, Trigger og begge akser af Joystick er stort set en. Her gennemgåes funktionen for JoystickY(). De andre funktioner kan findes på bilaget på CD-Rom[7]

[7] Se bilag på CD-rom I mappen /sourcekode/SPI

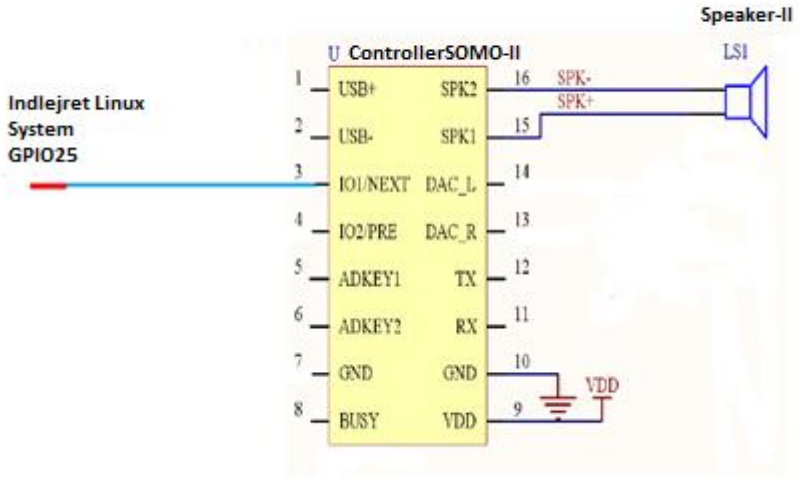
Hovedmålet med **JoystikY()** er, at der skal sendes tre bytes fra Indlejret Linux System's MOSI-GPIOport. Her skal den første byte indeholde et start bit, anden byte skal indeholde bits for, hvilken channel på ADC'en der skal samples på og tredje byte er don't care. Disse sendes til ADC'en vha. Transmission() funktionen (se projektdokumentation afsnit 9.6). Transmission funktionen retunerer 10 bits af de tre bytes der er blevet afsendt. Dette er disse data, der bliver sendt ind på MISO-GPIO'en på Indlejret Linux System. De 10 bits, der returneres som resultat, er nødt til at bit manipuleres, for at kunne retunere det rigtige resultat som en integer.

Den før omtalte **Transmission()** funktion tager sig af transaktionen imellem Indlejret Linux System og ADC. Her bliver sendt data til spidev device og data bliver sendt fra spidev device. I Tranmission() er der oprettet en struct (se projektdokumentation afsnit 9.6). Denne struct indeholder to buffere. Den ene er en pointer til usperace, med de data der skal afsendes, og den anden er også en pointer til usperspace dog med de data, der skal modtages.

Spi_exit nedlægger kommunikationen med spidev device.

10.3.1. Design og implementering af Controllerhøjtaler

ControllerHøjtaler modulet består af to blokke: ControllerSOMO-II og Speaker-II. For at afspille lyd ud af Speaker-II, er ControllerHøjtaler sat sammen som vist på Figur 11



Figur 11 - Opsætning af ControllerHøjtaller

Når der sendes 0 i mindre end 500ms til ControllerSOMO-II's ben 3, vil ControllerSOMO-II automatisk begynde at afspille lyd. Derfor er der udvikllet kode, der sætter GPIO25 fra Indlejret Linux System lavt, når PIRsensoren opfanger bevægelse, i 200ms, hvorefter lyden går høj igen. For at undgå at lyden ikke spiller oveni sig selv, er der indsat et delay, der forhindrer dette.

Koden er implementeret, så når PIRSensoren opfanger en bevægelse, så skal ControllerHøjtaleren begynde at afspille en lyd.

Lyden der bliver afspillet er et selvvalgt lydstykke. Der er indlagt tre lydsamples på microSD kortet, der sider i ControllerSOMO-II, som den automatisk skifter imellem, hver gang den begynder at afspille.

11. Software design og implementering [AEL, JDA, DT]

I dette afsnit beskrives kort design og implementering af softwaren til produktet. Softwaren vil være bekrevet med tekst og enkelte diagrammer. En mere teknisk og detaljeret gennemgang af design og implementering kan findes i projektdokumentationen.

11.1. Forsøg med brug af Microsoft Kinect [JDA]

Ved udarbejdelse af kravspecifikationen, var det Microsoft Kinect der skulle levere kamerafeed. Dette var for at der også kunne gøres brug af Kinectens motiontracking. Det var tanken at HDT uden hjælp fra brugeren, skulle kunne sigte mod dets mål. Derfor indledtes søgen efter information omkring Microsoft Kinect og udviklingsværktøjer hertil. Microsoft udbyder deres eget SDK[8], Kinect Studio. Dog var det et krav til projektet at bruge et Indlejret Linux System. Derfor var der brug for et udviklingsværktøj til en Linux platform. Websiden www. openkinect.org er et community site, hvor der kan findes information vedr. programmering og interfacing af Microsoft Kinect. Ved hjælp af adskillige Google søgninger, blev der fundet frem til en guide[9], der viser gennemgang af opsætning af Microsoft Kinect til Linux. Guiden brugte 3 open source biblioteker: libfreenect[10], OpenNI[11] og sensorKinect[12]. Opsætningen af udviklingsmiljøet lykkedes, dog ikke uden kamp. Programmering og interfacing af Kinecten var dog ikke ligetil. Bibliotekerne var dårligt dokumenterede, og eksempelkoden hertil var enormt lang og uoverskuelige. Efter længere tids forsøgen og kamp med at få blot et lille program til at virke, blev det besluttet i gruppen at vælge en anden løsning. I stedet blev det besluttet at bruge et almindeligt webcamera til at levere kamerafeedet i brugerinterfacet. Til dette bruges biblioteket OpenCV[13]. Dette bibliotek er beskrevet i afsnittet GUI i projektdokumentationen. OpenCV er er langt bedre dokumenteret end bibliotekerne til Kinect (Linux). OpenCV biblioteket byder også på funktionalitet til at tracke bevægelse (farvebestemt), og der er derfor også mulighed for at udvide produktet med denne funktionalitet, som en del af det fremtidige arbejde.

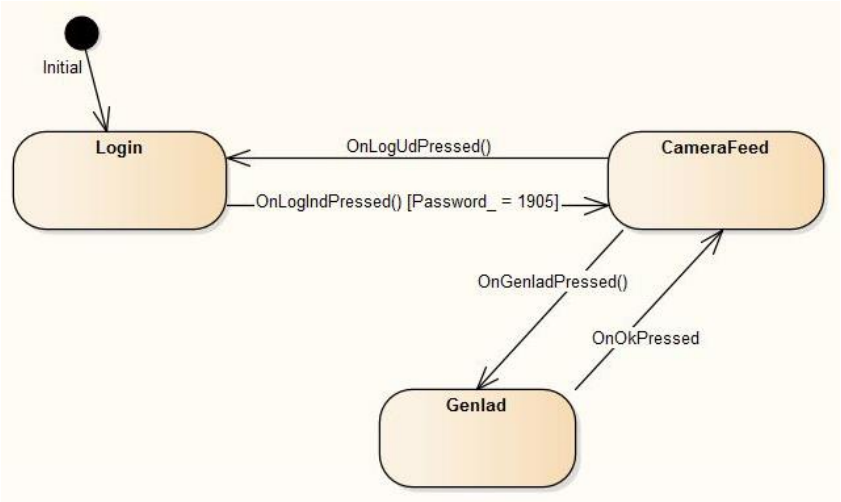
11.2. GUI [AEL]

Systemets GUI eller ”Graphical User Interface” bliver i dette projekt ofte omtalt som ”*brugerinterfacet*”. Det er her brugeren af systemet får et visuelt indtryk af de forskellige handlinger systemet udfører. Systemets *brugerinterface* er opdelt i tre vinduer: Login, CameraFeed og Genlad, og med disse tre vinduer kan der tilgås hele systemets underliggende funktionalitet.

I dette projekt er der brugt frameworket Qt4 til at skabe det visuelle *brugerinterface* da dette viste sig at fungere godt og være forholdsvis let at gå til. Udover Qt4, skulle der bruges et program eller et API til at håndterer de billeder der kommer fra systemets kamera. Det endte med at være OpenCV’s biblioteker der

[8] Software Development Kit
[9] http://www.kdab.com/setting-up-kinect-for-programming-in-linux-part-1/
[10] https://github.com/OpenKinect/libfreenect
[11] https://github.com/OpenNI/OpenNI
[12] https://github.com/avin2/SensorKinect
[13] Open Computer Vision

bedst kunne håndtere dette og derfor disse der blev brugt, i det endelige system. Dette kan dog læses nærmere om i projektdokumentationen.



Figur 12 - State machine diagram over brugerinterfacets vinduer

Her ses et simplificeret state machine diagram over de forskellige vinduer i systemets *brugerinterface* samt de forskellige triggers der gør at brugeren kan bevæge sig imellem disse.

11.2.1. Login

Denne klasse er det første man møder når programmet starter og fungerer som en lås for systemets underliggende funktionalitet. Når brugeren har indtastet et password og trykker ”Ok”, bliver det indtastede password sammenlignet med variablen password_ som er ”1905”. Hvis disse ikke matcher så vil der blive udskrevet en fejlmeddelelse der fortæller brugeren at kode er forkert. Hvis det indtastede data derimod stemmer overens med password_ så vil login vinduet lukkes og et objekt af CameraFeed klassen vil oprettes og skaber derved et nyt vindue. For yderligere information henvises her til projektdokumentationen.

11.2.2. CameraFeed

Dette er klassen der indeholder kamerabillederne fra systemets kamera og yderligere indeholder hovedfunktionaliteten fra *brugerinterfacet*. CameraFeed klassen indeholder fem knapper som alle har en underliggende funktion: Aktiver, Deaktiver, Genlad, Advarsel og Log ud.

Aktiver/Deaktiver sørger for at henholdsvis aktivere og deaktivere muligheden for inputs fra systemets PIR sensor der ellers aktiverer og medfører at alarmen vil lyde. Dette bliver yderligere meddelt på *brugerinterfacets* skærm og også noteret som en relevant handling i log filen.

Genlad knappen sørger for at oprette et nyt objekt af genlad klassen og initiere dette, hvilket åbner et genlad dialog vindue.

Advarsel knappen vil gennem UART klassen sende en besked ud til platformhøjtaleren og herfra sende en advarselstone, for at afskrække rovdyr eller lignende og på den måde mindske chancerne for at brugeren behøver at affyrer våbnet. Dette vil også blive meddelt som en statusbesked, på *brugerinterfacet*, og yderligere noteret i log filen.

Log ud knappen vil først oprette et objekt af login klassen, som åbner et nyt Login vindue. Herefter nedlægge CameraFeed objektet, hvilket lukker CameraFeed vinduet. Yderligere information om klassen og dennes funktioner kan findes i projektdokumentationen.

11.2.3. Genlad

Denne klasse bruges til at håndtere antallet af skud der er blevet indsat i våbnet og bliver oprettet når der bliver trykket på ”Genlad” knappen fra CameraFeed vinduet. Genlad klassen er den eneste klasse i *brugerinterfacet* der er af QDialog typen da dette muligøre at vinduet kan kræve input, før du kan gå videre i systemet, det kræver altså at man enten indtaster det antal skud man har indsat i våbnet eller at man anullerer handlingen.

Derudover var det nødvendigt at skabe restriktioner i indtastningsfeltet for at forhindre mulige fejl. Derfor blev det vedtaget at der kun kunne indtastes tal mellem 0 og 100, disse to inkluderede. For yderligere information om genlad klassen og dennes funktioner henvises der til projektdokumentationen.

11.3. Backend funktionalitet [AEL, JDA, DT]

Dette afsnit indeholder alt det software der ikke bliver visualiseret på *brugerinterfacet.*

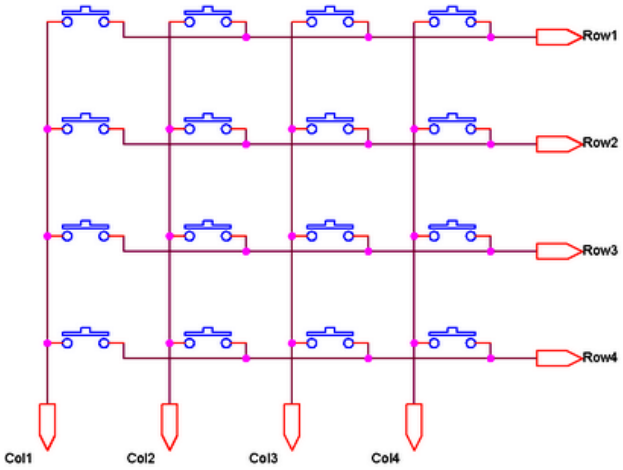
11.3.1. Log [AEL]

Denne klasse har til formål at holde en log over, for systemet, interessante handlinger. Det blev vurderet at de relevante handlinger var: Affyret skud, system aktiveret, system deaktiveret, advarsel givet og fejl i uart kommunikation.

Log klassen bliver oprettet sammen med CameraFeed klassen men ligger som en backend funktionalitet og noterer alle relevante handlinger i filen ”Log.txt”. Log klassen sørger desuden selv for at oprette og indsætte et tidstempel i lokal tid samt dato, for at man på den måde senere kan bruge loggen til at analysere et eventuelt handlingsforløb.

11.3.2. Design af matrixkeyboard [JDA]

Brugeren skal interagere med systemet gennem et matrixkeyboard. Matrixkeyboardet har 16 trykknapper som gennem 10 pins er forbundet til det Indlejrede Linux Systems GPIO pins[14], hvor to af dem er hhv. VCC og ground. Af hensyn til det Indlejrede Linux Systems spændingstolerance, bruges 3.3V som VCC. Matrixkeyboardets trykknapper repræsenterer tallene 0 til 9 samt karaktererne A, B, C , D, # og *. Matrixkeyboardets knapper er forbundet som set på Figur 13.

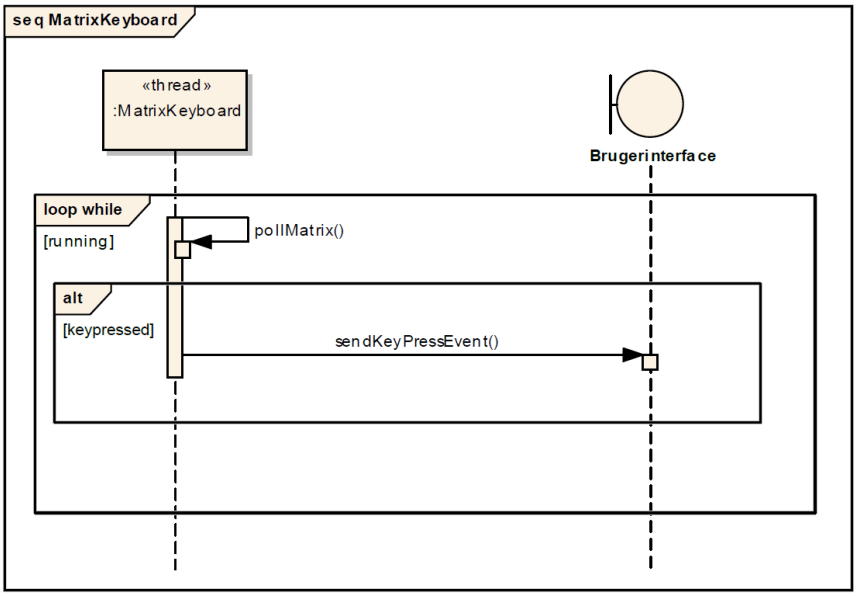


Figur 13 - Forbindelse mellem matrixkeyboardets knapper

Måden hvorpå der ønskes aflæsning fra matrixkeyboardet, er ved at skrive et logisk LAVT signal ind på tastaturets rækker, hvorefter der læses på kolonnerne, hvilken række den er kortsluttet med.

Til håndtering af kommunikationen mellem brugeren og systemet, er der udarbejdet et klasse-design til matrixkeyboardet. Klassen skal indeholde funktionalitet til aflæsning fra tastaturet, behandle dataen og videresende den, i et format der er læseligt for programmet der bruger den. Matrixkeyboardets funktionalitet skal bruges af *brugerinterfacet*[15]. Matrixkeyboard klassen vil indeholde en funktion der aktivt ”poller” eller ”checker” om der er trykket på en knap. Selve Matrixkeyboard klassen implementeres som en tråd, så den ikke blokerer hovedprogrammet. I forbindelse med designet af denne klasse, er der udarbejdet et sekvensdiagram, se Figur 14. Sekvensdiagrammet illustrerer matrixkeyboardets kernefunktionalitet, **poll** funktionen, samt videreformidling af data til *brugerinterfacet*.

[14] General Purpose Input/Output
[15] Se dokumentation om GUI (afsnit 11.2) og Matrixkeyboard design (afsnit 11.3.2)



Figur 14 - Sekvensdiagram for klassen MatrixKeyboard

I afsnittet omkring Design af matrixkeyboard i dokumentationen, findes dybdegående beskrivelse af matrixkeyboardets design, både af selve klassen og implementeringen i *brugerinterfacet*.

11.3.3. Implementering af matrixkeyboard [JDA]

Matrixkeyboardets implementering tager udgangspunkt i designdokumentet. Det har dog været nødvendigt at tilføje funktionalitet for at få aflæsning til at fungere efter hensigten. I forhold til det første udkast til algoritmen for aflæsning, må det siges at klassen har ændret sig en del. Af tilføjet funktionalitet, kan der blandt andet nævnes: Sikkerhed for at der ikke aflæses flere gange pr. tryk på en knap, og at den samme knap kan trykkes igen, efter den er sluppet.

For at kunne bruge det Indlejrede Linux Systems GPIO pins bruges det dertil indrettede bibliotek, wiringPi[16]. Ved hjælp af grundlæggende funktionalitet i dette bibliotek, kan der skrives til og læses fra det Indlejrede Linux Systems GPIO pins. I forbindelse med implementeringen er der desuden også udarbejdet et aktivitetsdiagram, der beskriver klassens **poll** funktion. Detaljeret beskrivelse af GPIO pins og implementeringen af poll funktionen er at finde i projektdokumentationen under implementeringen af Matrixkeyboard.

11.3.4. Design af protokol [JDA]

Systemet benytter seriel kommunikation over UART, til at sende kommandoer fra det Indlejrede Linux System til PSoC4. Til dette er der designet en protokol, som vil kunne bruges ”system-wide”, af de klasser,

[16] Se projektdokumentationen, afsnit 11.3.4.2

som har til opgave at sende information til PSoC4. Protokollen er udformet således at der pr. kommando, sendes 4 karakterer: en *start-byte,* som altid er '1', en kommando-karakter der bestemmer opgaven der skal udføres, en *option* karakter, der giver PSoC4 mulighed for at udføre en opgave med en indstilling (som fx hastighedsbestemmelse af en kommando, der henvender sig til motorerne). Den sidste karakter i karaktersekvensen er en checksum, af kommando-karakteren og option-karakterens XOR'ede værdi. Der ønskes implementeret en klasse der kan generere denne checksum, samt sammensætte de fire karakterer til en streng, der kan sendes videre til UARTQueue klassen, som vil blive forklaret mere om, i næste afsnit. Kommunikationsprotokollens overordnede design gør, at klassen fungerer som en "black-box", hvor output og input kendes, og som kan bruges over alt i hovedprogrammet. For yderligere information om protokolklassen, henvises til projektdokumentationen[17].

11.3.5. Implementering af protokol [JDA]

Protokolklassen er implementeret ud fra designdokumentet. Klassen er en mindre klasse med få metoder og attributter, og implementering afviger derfor ikke meget fra det ønskede design. Af relevante features i protokol-klassen, kan der nævnes den måde karaktersekvensen oprettes på. I stedet for at skulle huske den specifikke char for den kommando der ønskes sendt ud, kan der med fordel bruges en *enum* værdi, som er defineret i Protocol klassen. For hver kommando, er der en tilsvarende *enum* værdi, med et mere sigende navn, der er nemmere at huske (eksempelvis CMD_LASEROFF for at affyre skud, i stedet for 'K'). Denne tilgang er brugt idet at koden bliver lettere at læse og vedligeholde. En sidste ting er, at protokollens kommando-chars kan ændres, uden at programmet går i stykker.

11.3.6. Design og implementering af UART [DT]

UART klassen indeholder baggrunds-funktionalitet, og er altså ikke som sådan at se i *brugerinterfacet* for brugeren. UART klassen bliver dog brugt, hver gang brugeren bevæger Joysticket, eller benytter anden funktionalitet der kræver kommandoer sendt til PSoC4'en.

UART klassen gør brug af en besked-kø kaldet UARTQueue. Denne vil indeholde alle de beskeder (kommandoer) der skal sendes over UART til PSoC4. UART køen er beskyttet af en mutex[18], for at være sikker på, at selve kø-strukturen[19] der ligger i UARTQueue klassen kun bliver modificeret af én instans af gangen.

UART klassen vil blive implementeret som en tråd, som blot står og læser fra UARTQueue klassen. I al den tid, hvor der ikke er beskeder i køen vil tråden blokere. Denne funktionalitet ville ikke være mulig, hvis UART

[17] Se afsnit 11.3.3 i projektdokumentationen.
[18] Mutually exclusive.
[19] Implementeret som en std::queue.

klassen havde kørt i hovedprogrammet, da den derved ville blokere hovedprogrammet og hele systemet ville gå i stå.

UART klassen bliver aldrig tilgået fra udefrakommende instanser. Al kommunikation til UART klassen *SKAL* foregå gennem UARTQueue klassen.

For yderligere information om UART- og UARTQueue klasserne, henvises til dokumentationen, afnit 11.3.6 – 11.3.8.

11.3.7. Design og implementering af Joystick [DT]

Joystick klassen[20] står for at læse værdier fra ADC'en[21], evaluere disse og sende dem til UARTQueue klassen, hvis nogle bestemte krav er opfyldt. For selve joysticket er der nogle bestemte intervaller, hvor der bliver udsendt en "bevæg motor" kommando, når brugeren bevæger joysticket tilstrækkeligt. For triggeren bliver der sendt en "skyd" kommando, når triggeren trykkes ned. For PIR-sensoren bliver der *ikke* sendt kommando til PSoC4, men en højtaler med en alarm-lyd bliver startet i stedet, når PIR-sensoren opfanger bevægelse.

Joysticket er ligesom UART klassen en tråd. Denne tråd læser hele tiden fra de fire moduler koblet til ADC'en (joystick horisontal og vertikal, trigger og PIR-sensor). Hver gang der er aflæst, evalueres den aflæste data, og kommandoer sendes til PSoC4, når de førnævnte krav er opfyldt. Dette fortsætter indtil tråden bliver stoppet.

Til kommunikationen mellem Joystick og PSoC4 bruges den protokol, der er defineret til dette formål. Selve joysticket har to akser. En X- og en Y-akse. Begge akser kan returnere værdier mellem 0 og 1023, hvor 512 returneres, når joysticket er i "neutral" tilstand. Neutral tilstand er når joysticket ikke er flyttet, og altså står i midten af max og min positionerne.

I designfasen blev det besluttet, at joysticket skulle have en del intervaller, hvor der skulle sendes kommandoer til PSoC4. I den reelle implementering er der 2 intervaller (3 med neutral position). Interval 1 er hastighed 1, og interval 2 er hastighed 2. Dette skyldes, at joysticket er ekstremt fintfølende, så det er svært at flytte det fra eksempelvis 620 til 700. Endvidere blev det opdaget under test, at joysticket returnerer 1023 (max værdi), allerede når det er halvvejs mod dets yderposition. Dette betød at intervallet fra 512 til 1023 blev løbet igennem utrolig hurtigt, og det derfor var urealistisk med så mange forskellige

[20] Implementeret som JoystickThread.
[21] Analog to digital converter.

hastighedsintervaller, indenfor så lille et fysisk område. Det samme var tilfældet for den modsatte retning, altså 512 til 0.

For yderligere information angående Joystickets design og implementeret, henvises der til projektdokumentationen, afsnit 11.3.9 og 11.3.10.

12. Resultater og diskussion [AEL og ATT]

På tidligere semestre er der blevet tildelt projekter som var defineret i forvejen. Til dette semesters projekt er der dog blevet givet noget friere hænder mht. valg af emne, hvilket betyder at man kunne lave sine egne krav til dette projekt, som det færdige produkt skulle opfylde.

Der var til dette semesters projekt dog opsat nogle faste krav der skulle opnås og derudover blev der fastsat nogle yderligere krav der blev udviklet med hjælp fra erfaringer fra dette semesters kurser samt tidligere erfaringer.

Resultatet for projektet er en prototype af en Home Defense Turret, der bortset fra få skønheds fejl faktisk ville kunne bruges til første serie i en produktrække. Alle krav fra projektets kravspecifikation blev opfyldt med stor tilfredshed og det kunne konstateres at de fastsatte ”Must” og ”Should” faktorer i projektets MoSCoW alle endte med at blive en realitet. Derudover blev alle accepttestens punkter opfyldt og prototypen kunne dermed officielt kaldes en succes.

Hovedideen var at dele projektet op i to dele: Controller delen og Platform delen. Ideen er at en bruger kan kommunikere med en platform som ligger et andet sted og stadig være i stand til at styre denne.

Controller delen består af en skærm som viser et kamerafeed og et *brugerinterface* samt er udstyret med et joystick og en trigger. Det lykkedes forholdsvis smertefrit at få controlleren til at kommunikere med platformen via en hjemmelavet protokol. Dog var der lidt problemer med at få PSoC4 til at modtage fra UART’en da det viste sig, at en enkelt funktion manglede. Dette blev dog hurtigt løst, og dette kan der læses mere om i projektdokumentationen.

Brugerinterfacet blev lavet i Qt Creator og dette program blev netop valgt da det skulle være enkelt at overføre til andre enheder. Dette var dog på ingen måde tilfældet fra start, Qt i sig selv virkede som lovet, men i og med at flere biblioteker var tilføjet og skulle linkes, opstod der mange problemer. Problemerne løste sig dog til sidst og dette kan også læses mere om i projektets systemdokumentation.

Joysticket og triggeren fungerede efter hensigten men implementeringen af joysticket var fra starten tænkt anderledes, men i og med kredsløbet ikke reagerede på interrupts hertil, blev det implementeret med et poll, hvilket bruger en del CPU kraft, men det blev vurderet at dette ikke var noget der ville have dybere indflydelse på det resterende system.

Platform delen indeholder en motorkontrolleret våbenholder som kan styres af en PSoC4. Hardwaren fungerede som den skulle for selve motorstyringen, men der var en del kurre på tråden mht. implementeringen af dødswitchene. Problemet var at våbenholderen ikke altid ville bevæge sig kontinuerligt

og begyndte at ”hakke” i sine bevægelser på vilkårlige tidspunkter. Dette viste sig at skyldes nogle interne funktioner i PSoC4 og hvordan dens indstillinger var sat op. Alt dette kan dog læses mere om i projektdokumentationen.

13. Udviklingsværktøjer [AEL]

Dette er en liste over udviklingsværktøjer, der er brugt til udarbejdelsen af dette semesterprojekt.

Qt Creator

Dette program har været en stor del af den visuelle udvikling til projektet. Gruppen blev hurtigt enige om at det var smart at bruge da Qt er utroligt kompatibelt og kan bruges på de fleste platforme med meget få ændringer i koden. IDE[22]'en er generelt let at lære men som så meget andet af sin slags viste den sig svær at mestre.

Microsoft Visual Studio 2013 Ultimate

Visual Studio har været brugt til at lave store dele af projektet, kun med undtagelse af de elementer der skulle vises på *brugerinterfacet*. Derudover har compileren fra Visual Studio været brugt som compiler for Qt Creator da denne ikke havde sin egen.

Microsoft Visio

Visio har været brugt til de tidlige systemdiagrammer. Enterprise Architect som ellers har været brugt, kunne ikke håndterer BDD'er og IBD'er lige så godt som Visio og det blev derfor besluttet at disse skulle laves i Visio og resten i Enterprise Architect.

Enterprise Architect

Dette program har været det program der hovedsageligt er brugt til systemdiagrammer og lignende. Der var en bred enighed i gruppen om at Enterprise Architect var bedst at arbejde med og skabte de flotteste resultater, dog med undtagelse af IBD'er som ingen kunne få sat ordentligt op. Det blev derfor besluttet at IBD'er og BDD'er skulle laves i Microsoft Visio i stedet.

Multisim 13

Multisim har været brugt som værktøj til at simulerere de fleste kredsløb i projektet. Multisim laver en simulering af komponenterne i en ideel verden, og dette skaber gode muligheder for beregninger og videreudvikling.

Maple

[22] Integrated Development Environment

Maple er et matematik program og blev brugt specielt til udregning af modstandsstørrelser og effektudregning.

Microsoft Word

Word blev tidligt I processen valgt som det primære teksthåndteringsværktøj. Dette virkede som det mest naturlige valg da alle havde god erfaring med dette program.

Fritzing

Fritzing er et program der er godt til at lave kredsløbstegninger, her kan komponenter som sensorer og ADC'er findes, hvilket gjorde det ideelt for dette projekt og det blev derfor brugt til at lave kredsløbstegningen over det Indlejrede Linux System og dens forbindelser til andre komponenter.

Multimeter

Dette er et måleværktøj der primært er blevet brugt til at teste outputtet af de elektriske strømme og spændinger i systemet.

PSoC Creator

PSoC Creator er et IDE specielt designet til at håndtere en PSoC. Dette har derfor været brugt til at udvikle funktionaliteten til projektets PSoC4, og derefter kompilere og programmere det til den fysiske PSoC4.

Tera Term

Dette er et program der kan aflæse char værdier af en UART forbindelse og har derfor været brugt i forbindelse med test af UART kommunikationen imellem PSoC4'eren og det Indlejrede Linux System.

PuTTY

PuTTY er en SSH klient, som i projektet bruges til at oprette forbindelse til det Indlejrede Linux System. PuTTY har gjort det lette at sende filer til og fra enheden, specielt mht. krydskompilering af software.

Windows Command Prompt

I forbindelse med krydskompilering fra Windows til det Indlejrede Linux System, bruges Windows Command Prompt til at kalde Qt's qmake program.

Qt Cross tool

Qt cross tool bruges i forbindelse med krydskompilering af software til den Indlejrede Linux Platform. Dette værktøj hjælper med opsætning af et compiler-toolchain på host.

Silicon Labs CP210x

Dette er en UART til USB bro og har været brugt til test af UART kommunikation imellem PSoC4 og det Indlejrede Linux System.

Digilent Waveforms/Analog Discovery

Dette er blevet brugt som oscilloskop til test af hardware, efter det blev realiseret og senere til test af hardwarens funktionalitet, efter de forskellige elementer er blevet implementerede.

Fumlebræt

Fumlebrættet har fungeret som en testplatform for den fysiske hardware og været mellemled mellem det tænkte system og det endelige fysiske system.

14. Opnåede erfaringer [LRA]

Ved arbejdet på dette projekt har vi arbejdet med på forskellige områder og har fået mange faglige erfaringer. Da gruppen har været opdelt i software og hardware, har erfaringerne derfor ikke været de samme.

I software har de lært at benytte sig af en alternativ plan, når den første viser sig at være alt for omfattende. Her viste Microsoft Kinect sig at være en større opgave end først antaget. Se mere om dette i afsnit 11.1. Samtidig har software gruppen beskæftiget sig med Qt frameworket, og derved pådraget sig viden herom.

Hardware gruppen erfarede at man ikke kan komme udenom at skulle programmere, og at hver gang man tilføjer mere hardware så kommer der også hurtigt meget ekstra software. Og derfor har hardware gruppen været nødt til at påtage sig nogle programmerings opgaver. Disse var hovedsageligt grænseflade baseret.

Hardware har fx lært at man bør slukke for sit kredsløb før man laver nogle ændringer, i tilfælde af uforudsete fejl som en kortslutning, eller at man ikke skal presse komponenterne til deres yderste. Dette gav sig til kende da modstandende der blev brugt begyndte at ryge efter at systemet havde været tændt i nogle minutter.

15. Fremtidigt arbejde [ATT]

HDT'ens hovedfunktionalitet er at beskytte et område fra fjendtlige dyr, så fremtidigt arbejde vil være at gøre Platformen vandtæt, i og med den vil stå udenfor. Andet fremtidigt arbejde kunne være at tilføje en Kinect sensor til HDT'en som kan registrere et mål og selv sigte efter det. Idéen er da at brugeren kan vælge et "Fuldautomatisk" tilstand I menuen. På denne måde er brugeren ikke nødt til at sigte efter sit mål manuelt. HDT'en kan således være funktionel mens brugeren ikke er til stede. For at sikre at der ved "Fuldautomatisk" tilstand, ikke skydes på uskyldige ofre skal sensoren programmeres således at HDT'en kan registrere hvilken slags dyreart den skyder mod. HDT'en vil derfor ikke kunne skyde mod en person. Andet som kan tilføjes til HDT'en kunne være et "Fuldautomatisk safe mode". Denne funktion vil aktivere Kinecten så den kan registrere et offer, men i stedet for at sigte og skyde mod offeret, vil HDT'en finde ud af hvilken slags dyreart der registreres. Derefter vil den give en advarselslyd med det formål at skræmme dyret bort. Lyden der afspilles vil da være tilpasset den specifikke dyreart, som registreres. Hvis den registrerer en person vil den give en advarsel om at det er et privat område. Andet fremtidligt arbejde er at opgradere holderen for våbnet således at der kan påsættes et hvilket som helst våben, dog inden en størrelsesgrænse. Andet fremtidligt arbejde er at brugeren kan skifte adgangskode når der er logget ind på menuen. Joysticket, triggeren og keyboardet kan ændres til trådløse systemer, således at brugeren kan kontrollere systemet inden en fx 10m radius.

16. Individuelle konklusioner [Alle]

I følgende afsnit kan der findes individuelle konklusioner beskrevet af projektgruppens medlemmer.

16.1. Konklusion [JDA]

Da vores gruppe består af studerende fra både Elektro, IKT og stærkstrøm var det oplagt at dele gruppen op i to hold, hvor det ene stod for at udvikle hardware og det andet software. Min egen retning er IKT og det var derfor oplagt at min rolle i projektet, var udvikling af software. Mere specifikt, har jeg udarbejdet samt implementeret en protokol, samt udviklet software til matrixkeyboardet, og implementeret det i brugerinterfacet. Herudover har jeg været med til at udarbejde diagrammer og modeller til softwaren, samt systemet som helhed. Som den første gang på studiet, hvor vi som gruppe, selv får lov til at udforme en problemformulering, vil jeg mene at vi har truffet et godt valg. Ikke nok med at det alt i alt har været en lærerig proces, har det også været sjovt at udvikle vores produkt, hvilket jeg helt klart vil mene har givet et boost til samarbejdet for alle i gruppen. Jeg synes den endelige prototype er blevet god, og jeg mener at vi har fået inddraget mange aspekter fra undervisningen i kurserne på semestret, hvilket har været med til at ruste os til at bruge det vi har lært i praksis.

16.2. Konklusion [ATT]

Dette semester projekt har vi lov til at velje hvad vi vil gør. Vi har valgt at bygge op en automatisk turret som kan styres i gennem en brugergrænseflade. Dette idee synes jeg er virkelig intressant. Derfor har det været meget sjovere at arbjede med dette projekt på grund af brændende entusiasme. Alle i gruppen har haft den samme intressant og derfor bliver det en rigtige god humor i gruppen. Min del af projektet var at designe og implementere en motorstyring som jeg synes er meget intressant, fordi det er noget som jeg vil rigtige gerne arbejde med i fremtiden. Dette projekt har så udvid mit sind med at styre motorer med hjælp af microcontroller. Sammen med den erfaring ligger der også mere videnskab og hvorden det er at arbejde I en gruppe og hvorden man skal opføre sig i en gruppe. Dette gører man mere klar til at gå ud på arbejdsmarkeden. Men min konklution er at det har veret rigtigt rart at arbejde med noget som man selv har kommed op med og bygge det op fra starten. Når man ser den færdige produkt bliver man faktiskt lidt stolt.

16.3. Konklusion [DT]

I dette projekt har jeg hovedsageligt arbejdet med design og implementering, af joystick og UART klasserne. Derudover har jeg givet en hånd med de steder, hvor min hjælp var nødvendig. Jeg har fra tidligere semesterprojekter været i grupper, hvor jeg var træt af gruppens struktur og det arbejde der blev lavet. I

dette semesterprojekt har jeg været glad for projektgruppen, og jeg synes vi har arbejdet godt sammen, og fået lavet et godt projekt.

I forhold til egne personlige erfaringer, føler jeg mig bedre rustet til nye udfordringer. Der har været store software-mæssige problemer i løbet af projektforløbet. Sommetider har vi været ved at smide håndklædet i ringen. Til slut har vi dog fundet frem til en løsning, og står nu tilbage med en funktionel prototype, som har fået misundende blikke fra mange andre grupper. Det synes jeg er fedt. Det har givet mig en større vilje, til at kaste mig ud i nye og mere avancerede software projekter.

I løbet af semestrets kurser har vi lært om forskellige programmeringsmæssige principper, og nogle af disse er også anvendt i projektet, der hvor det giver mening at bruge pågældende princip. Jeg var dog ude for, at et af kursernes allersidste undervisningstime involverede teori der kunne have sparet mange timer i projektet, hvis det havde været viden jeg havde kendt noget tid før. Det ærgrede mig meget. Dog kan dette bruges til næste semesterprojekt. Jeg bliver aldrig helt gode venner med det faktum, at alt det teori vi skal bruge i semesterprojekterne læres imens vi laver projektet.

16.4. Konklusion [KB]

Dette semesterprojekt har været en spændende og lærerig proces. Vi har anvendt de erfaringer, som vi har lært i løbet af det forgangene semester, og derved brugt diverse redskaber, som vi er blevet undervist i til at udarbejde projektet Home Defense Turret. Der har været en ligelig fordeling mellem hardware og software, som gruppen har brugt i konstruktionen af dette projekt. Vi har konstrueret et produkt, der virker som det var ønskeligt.

Jeg hørte til projektets hardware-gruppe, hvor jeg har været med til at stå for kommunikationen mellem PIRsensor, Joystick, Trigger, ADC, ControllerHøjtaler med SOMO-II samt Raspberry Pi. Der har på den måde udover en hardwaremæssig forståelse, også været inddraget software til at beskrive kommunikationen imellem de enkelte dele. Disse færdigheder har vi lært igennem kurserne på 3. semester, samt den grundlæggende forståelse for SysML, som vi lærte i kurset ISE på 2. semester.

Jeg føler at gruppen har været engageret i projektet og været interesseret i de fælles samt individuelle områder/teams, hvilket er vigtigt for at få en større gruppe til at arbejde sammen om et fælles mål. Det har som udgangspunkt fungeret godt med at uddelegere hvem der har stået for at lave de enkelte projekts dele. Vi har i stor stil igen lært at være i et teamwork, hvor hver man bidrager med hver sin lidenskab, som

tilsammen munder ud i et færdigt produkt, gældende ligesom i en rigtig virksomhed, så denne arbejdsform giver os et godt kendskab om denne arbejdsform.

16.5. Konklusion [SF]

Tredje semesterprojekt har været en nogenlunde oplevelse. Personligt har jeg lært en del op SPIkommuniktation. Jeg kunne dog godt tænke mig at have arbejdet med noget mere hardware, men det var ikke helt til at se, hvor meget hardware, der ville være i projektet, dengang det blev besluttet. Ser jeg tilbage, ville jeg dog ændrer projektets ADC'blok til en PSOC4/5, da dette ville have lagt mere op af undervisningen, og der havde været mere i det end det, der er blevet lavet med hensyn til ADC og spikommunikation. Jeg føler dog alligevel, at jeg har bidraget med meget i gruppen, om ikke andet i den fase, hvor der skulle laves systemarkitektur og kravspecifikation. Selve produktet af projektet synes jeg dog har været rigtgit godt og sjovt at være med til lave!

16.6. Konklusion [AEL]

Personligt har jeg igennem hele projektet følt at vi som gruppe havde en god dynamik og at vi var gode til at supplerede hinandens kompetencer hvor det var relevant. Vores projekthåndtering har for mit vedkommende fungeret rigtig godt da "Scrum light" gav en masse frihed til at udvikle og gå i nogle forskellige retninger samtidigt med at vi alle i gruppen havde styr på hvor vi var, ikke mindst for vores egen men også for hinandens skyld. Mit arbejde har primært været at få styr på projektets brugerinterface, forbindelserne hertil og generelt alt det visuelle som brugeren kommer i kontakt med gennem vores system. Det har været en enormt lærerig process da jeg aldrig har arbejdet med GUI og lignende før, så der var en hel del ting der skulle tages fra bunden, og mange timer gik i brugerflader som endte med at blive skrottet til fordel for noget "pænere" eller mere funktionelt. Alt i alt har det dog været et rigtig godt projekt at arbejde på, i og med og jeg føler selv at jeg har bevæget mig rigtig langt både mht. projektstyring og programmering gennem dette semesters projekt.

16.7. Konklusion [LRA]

Jeg har lært at arbejde med grænseflader på PSoC4 og jeg har brugt meget tid på at arbejde med motorne og laseren og spændingsreguleringsprintet.

Motorne blev skaffede jeg ved at lave en aftale med en skrothandel, og jeg har derfor lært at man kan skaffe billige materialer ved at lede utraditionelle steder.

Jeg fik arbejdet tæt med grænserne for hvor meget strøm der kan trækkes igennem forskellige komponenter, og har lært at man altid bør lave udregninger inden man bare forsøger sig frem. Da man hurtigt kan spare

meget tid. Jeg har også udvidet min viden i simuleringsværktøjet multisim hvilket er godt et værktøj at bruge til at se om ens design virker inden man bygger det på fumlebræt.

Dette projekt har været sjovere at arbejde med end det på 2. semester da vi har haft meget større frihed til at vælge hvad vi ville arbejde med.

17. Fælles konklusion [Alle]

Vi har igennem dette projektforløb valgt vores eget emne. Dette har været en spændende og utrolig lærerig proces. Det har også været udfordrende. Især med hensyn til at få defineret projektet, da det har været en evig bekymring for nogle folk, om der især var hardware nok til fire mennesker. Dette skyldtes, at det var svært at få defineret projektet i starten af semesteret, da det på daværende tidspunkt ikke var overskueligt, hvor meget hardware og software der egentligt vil være i selve projektet. Det endte med, at der var en del mere software end hardware, og det var svært at smide mere hardware i projektet, uden at indsætte mere software.

Det var godt, at gruppen blev delt op studerene fra både Elektro, Stærkstrøm og IKT, da dette gav et godt indblik i, hvordan det er at arbejde tværfagligt i et udviklingsprojekt ude i erhvervslivet. Arbejdsprocessen for gruppen har været udmærket, især i starten af projektet, hvor vi fik uddelt folk fra de forskellige ingeniørlinjer på hver deres. Folk har generelt været meget engageret i projektet, og dette har været godt, da det har givet en god stemning internt i gruppen. Nogle i gruppen har været med nede på Systematic til et SCRUM-minikursus. Dette gav et indblik i, hvordan man kunne strukturere og implementere en gennemgang af processen til dette projekt, hvilket vi syntes har hjulpet os rigtigt meget. Ikke alle har overholdt samarbejdskontrakten, men på grund af kontrakten har været fælles lavet, har de individer som har overholdt den, fået den retfærdighed, som de fortjener.

Der var lidt problemer med at få sat hardware og software sammen. Dette gav lidt tidspres i sidste ende, men i skrivende stund skinner solen. Det havde været lidt smartere at sætte hardware og software sammen tidligere, end det blev gjort, men softwarefolkene havde rigeligt at se til, så dette var ikke muligt.

Vi er meget tilfredse med det endelige produkt, især fordi vi har fået opfyldt vores must og should punkter i MoSCoW kravene, og vi har desuden fået opfyldt de ikke-funktionelle krav.