

Systemarkitektur

RoboPlay

Semesterprojektgruppe: 10

Afleveringsdato: 19. december 2018

Version: 2.4.0

#	Stud.nr.	Navn	Initialer
1	201704441	Frank Andersen	FA
2	201705186	Kristian Bang Nielsen	KN
3	201509800	Christian Lundtoft Trebbien	CT
4	201707807	Frederik Munch-Hansen	FM
5	06923	Niels Pallisgaard Thøgersen	NT
6	201505470	Mads Skytte Nielsen	MN
7	201704714	Michael Møller-Hansen	MM
Vejleder: Martin Ansbjerg Kjær			

Tabel 1 - Medlemmer i gruppe 10

1 Indholdsfortegnelse

1 Indholdsfortegnelse	1
2 Versionshistorik	4
3 Ordbog	6
4 Indledning	8
5 Overordnet systemarkitektur	8
5.1 Blok-definitions-diagram	8
5.2 Systemsekvensdiagrammer for udvalgte use cases	10
5.2.1 Systemsekvensdiagram for UC1's hovedscenarie	10
5.2.2 Systemsekvensdiagram for UC1's extension 1	11
5.2.3 Systemsekvensdiagram for UC1's extension 2	13
5.2.4 Systemsekvensdiagram for UC2's hovedscenarie	13
5.2.5 Systemsekvensdiagram for UC6's hovedscenarie	14
5.2.6 Systemsekvensdiagram for UC6's extension 1	15
5.2.7 Systemsekvensdiagram for UC7's hovedscenarie	15
5.2.8 Systemsekvensdiagram for UC7's extension 1	16
6 Hardwarearkitektur	17
6.1 Internt blokdiagram	17
6.2 Systemets hardwareenheder	21
6.2.1 Mobiltelefon	21
6.2.2 RPi	21
6.2.3 I2CLevelConverter	22
6.2.4 PSoC	22
6.2.5 Servomotor 1-6 : Motor	23
6.2.6 Strømdistribution	23
6.2.7 Positionssensor	24

6.3 Strømdistribution	25
6.4 Strømsensor	28
6.5 Positionssensor	28
6.6 PSoC	31
6.6.1 Fordeling af ressourcer	32
7 Kommunikationsprotokoller	36
7.1 Protokol mellem MobilApp og RPiApp	36
7.1.1 TCP klient/server	36
7.1.2 Protokol for udførelse af bevægelser	37
7.1.3 Protokol for indstilling af robot til startposition	39
7.2 Protokol mellem RPiApp og PSoCApp	39
7.2.1 Opsætning	39
7.2.2 Besked-struktur for skrivning	39
7.2.2.1 Byte #1-6	39
7.2.2.2 Byte #7	40
7.2.3 Besked-struktur for læsning	41
7.2.3.1 Byte #1-6	41
7.2.3.2 Byte #7	41
8 Softwarearkitektur	41
8.1 MobilApp	42
8.1.1 Applikationsmodel for UC1 og UC7	44
8.1.1.1 Klassediagram	44
8.1.1.2 Sekvensdiagram for hovedscenariet	46
8.1.2 Applikationsmodel for UC2	47
8.1.3 Applikationsmodel for UC6	48
8.1.3.1 Klassediagram	48
8.1.3.2 Sekvensdiagram for hovedscenariet	49

8.2 RPiApp	50
8.2.1 Applikationsmodel for UC1	50
8.2.1.1 Klassediagram	50
8.2.1.2 Sekvensdiagram for hovedscenariet	52
8.2.2 Applikationsmodel for UC2	53
8.2.2.1 Klassediagram	53
8.2.2.2 Sekvensdiagram for hovedscenariet	54
8.2.3 Applikationsmodel for UC6	55
8.2.3.1 Klassediagram	55
8.2.3.2 Sekvensdiagram for hovedscenariet	56
8.2.4 Applikationsmodel for UC7	56
8.3 PSoCApp	56
8.3.1 Applikationsmodel for UC1	57
8.3.1.1 Klassediagram	57
8.3.1.2 Sekvensdiagram for hovedscenariet	59
8.3.1.3 Sekvensdiagram for extension 1(Step Motor)	61
8.3.1.4 Sekvensdiagram for extension 2	62
8.3.2 Applikationsmodel for UC2	63
8.3.2.1 Klassediagram	64
8.3.2.2 Sekvensdiagram	65
8.3.3 Applikationsmodel for UC6	67
8.3.4 Applikationsmodel for UC7	67

2 Versionshistorik

Vers	Dato (Y-M-D)	Navn	Ændringer
v0.1.0	2018-09-17	NT	Dokument oprettet
v0.2.0	2018-09-25	MM	Forslag til struktur for dokumentet lavet Indledning til dokumentet lavet Protokol mellem MobilApp og RPiApp lavet
v0.3.0	2018-10-03	MM & FA	Opdateret BDD og tilføjelse af systemsekvensdiagrammer.
v0.4.0	2018-10-04	NT	IBD og domænemodel tilføjet
v0.4.1	2018-10-05	NT	Opdateret domænemodel og IBD. Rettet til Dansk
v1.0.0	2018-10-08	Alle	Sidste ændringer - klar til review #1
v1.0.1	2018-10-14	MM og FA	Ændringer som følge af review foretaget
v1.1.0	2018-10-14	NT	Opdateret overordnet BDD og IBD, BDD og IBD for Strømdistribution og Positionssensor. Overordnet IBD tabel påbegyndt.
v1.1.1	2018-10-16	NT	Opdaterede BDD'er og IBD'er. De sidste tabeller dertil lavet, ordbog tilføjet.
v1.2.0	2018-10-17	MM, FA	Applikationsmodel for RPiApp lavet.
v1.3.0	2018-10-20	NT	Udvidede blok-beskrivelser hardware, tekst til Strømdistribution, afsnit Strømsensor.
v1.3.1	2018-10-22	NT	Omstruktureret
v1.3.2	2018-10-23	MN	Tekst i Overordnet.
v1.3.3	2018-10-24	NT	Midre ting i top-BDD, Overordnet IBD + tabel, Strømdistribution BDD og IBD samt tabeller, rette blokbeskrivelser. Ikke helt færdigt, men er godt ryk.
v1.3.4	2018-10-25	NT	Fortsat Strømdistribution. Positionssensor.
v1.3.5	2018-10-25	FA	Protokol mellem RPiApp og PSoCApp skrevet om.



v2.0.0	2018-10-29	MM	Rettelser som følge af gruppemøde den 291018 foretaget.
v2.0.1	2018-10-30	FA	Rettelser til protokolafsnit mellem RPiApp og PSoCApp som følge af gruppemøde den 291018
v2.1.0	2018-11-01	NT	Hardwarearkitektur: navne ændret, opdaterede diagrammer
v2.2.0	2018-11-04	MM	Sidste hånd lagt på MobilApp's applikationsmodel.
v2.3.0	2018-11-06	MM	Afsnit "5.1 Domænemodel" fjernet som besluttet på gruppemødet den 051118.
v2.3.1	2018-11-18	NT	BDD PSoC
v2.3.2	2018-11-20	NT	Opdateret IBD igen igen.
v2.4.0	2018-11-26	MM	Denne version er sendt til reviewgruppe i forbindelse med review #2.

Tabel 2 - Versionshistorik

3 Ordbog

I *tabel 3* herunder ses en oversigt over de begreber, der er benyttet i dette projekt samt forklaring af deres betydning. Denne ordbog kan benyttes som et opslagsværk under læsningen af dette dokument.

Ord	Betydning
RoboPlay	Det system, der udvikles, inklusive hard- og software
Robot	Den fysiske robotarm, inklusive motorer, sensore, mfl.
RPi	Raspberry Pi Zero W med Linux installeret
RPiApp	Software, der udvikles til RPi, inklusive eventuelle kernemoduler
PSoC	Cypress PSoC 5LP development board
PSoCApp	Software, der udvikles til PSoC
Mobiltelefon	Smartphone med Android operativsystem
MobilApp	Software, der udvikles til Mobiltelefon
Invers kinematisk bevægelse	En bevægelse mod et fastlagt punkt, hvor det skal beregnes, hvordan man kommer derhen.
Xamarin	En cross-platform teknologi udviklet af Microsoft. Xamarin giver mulighed for en høj grad af kodegenbrug mellem forskellige platforme såsom iOS og Android.
MVVM	Forkortelse for "Model-View-ViewModel. Den foretrukne arkitektur når Xamarin benyttes.
Yderpunkt	Det yderste punkt en given akse kan komme i.
Kontakt	Kontakten som tænder og slukker robotten
Servomotor	Servomotor af typen MG996R. Begreberne <i>motor</i> og <i>servomotor</i> ses som synonymme.
Hjul	<i>Encoder disc</i> til optocoupler.
Mellemrum	Hul i hjul, der tillader lys at passere igennem.
CRC	Cyclic Redundancy Check. En metode til at detektere fejl i data-transmission.
Motor	Servomotor af typen MG996R. Begreberne <i>motor</i> og <i>servomotor</i> ses som synonymme.

Tabel 3 - Ordbog

4 Indledning

I dette dokument beskrives systemarkitekturen for RoboPlay i detaljer. Det indledes med en helt overordnet beskrivelse af systemet under brug af et blokdefinitionsdiagram med tilhørende blok- og signalbeskrivelser. I den overordnede beskrivelse indgår også systemsekvensdiagrammer for relevante use cases. Med "relevante use cases" menes de use cases som er beskrevet med "fully dressed" use case beskrivelser, og som der er specificeret en accepttest for.

Dernæst følger et afsnit omhandlende hardwarearkitektur. Her ses en nærmere beskrivelse af de enkelte hardwareblokke og deres indbyrdes forbindelser.

Dette følges op af en beskrivelse af de protokoller, som systemets forskellige blokke benytter sig af for at kunne kommunikere med hinanden.

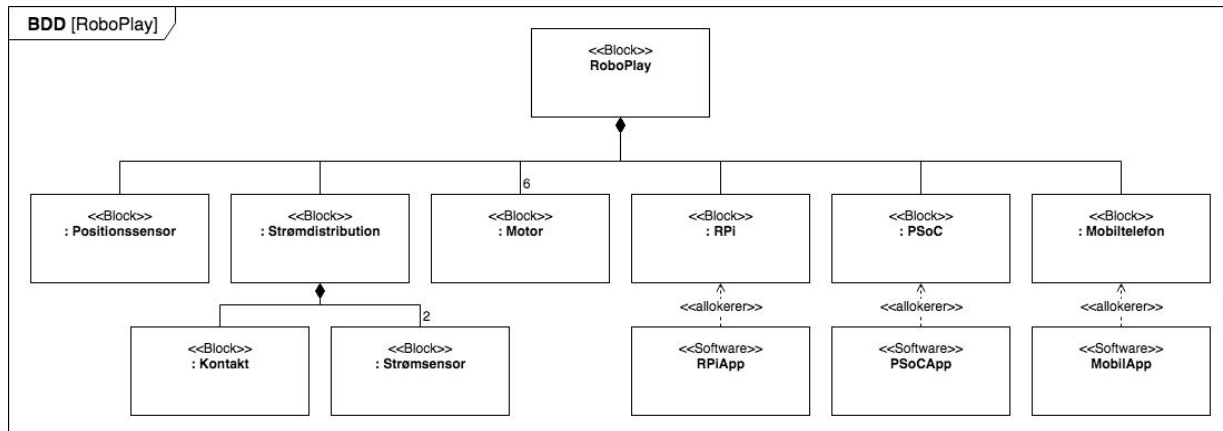
Afslutningsvis beskrives softwarearkitekturen under brug af applikationsmodeller, hvorfor der her gøres brug af sekvens-, klasse- og tilstandsdiagrammer.

5 Overordnet systemarkitektur

Dette afsnit beskriver systemarkitekturen på et overordnet plan.

5.1 Blok-definitions-diagram

På *figur 1* ses et BDD for RoboPlay og herpå fremgår også hvilken software, der allokeres til de forskellige blokke. Hvor der er tale om 1-1 relationer mellem blokke, er multiplicitet ikke anført.



Figur 1 - Overordnet BDD med softwareallokering.

I de følgende afsnit refereres der til de forskellige blokke og deres softwareapplikationer med de navne, som er angivet på BDD'et. De blokke, der kun er én af, vil blive refereret til ved deres type fremfor deres navn.

En kort beskrivelse af de enkelte blokke ses i *tabel 4*. Udvidede beskrivelser for hardwareblokke kan ses i afsnittet *Hardwarearkitektur* og tilsvarende i *Softwarearkitektur*.

Blok	Beskrivelse
RoboPlay	Det samlede system som består af de øvrige blokke.
Positionssensor	En sensor der dels sætter yderpunkter for første akse bevægelse og dels angiver aksens nuværende position.
Strømdistribution	Print til fordeling af forsyningsspændinger til RoboPlays enheder samt måling af strøm til motorer.
Kontakt	Systemets hovedafbryder. Alle enheder forsynes gennem denne.
Strømsensor	Måler den strøm, der løber igennem motorerne der styrer henholdsvis første og sjette akse.
Motor	Robotten har seks motorer, som styrer robottens bevægelige led.
primController : RPi	Robottens "hjerne" som er ansvarlig for at omsætte instruktioner, der er modtaget fra mobiltelefonen til bevægelser i robotten via PSoC'en.
motorstyring : PSoC	PSoC'en er tilsluttet robottens sensorer og styrer på baggrund af input fra disse samt kommandoer modtaget fra RPi'en robottens motorer.
Mobiltelefon	Brugeren benytter en applikation, der er installeret på mobiltelefonen, til

	at styre robotten.
RPiApp	Softwareapplikationen der afvikles på RPi'en.
PSoCApp	Softwareapplikationen der afvikles på PSoC'en.
MobilApp	Softwareapplikationen der afvikles på mobiltelefonen.

Tabel 4 - Blokbetegnelse for systemet.

Som det ses af *figur 1* og *tabel 4*, laves arkitekturen med udgangspunkt i at én positionssensor og to strømsensorer implementeres. Årsagen hertil findes i prioriteringen af de funktionelle og ikke-funktionelle krav, som ses i bilaget *Kravspecifikation*.

Positionssensoren placeres på robottens første akse, hvorved yderpunkterne for bevægelserne "Drej venstre" og "Drej højre" såvel som aksens aktuelle position kan detekteres.

Den ene strømsensor er placeret på den første akse, hvorved forhindringer, der står i vejen for udførelsen af bevægelserne "Drej venstre" og "Drej højre", kan detekteres.

Den anden strømsensor er placeret på den sjette akse, hvorved et objekt, som robottens fingre klemmer om, kan detekteres.

5.2 Systemsekvensdiagrammer for udvalgte use cases

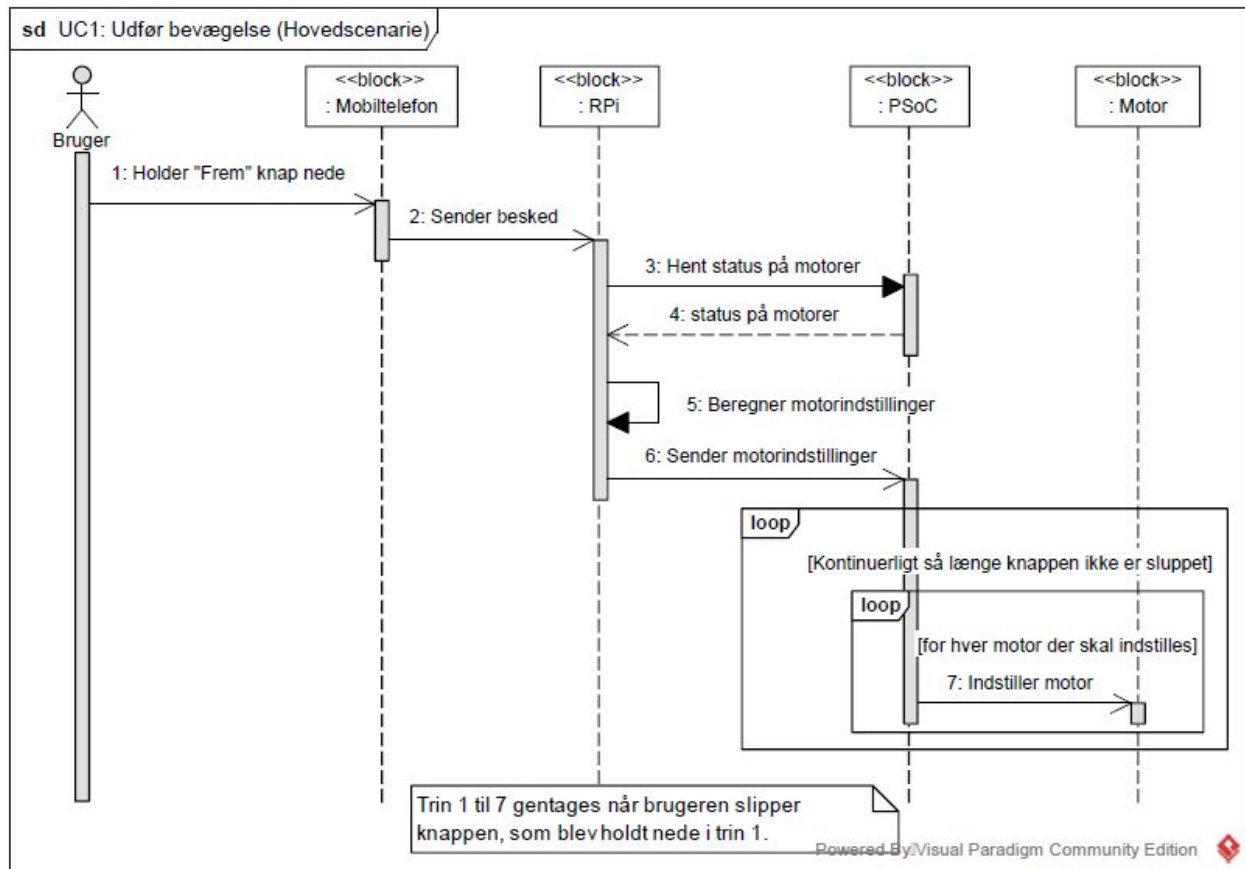
I de følgende underafsnit ses en række systemsekvensdiagrammer for de vigtigste use cases. Diagrammerne giver et overblik over hvordan de forskellige blokke på det overordnede BDD, der ses på *figur 1*, taler sammen.

5.2.1 Systemsekvensdiagram for UC1's hovedscenarie

Systemsekvensdiagrammet på *figur 2*, viser fremgangsmåden for samtlige bevægelser som brugeren kan få robotten til at udføre via mobiltelefonen - dog undtaget bevægelsen "Luk" som håndteres i UC7.

Uanset om en besked er kinematisk eller ej, så spørges PSoC'en altid om motorenes nuværende status og dermed også om positionen for robottens akser. Med baggrund i dette udregnes de nye motorindstillinger, som sendes til PSoC'en, der indstiller hver motor.

På *figur 2* startes UC1 med bevægelsen "Frem". Det at brugeren standser bevægelsen igen er ikke vist på diagrammet, men trin 1 til 7 gentages blot, hvor trin 1 i stedet ville have teksten: "Slipper 'Frem' knap".



Figur 2 - Systemsekvensdiagram for UC1's hovedscenarie.

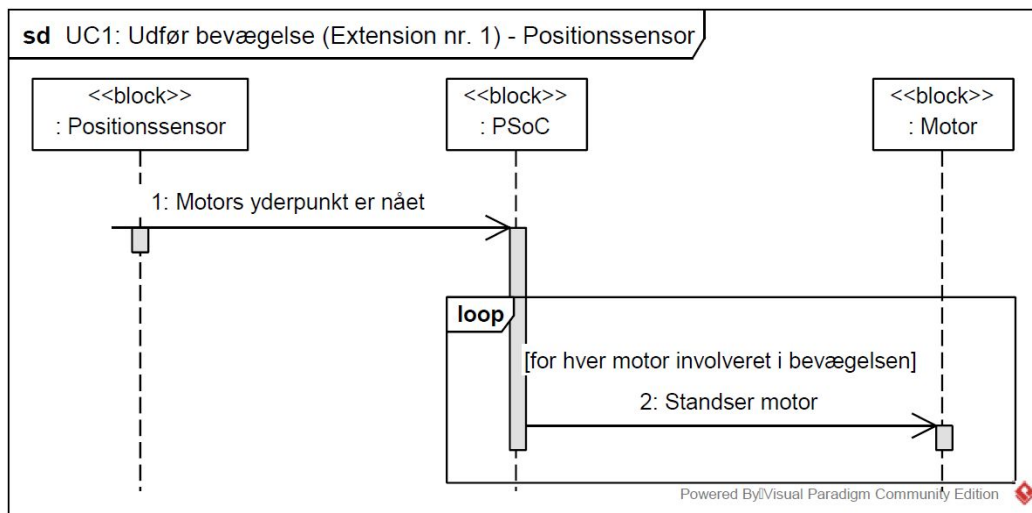
5.2.2 Systemsekvensdiagram for UC1's extension 1

Idet positionssensorer kun implementeres på første akse, så er det nødvendigt at definere de øvrige bevægelseres yderpunkter i software. Derfor er denne extensions vist på to forskellige diagrammer: Et hvor en positionssensor standser bevægelsen og et hvor bevægelsen standses i softwaren.

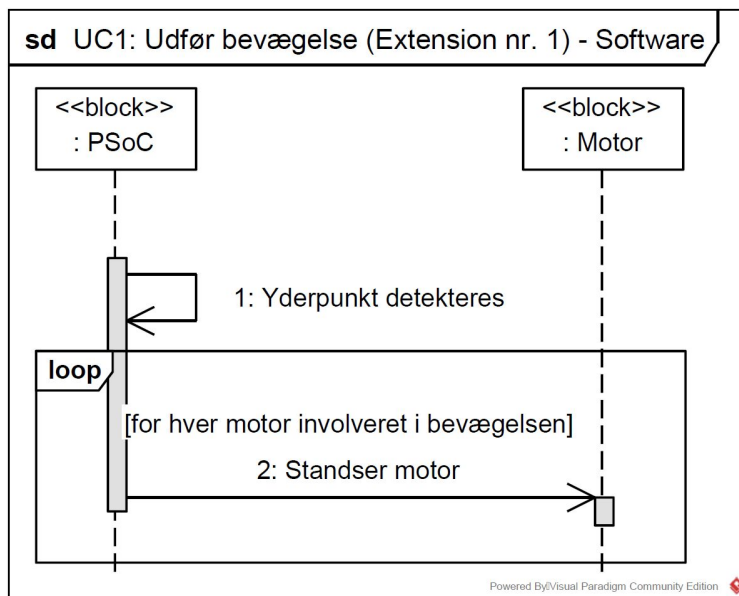
På *figur 3* ses situationen hvor en bevægelses yderpunkter detekteres af en positionssensor og på *figur 4* ses situationen hvor bevægelsen standses i software. Brugeren har i begge tilfælde

igangsat en bevægelse som vist på *figur 2*'s punkt 1-7. Inden brugeren når at slippe knappen igen når bevægelsen sit yderpunkt. Som konsekvens heraf standser systemet bevægelsen.

Er der tale om en kinematisk bevægelse, så er det nødvendigt at standse mere end én motor, hvilket medfører det på *figur 3* og *figur 4* viste loop.



Figur 3 - Systemsekvensdiagram for UC1's extension 1. En bevægelse er sat i gang i UC1's hovedscenarie, men inden brugeren når at standse bevægelsen selv, standser systemet bevægelsen, idet motoren, der driver bevægelsen, når sit yderpunkt.

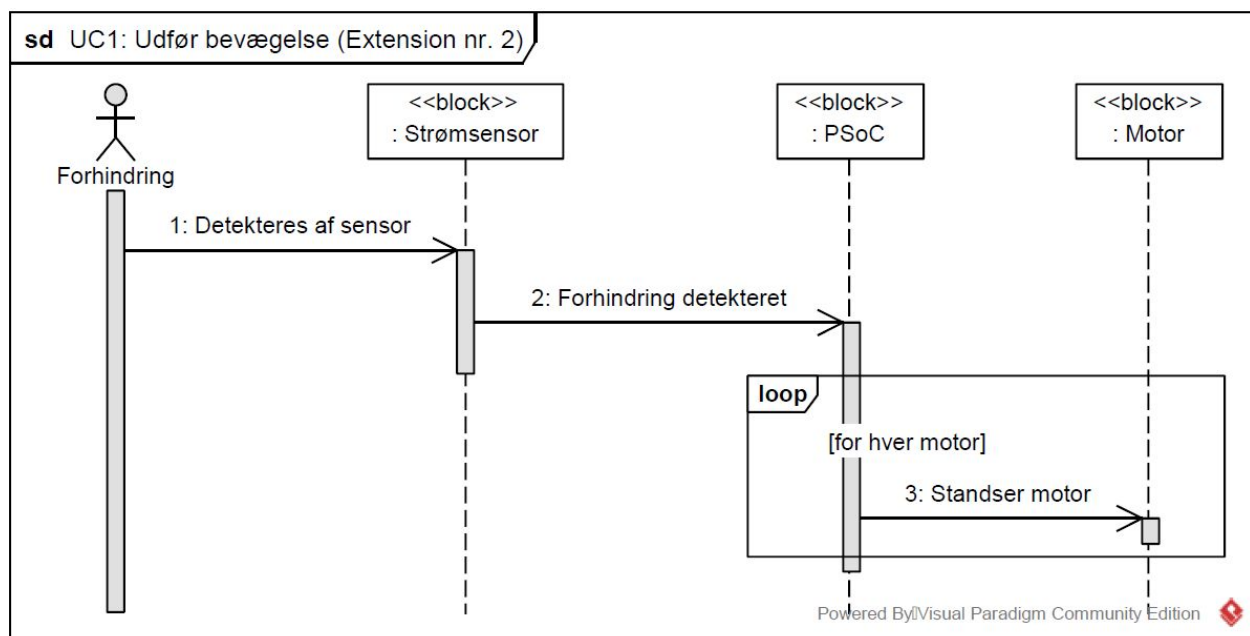


Figur 4 - Yderpunktet detekteres i software.

5.2.3 Systemsekvensdiagram for UC1's extension 2

I systemsekvensdiagrammet vist på *figur 5* ses situationen, hvor en bevægelse er påbegyndt, som vist på *figur 2*, men hvor bevægelsen ikke afbrydes af brugeren men af en forhindring, som står i vejen for bevægelsens videre udførelse. Et eksempel på en sådan forhindring kunne være en væg.

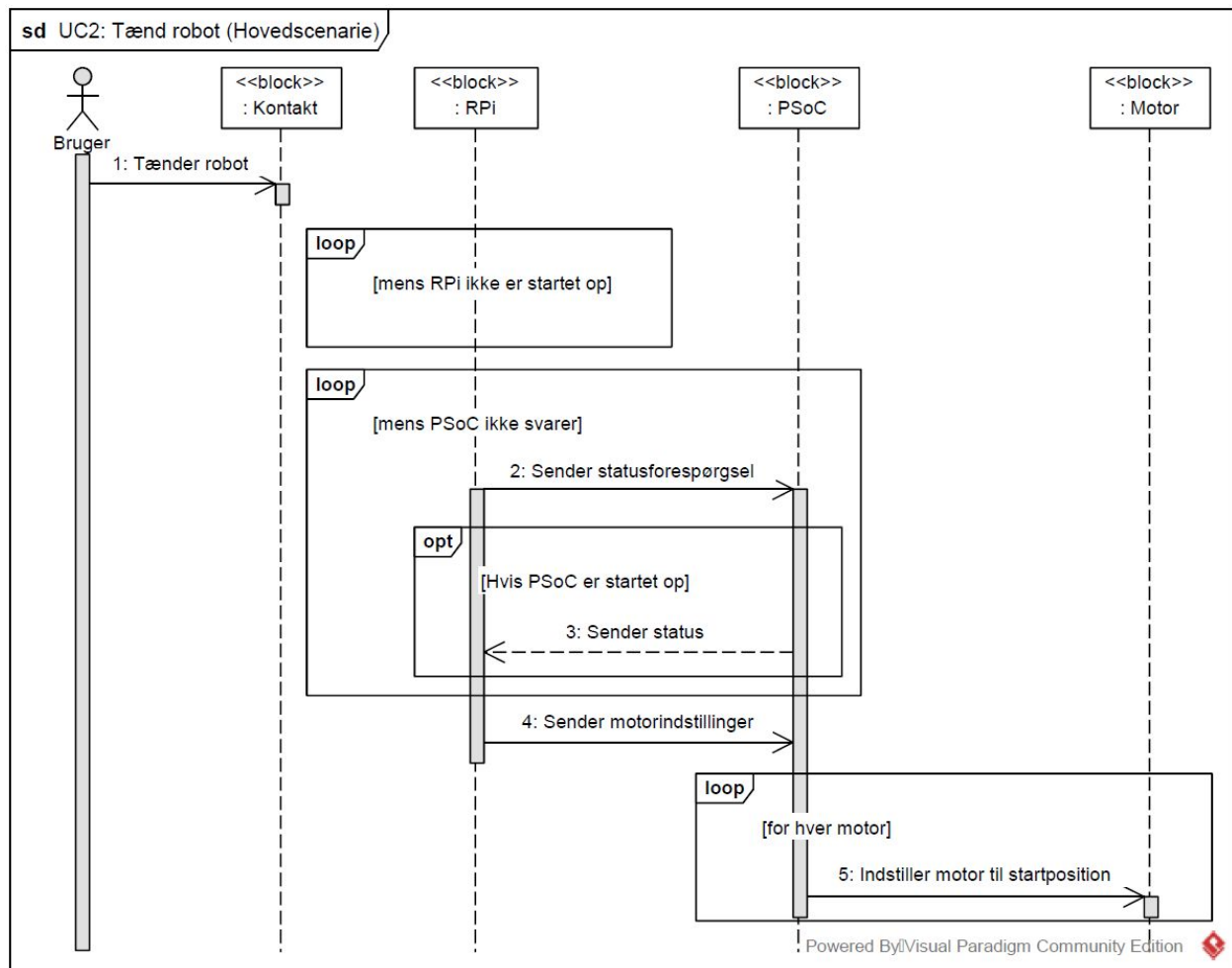
I denne extension standses alle bevægelser som en sikkerhedsforanstaltning, hvilket ikke er tilfældet i extension 1.



Figur 5 - Systemsekvensdiagram for UC1's extension 2, hvor en sensor detekterer en forhindring, hvorefter alle igangværende bevægelser standses. Denne extension er kun gældende for bevægelserne "Drej venstre" og "Drej højre".

5.2.4 Systemsekvensdiagram for UC2's hovedscenarie

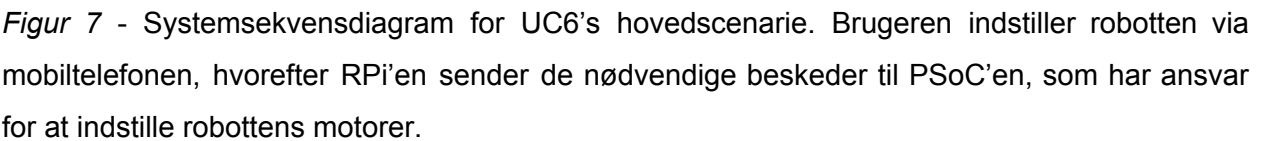
Situationen hvor brugeren tænder for robotten ved at påvirke en fysisk kontakt ses på *figur 6*. Først afventes det at RPi'en starter op. Derefter afventer RPi'en, at PSoC'en er opstartet. Når PSoC'en har tilkendegivet, at den er klar til at modtage instruktioner, sender RPi'en de motorindstillinger, der er nødvendige for at indstille robottens akser til deres startpositioner.



Figur 6 - Systemsekvensdiagram for UC2's hovedscenarie. Brugeren tænder robotten, hvorefter det afventes at RPi'en starter op. Derpå afventer RPi'en, at PSoC'en er klar til kommunikation og til sidst indstilles robottens akser til deres startpositioner.

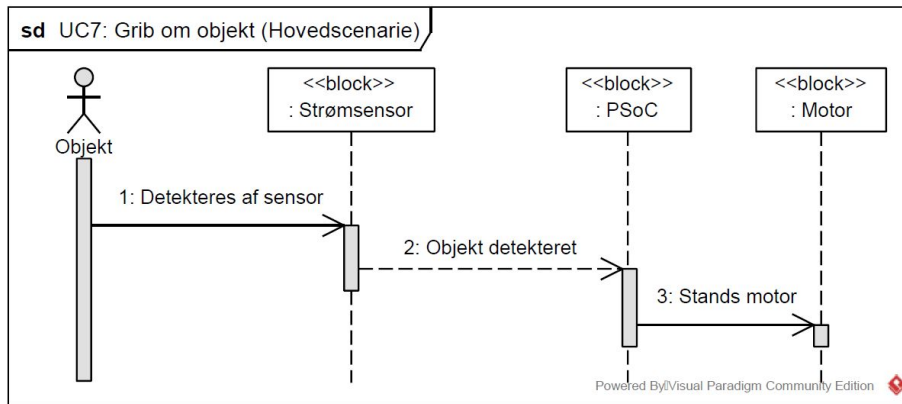
5.2.5 Systemsekvensdiagram for UC6's hovedscenarie

Systemsekvensdiagrammet for UC6's hovedscenarie, hvor brugeren sætter robottens akser til deres startpositioner via MobilApp'en, ses på figur 7.



Hvis en forhindring detekteres mens robotten er ved at indstille sine akser til deres startpositioner, som vist på *figur 7*, så standses alle bevægelser. Fremgangsmåden er identisk med UC1's extension 2, som ses illustreret på *figur 5*.

Systemsekvensdiagrammet for UC7's hovedscenarie, hvor robottens hånd lukkes i et forsøg på at gribe om et objekt, tager udgangspunkt i *figur 2*, hvor bevægelsen i trin 1 erstattes af bevægelsen "Luk". Efter sidste trin på *figur 2* indtræder *figur 8*, som vist nedenfor, hvor en sensor detekterer et objekt og motoren standser sin bevægelse.




Figur 8 - Systemsekvensdiagram for UC7's hovedscenarie. Brugeren lukker robottens hånd for at gribe om et objekt og denne bevægelse standses ved at hånden klemmer om et objekt.

5.2.8 Systemsekvensdiagram for UC7's extension 1

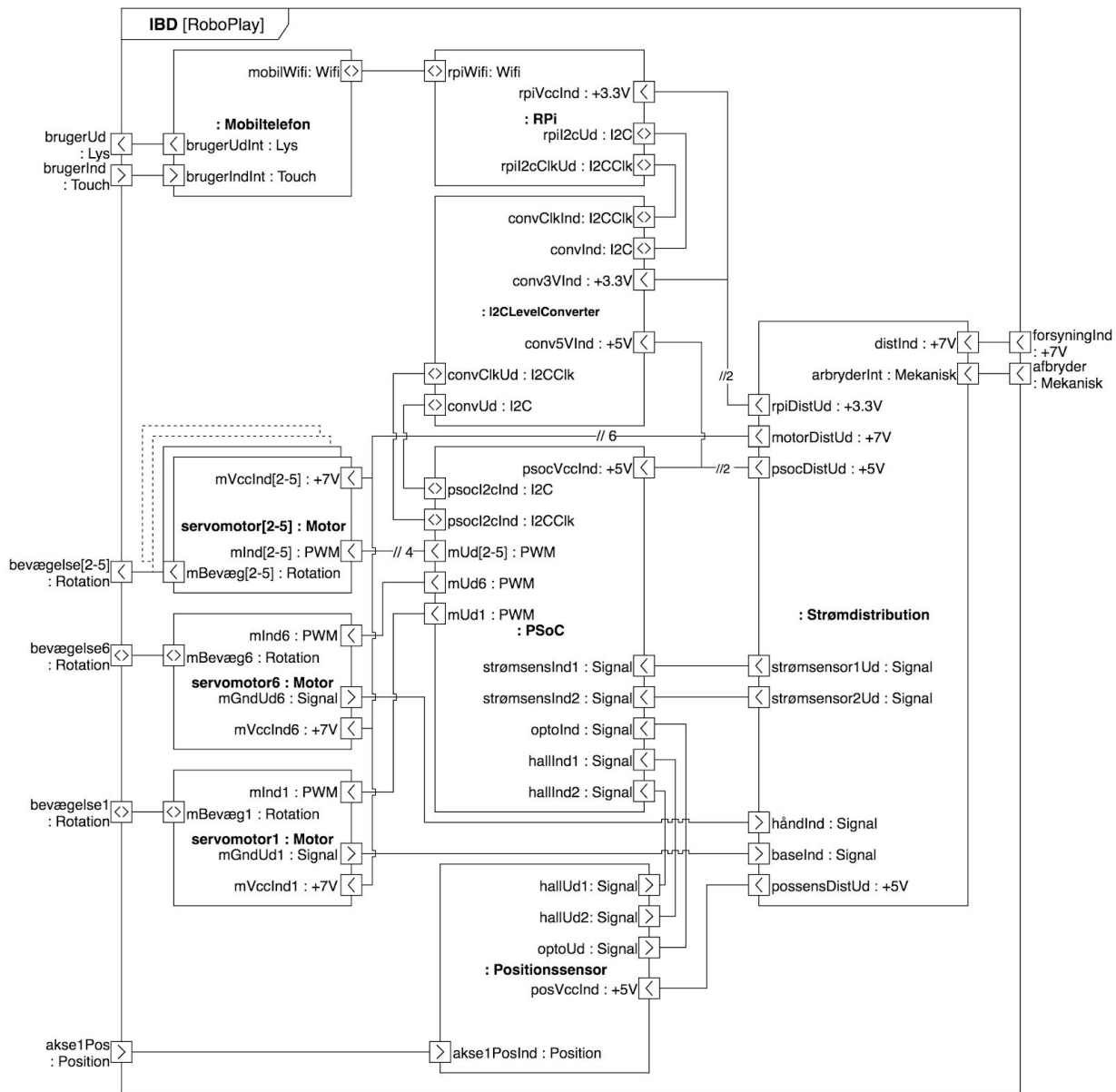
Fremgangsmåden for UC7's extension 1 er identisk med figur 2, hvor bevægelsen i trin 1 erstattes af at knappen "Luk" slippes, hvormed brugeren har afsluttet bevægelsen "Luk" inden robotten har grebet om objektet.

6 Hardwarearkitektur

Dette afsnit beskriver hardwarearkitekturen af RoboPlay. Som det fremgår af *figur 1*, er hardwaren opdelt i de primære blokke *Motor*, *Positionssensor* og *Strømdistribution*.  Det følgende vil disse blokkes indholdsblokke defineres og signalvejene imellem dem vil blive beskrevet.

6.1 Internt blokdiagram

Herefter, i *figur 9* ses et internt blokdiagram for RoboPlay. Stelforbindelser er underforståede og signalnavne er definerede i *tabel 5* nedenfor for at sikre læsbarhed af diagrammet. Som det ses, er *servomotor[2-5]* samlet i en blok for at forsimple diagrammet. Signalerne med navne, der ender med [2-5] skal forstås som fx *mVccInd2* til *mVccInd5*, *mUd2* til *mUd5* osv. Se *tabel 5* for detaljer.






Figur 9 - IBD for RoboPlay


På tabel 5 herunder ses en oversigt over de signaler, der indgår i figur 9.

Signalnavn	Funktion	Område	Port 1	Port 2
forsyning	Primær spændings-forsyning	+7V +/- 0.2V	forsyningInd	distInd
rpiVcc	Forsynings-spænding til RPi	+3.3V +/- 0.2V	rpiDistUd	rpiVccInd conv3VInd



psocVcc	Forsynings-spænding til <i>PSoC</i>	+5V +/- 0.2V	psocDistUd	psocVccInd conv5VInd
motorVcc	Forsynings-spænding til servomotorer	+7V +/- 0.2V	motorDistUd	mVccInd1 mVccInd2 mVccInd3 mVccInd4 mVccInd5 mVccInd6
possensVcc	Forsynings-spænding til <i>Positionssensor</i>	+5V +/- 0.2V	possensDistUd	posVccInd
håndGndSig	Stel for Motor 6 til måling af strøm	0-5V 	håndInd	mGndUd6
baseGndSig	Stel for Motor 1 til måling af strøm	0-5V	baseInd	mGndUd1
strømsensSig1	 alogt signal fra måling af strømstyrke 	0-5V	strømsensorUd1	strømsensInd1
strømsensSig2	Analogt signal fra måling af strømstyrke	0-5V	strømsensorUd2	strømsensInd2
i2cSigPre	Kontrolsignal til <i>PSoC</i>	0-3.3V	rpil2cCUd	convInd
i2cClkPre	I2C clock	0-3.3V	rpil2cClkUd	convClkInd
i2cSigPost	Kontrolsignal til <i>PSoC</i>	0-5V	convUd	psocI2cInd
i2cClkPost	Sc clock	0-5V	convClkUd	psocI2cClkInd
wifiSig	Trådløs netværks-forbindelse	Wifi	mobilWifi	rpiWifi
optoSig	Signal fra optocoupler	0-5V	optoUd	optoInd



hallSig1	Signal fra Hallsensor	0-5V	hallUd1	hallInd1
hallSig2	Signal fra Hallsensor	0-5V	hallUd2	hallInd2
mPWM1	PWM-styring af motor 1	0-5V 	mUd1	mInd1
mPWM2	PWM-styring af motor 2	0-5V	mUd2	mInd2
mPWM3	PWM-styring af motor 3	0-5V	mUd3	mInd3
mPWM4	PWM-styring af motor 4	0-5V	mUd4	mInd4
mPWM5	PWM-styring af motor 5	0-5V	mUd5	mInd5
mPWM6	PWM-styring af motor 6	0-5V	mUd6	mInd6
brugerIndSig	Interaktion med brugeren gennem touchskærm	Touch input fra skærm på mobiltelefon	brugerInd	brugerIndInt
brugerUdSig	Interaktion med brugeren gennem touchskærm	Lys i form af billede på skærm på mobiltelefon	brugerUd	brugerUdInt
bevægelse1Sig	Rotation af akse	0-180°	mBevæg1	bevægelse1
bevægelse2Sig	Rotation af akse	0-180°	mBevæg2	bevægelse2
bevægelse3Sig	Rotation af akse	0-180°	mBevæg3	bevægelse3
bevægelse4Sig	Rotation af akse	0-180°	mBevæg4	bevægelse4
bevægelse5Sig	Rotation af akse	0-180°	mBevæg5	bevægelse5
bevægelse6Sig	Rotation af akse	0-180°	mBevæg6	bevægelse6
akse1PosSig	Positionen af 1. Akse	Mekanisk position	akse1Pos	akse1PosInd

Tabel 5 - Tabel over signaler i overordnet IBD

6.2 Systemets hardwareenheder

Herefter følger en gennemgang af systemets forskellige enheder, som de fremstår på *figur 9*, deres funktion i systemet og hvordan de forbindes til omverdenen og de øvrige enheder.

6.2.1 Mobiltelefon

Denne blok består af en smartphone med softwaren *MobilApp* installeret. Kravene til denne er, at den har en touchskærm og at den har styresystemet Android Oreo installeret. Denne blok er købt ind udefra og bliver ikke udviklet som en del af dette projekt.

Mobiltelefon indeholder følgende porte:


- **mobilWifi : Wifi** er en trådløs forbindelse fra mobiltelefonen, og dermed dens software, til *RPi*.
- **brugernIndInt : Touch** er indgående kommunikation med brugeren gennem mobiltelefonens indbyggede touchskærm.
- **brugernUdInt : Lys** er udgående kommunikation med brugeren gennem mobiltelefonens indbyggede touchskærm.

6.2.2 RPi

RPi er en forkortelse for *Raspberry Pi Zero Wifi*, og er en ARM-baseret computer, der har softwaren *RPiApp* installeret på et Linux-operativsystem. Denne blok er købt ind udefra og bliver ikke udviklet som en del af dette projekt.

Blokkens primære funktion i dette projekt er at modtage kommandoer over wifi fra *Mobiltelefon* og omsætte disse til lavniveau-kommandoer, der sendes til *PSoC*.

RPi indeholder følgende porte:

- **rpilWifi : Wifi** er en trådløs forbindelse fra *Mobiltelefon*.
- **rpil2cUd : I2C** er forbindelsen til blokken *PSoC*. 
- **rpil2cClkUd : I2c clock** er clock signalet til I2C-kommunikationen med *PSoC*.

- **rpiVccInd : +3.3V** er spændingsforsyning til blokken.

6.2.3 I2CLevelConverter

Blokken har til opgave at konvertere spændingsniveauet for I2C-signalet fra *RPi* til *PSoC* fra 3.3V ved *RPi* til 5V ved *PSoC*.

Blokken indeholder følgende porte:

- **convInd : I2C** er signalet fra *RPi*.
- **convClkInd : I2C clock** er clock signalet for I2C på 3.3V-siden af converteren.
- **convUd : I2C** er signalet til *PSoC*.
- **convClkUd : I2C clock** er clock signalet for I2C på 5V-siden.
- **conv3VInd : +3.3V** er 3.3V forsyningsspænding.
- **conv5VInd : +5V** er 5V forsyningsspænding.

6.2.4 PSoC

PSoC er en forkortelse for *Programmable System on Chip* og er som navnet antyder en programmerbar chip, der indeholder en ARM-processor samt programmerbar logik og analog elektronik. Chippen, der benyttes i dette projekt, er monteret på et CY8CKIT-udviklerkit, der indeholder bl.a. en nødvendig *programmer*. Denne blok er købt ind udefra og bliver ikke udviklet som en del af dette projekt.

I dette projekt benyttes blokken til at omsætte højniveaukommandoer fra *RPi* så som ønskede vinkeler på motorer til PWM-signaler til motorstyring, samt håndtering af returverdier fra sensorer og reaktion på disse.

PSoC indeholder følgende porte:


- **psocI2cInd : I2C** er forbindelsen mellem *PSoC* og *RPi*.
- **psocI2cClkInd : I2C** er clocksignalet til I2C-kommunikationen med *RPi*.
- **psocVccInd : +5V** er spændingsforsyning til *PSoC*.
- **mUd[1-6] : PWM** er PWM-signaler til servomotorene på hver akse.

- **strømsensInd1 : Signal** er sensorsignalet fra *Strømsensor*, der indgår i *Strømdistribution*. Signalet er et analogt signal der benyttes til begrænsning af robottens fysiske styrke. Måler strømmen i *servomotor6*, der opererer robottens hånd.
- **strømsensInd2 : Signal** er sensorsignalet fra *Strømsensor*, der indgår i *Strømdistribution*. Signalet er et analogt signal, der benyttes til begrænsning af robottens fysiske styrke. Måler strømmen i *servomotor1*, der opererer robottens base.
- **optoInd : Signal** er et signal fra blokken *Positionssensor*, som benyttes til at bestemme 1. akse position.
- **hallInd1 : Signal** er et signal fra blokken *Positionssensor*, som benyttes til at bestemme 1. akse position. Registrere 0° på aksen.
- **hallInd2 : Signal** er et signal fra blokken *Positionssensor*, som benyttes til at bestemme 1. akse position. Registrere 180° på aksen.

6.2.5 Servomotor 1-6 : Motor

Disse blokke beskriver de 6 servomotorer, der indgår i RoboPlay. Disse enheder består hver især af en MG996R servomotor, der er indkøbt til projektet. Hver motor sidder på den akse, hvis nummer korresponderer med den pågældende servomotors nummer. *servomotor1* og *servomotor6* har separate blokke i *figur 9*, da deres respektive stelforbindelser forbindes separat til *Strømdistribution* for måling af strøm.

Hver servomotor har følgende porte:

- **mVccInd[1-6] : +7V** er den primære spændingsforsyning til hver servomotor.
- **mInd[1-6] : PWM** er det PWM-signal, der kontrollerer hver servomotor.
- **mGnd[1-6] : Signal** er stelforbindelsen for hver servomotor. Kun mGnd1 og mGnd6 er inkluderet i dette  iagram, da de øvrige forbindes direkte til stel, og alle direkte stelforbindelser er udeladt.

6.2.6 Strømdistribution

Denne blok er en overordnet blok, der indeholder til dels strømdistribution til hver af systemets enheder (foruden *Mobiltelefon*, der forsynes af sit interne batteri), til dels strømsensorkredsløb til måling af strømmen igennem to motorer. Blokken vil blive udviklet som en del af dette projekt.

Blokken indeholde følgende porte:

- **distInd : +7V** er den primære spændingsforsyning til systemet. Alle systemets enheder trækker på denne forsyning.
- **psocDistUd : +5V** er en reguleret spændingsforsyning til *PSoC*.
- **rpiDistUd : +3.3V** er en reguleret spændingsforsyning til *RPI*.
- **motorDistUd : +7V** er spændingsforsyning til robotens motorer. Reguleres ikke.
- **strømsensor1Ud : Signal** er et analogt spændingssignal, der er ækvivalent til den strøm, der løber i *servomotor6*.
- **strømsensor2Ud : Signal** er et analogt spændingssignal, der er ækvivalent til den strøm, der løber i *servomotor1*.
- **possensDistUd : +5V** er forsyningsspændingen til blokken *Positionssensor*.
- **håndInd : Signal** er stilledningen fra *servomotor6*. Denne port benyttes til at måle strømmen gennem denne motor.
- **baseInd : Signal** er stilledningen fra *servomotor1*. Denne port benyttes til at måle strømmen gennem denne motor.
- **afbryderInd : Mekanisk** repræsenterer brugerens operation af systemets hovedafbryder.

6.2.7 Positionssensor

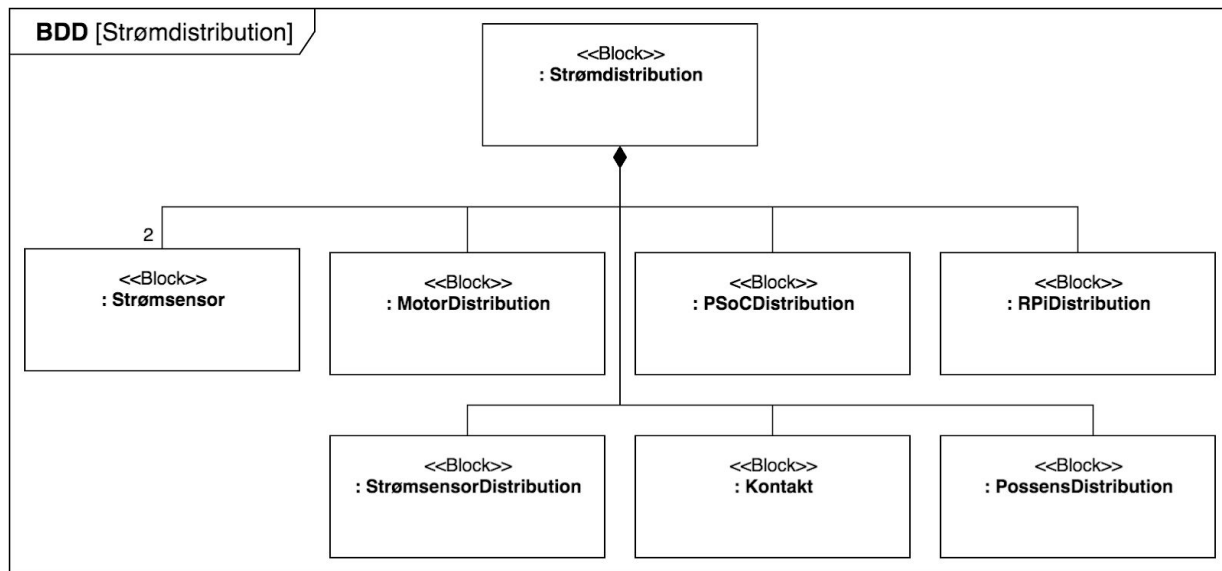
Denne overordnede blok har til opgave at registrere positionen af *Akse 1*.

Blokken indeholder følgende porte:

- **hallUd[1-2] : Signal** fra *Positionssensor* til *PSoC*, der indikere at en Hallsensor står ud for en magnet.
- **optoUd : Signal** fra *Positionssensor* til *PSoC*, der indikere at optocoupleren registrere et mellemrum i *Hjul*.
- **posVccInd : +5V** er forsyningsspænding til *Positionssensor*.
- **akse1PosInd : Position** er den fysiske position i mekaniske position af robotens 1. akse. Dvs. basen, som armen reterer om.

6.3 Strømdistribution

Herunder, på *figur 10* ses et blokdefinitionsdiagram for blokken *Strømdistribution*. Blokken har til opgave at tage en fælles forsyningsspænding til RoboPlay og fordele de relevante spændinger til systemets dele samt at måle strømmen til *servomotor1* og *servomotor6* og levere en ækvivalent spænding til måling i *PSoC*.



Figur 10 - BDD for Strømdistribution

Som det ses på *figur 10*, er denne blok delt op i de mindst mulige delblokke. Alle blokke, hvis navn ender på *Distribution*, repræsenterer spændingsforsyning til de relevante delsystemer. Blokken *Kontakt* repræsenterer systemets hovedafbryder, og blokken *Strømsensor* vil blive behandlet i det følgende.

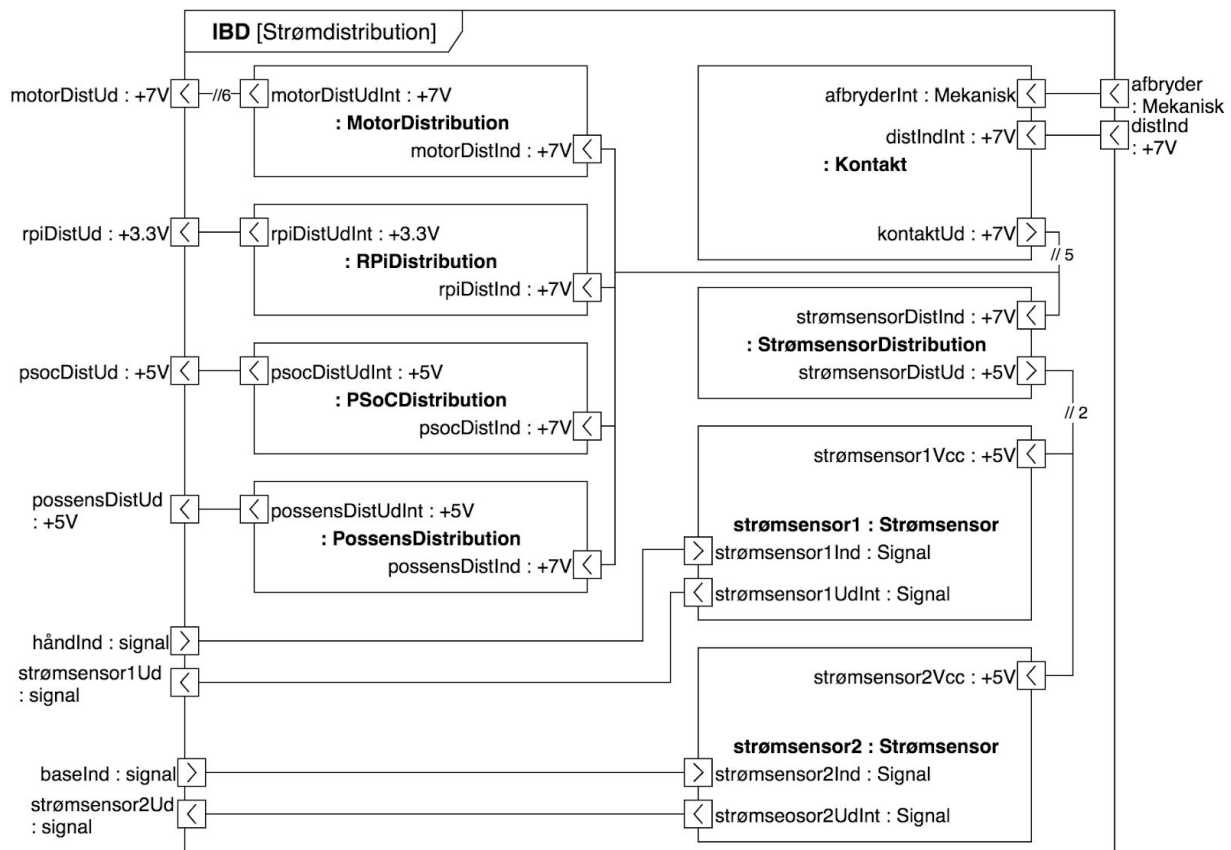
På *tabel 6* herunder ses beskrivelser af de blokke, der indgår i blokken *Strømdistribution*.

Blok	Beskrivelse
Strømdistribution	Den overordnede blok.
Strømsensor	Måler strømmen fra en enhed til stel. Se separat afsnit om denne enhed nedenfor.
Forstærker-Distribution	Leverer forsyningsspænding til <i>Strømsensor</i> .

MotorDistribution	Fordeler forsyningsspændinger til alle motorer.
PSoCDistribution	Leverer reguleret forsyningsspænding til <i>PSoC</i> .
RPiDistribution	Leverer reguleret forsyningsspænding til <i>RPI</i> .
Kontakt	Hovedafbryder for systemet.
Possens-Distribution	Leverer reguleret forsyningsspænding til <i>Positionssensor</i> .

Tabel 6 - Oversigt over delblokke i *Strømdistribution*

Herunder, på *figur 11*, ses et internt blokdiagram for blokken *Strømdistribution*. Som det ses, fordeles den primære forsyningsspænding fra inputtet *distInd* til de øvrige strømforsyninger igennem en hovedafbryder, *Kontakt*. Desuden er blokkene *strømsensor[1-2]* inkluderet i dette delsystem, da det er naturligt at have en enhed til måling af den strøm, der løber i et system i forbindelse med strømforsyningen til systemet. Stelforbindelser er underforståede.



Figur 11 - IBD for Strømdistribution

På *tabel 7* herunder ses en oversigt over hvilke signaler og porte, der indgår i IBD for *Strømdistribution* fra *figur 11*.

Signalnavn	Funktion	Område	Port 1	Port 2
afbryderSig	Bestemmer om der er spænding på systemet	Mekanisk bool	afbryder	afbryderInt
distForsyning	Primær spændingsforsyning til systemet	+7V +/- 0.2V	distInd	distIndInt
primær-Spænding	Primær spænding til forsyningsenheder	+7V +/- 0.2V	kontaktUd	strømsensor-DistInd motordistInd rpiDistInd psocdistInd possensDistInd
strømsensor-Spænding	Spændingsforsyning til <i>StrømSensor-Forstærker</i>	+5V +/- 0.2V	strømsensor-DistUd	strømsensorVcc
motorSpænding	Forsynings-spænding til <i>servomotor[1-6]</i> 1-6	+7V +/- 0.2V	motorDistUdInt	motorDistUd
rpiSpænding	Forsynings-spænding til RPi	+3.3V +/- 0.2V	rpiDistUdInt	rpiDistUd
psocSpænding	Forsynings-spænding til PSoC	+5V +/- 0.2V	psocDistUdInt	psocDistUd
possens-Spænding	Forsynings-spænding til PSoC	+5V +/- 0.2V	possensDist-UdInt	possensDistUd
håndGndSigInt	Stel for <i>servomotor6</i> til måling af strøm	0-5V	håndInd	strømsensorInd1
baseIndSig	Stel for	0-5V	baseInd	strømsensorInd2

	<i>Servomotor1</i> til måling af strøm			
strømsensor1-udSig	Analogt signal fra måling af strømstyrke	0-5V	strømsensor1Ud	strømsensor1-UdInt
strømsensor2-UdSig	Analogt signal fra måling af strømstyrke	0-5V	strømsensor2Ud	strømsensor2-UdInt

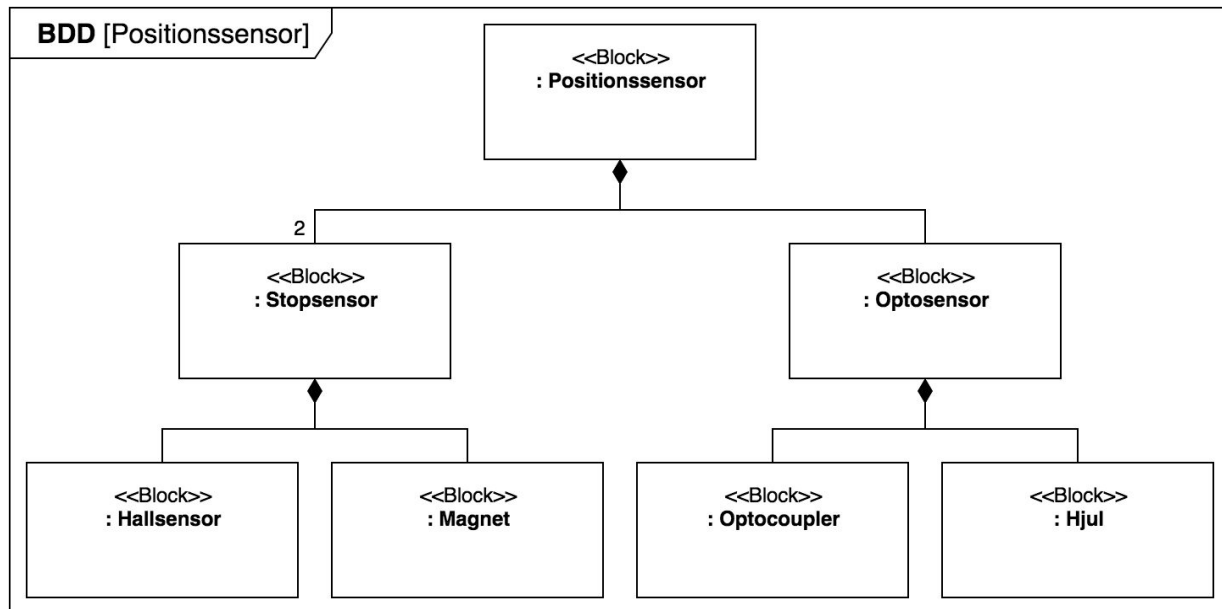
Tabel 7 - Oversigt over signaler i *Strømdistribution*

6.4 Strømsensor

Som det fremgår af *figur 11*, indgår der i dette projekt en enhed til måling af strøm. Denne enhed implementeres kun på to servomotorer i systemet for at bevise dens funktion, men et endeligt, produktionsmodent produkt ville inkludere disse på hver motor. Formålet med denne enhed er at give feedback til systemets software, der kan benyttes til at begrænse robotens styrke for derved dels at undgå at en motor kan beskadige selve roboten, dels at sikre at systemet ikke kan forvolde personskaade. Dette opnås ved at stelledningen fra den servomotor, der skal begrænses, føres igennem kredsløbet *Strømsensor*, der aflæser strømmen igennem denne stelledning og leverer en ækvivalent spænding til en indgang *strømsensInd[1-2]* på *PSoC*.

6.5 Positionssensor

Til detektion af hver akse position, benyttes en kombination af en magnetisk og en optisk sensor. På *figur 12* ses definitionen af den overordnede blok *Positionssensor*. Som det fremgår, er *Positionssensor* opdelt i *Stopsensor* og *Optosensor*. Blokken *Stopsensor* benyttes til at registrere når hjulet når sine yderpunkter, så en kendt vinkel kan benyttes som udgangspunkt for at *Optosensor* kan tælle mellemrum i *Hjul*, for derved at kunne bestemme første akse position i grader.



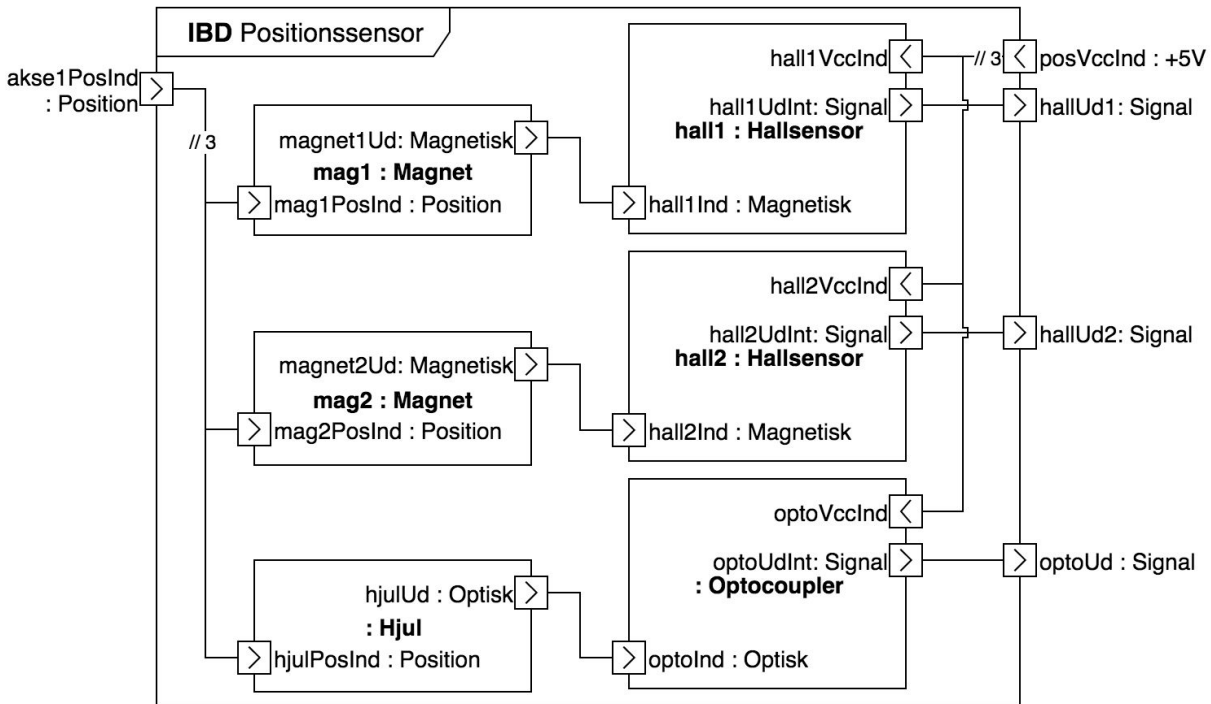
Figur 12 - BDD for Positionssensor

På *tabel 8* herunder ses en tabel over de blokke, der indgår i *Positionssensor*, samt beskrivelse af deres funktioner.

Blok	Beskrivelse
Positionssensor	Den overordnede blok, der beskrives.
Stopsensor	Overordnet blok, der beskriver magneter og Hallsensorer til registrering af yderpunkt for akse.
Optosensor	Overordnet blok, der beskriver hjul og optosensor til registrering af akse vinkel.
Hallsensor	Enhed, der registrere om der er en magnet ud for den.
Magnet	Magnet monteret på robot som markering af yderpunkt.
Optocoupler	Enhed, der registrerer mellemrum i hjul.
Hjul	Hjul med mellemrum, der kan registreres af <i>Optocoupler</i> .

Tabel 8 - Oversigt over delblokke i *Positionssensor*

På figur 13 herunder ses et internt blokdiagram for *Positionssensor*. Porten *akse1PosInd* beskriver første akse fysiske position i grader, og er den information, blokken har til opgave at registrere. Stelforbindelser er underforståede.



Figur 13 - IBD for *Positionssensor*

På tabel 9 herunder ses en tabel over de signaler, der indgår i figur 13.

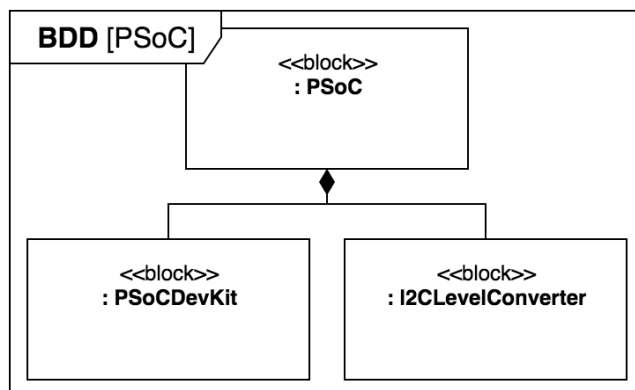
Signalnavn	Funktion	Område	Port 1	Port 2
possensVcc	Forsynings-spænding til <i>Possens</i>	+5V	posVccInd	hall1VccInd hall2VccInd optoVccInd
hall1UdSig	Signal om registrering af magnet ved akse yderpunkt.	0-5V	hall1UdInt	hall1Ud
hall2UdSig	Signal om registrering af magnet ved akse	0-5V	hall2UdInt	hall2Ud

	yderpunkt.			
optoUdSig	Signal om registrering af mellemrum i hjul	0-5V	optoUdInt	optoUd
magnetiskKraft1	Magnetisk kraft, der trækker Hall-sensor	Magnetisk bool	magnet1Ud	hall1Ind
magnetiskKraft2	Magnetisk kraft, der trækker Hall-sensor	Magnetisk bool	magnet2Ud	hall2Ind
optiskLys	Lys, der blokeres af hjul til registrering af akse vinkel	Optisk bool	hjulUd	optoInd
position	Fysisk position af akse, der ønskes detekteret	0°-180°	akse1PosInd	mag1PosInd mag2PosInd hjulPosInd

Tabel 9 - Signaler i *Positionssensor*

6.6 PSoC

Under blokken *PSoC* plaseres blokken *I2CLevelConverter* som det fremgår af figur 14 herudover. Blokken kunne lige så godt tilhøre *RPi* eller det overordnede niveau, men det er besluttet at lægge den her for at holde det overordnede BDD til de mest essentielle blokke.



Figur 14 - BDD over PSoC

På *tabel 10* herunder ses er oversigt over de blokke, der indgår i *figur 14*.

Blok	Beskrivelse
PSoC	Den overordnede blok, der beskrives.
PSoCDevKit	Det fysiske PSoC 5LP development kit, der benyttes i projektet.
I2CLevelConverter	En chip til konvertering af I2C signalniveauer fra PSoC's 5V nominelt til RPi's 3.3V.

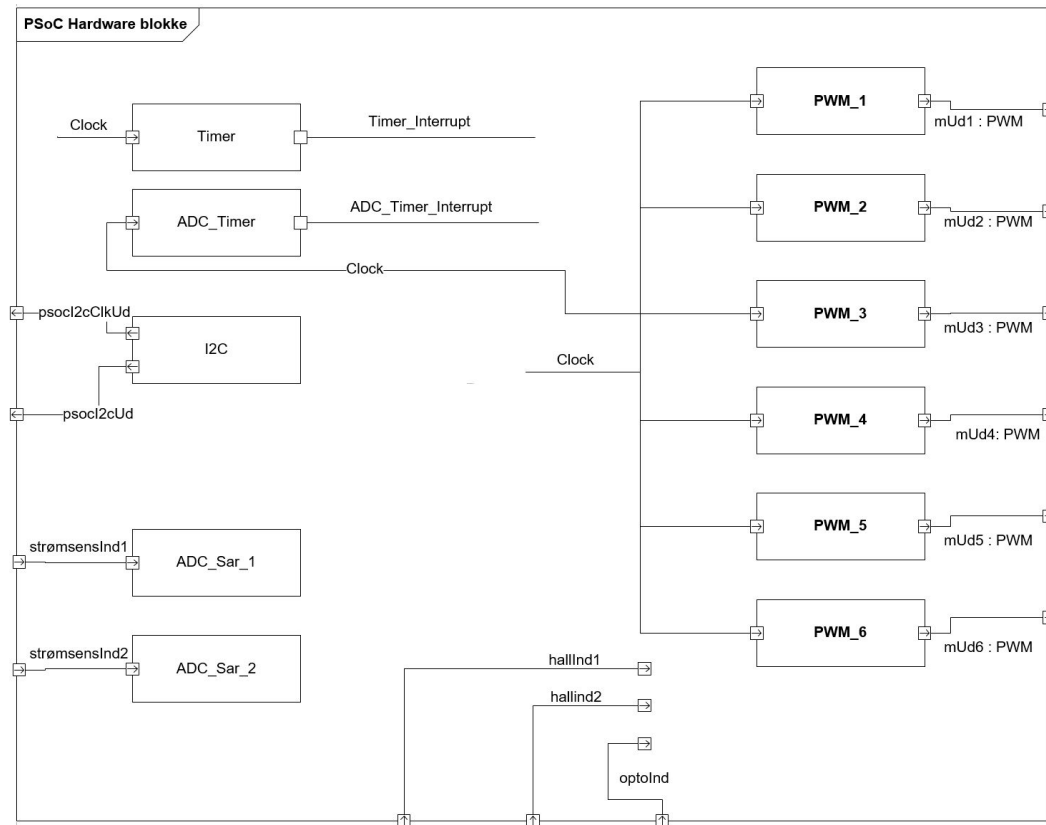
Tabel 10 - Oversigt over delblokke i *PSoC*.

6.6.1 Fordeling af ressourcer

Da flere gruppemedlemmer hver for sig implementerer forskellige dele af systemet, der benytter ressourcer på PSoC'en, er det nødvendigt at lave en fordeling af disse ressourcer i forbindelse med systemarkitekturen.

På PSoC'en bruges der af de interne hardware enheder, en SAR converter pr. strømsensor, en PWM blok pr. Motor, samt en samlet clock til at alle PWM-blokke, en timer og en I2C blok samt digitale input pins fra optocoupler og hall sensors.

Hvordan disse blokke er sat sammen kan ses på *figur 15*.



Figur 15

Blok	Beskrivelse
Timer	Har til opgave at styre hvor mange gange i sekundet der laves en ændring i motoren. Denne sættes til 50 gange i sekundet. Har sin egen clock.
ADC_Timer	Har til opgave at styre hvor mange gange i sekunder der converteres i ADC'en. Denne sættes til 100 gange i sekundet. Deler clock med de 6 PWM-blokke
I2C	Har til opgave at stå for kommunikation mellem PSoC og RPI. Har 2 signaler ud af PSoC'en i form af psocI2cClkUd og psocI2cUd
ADC_Sar_1	Denne blok skal modtage et signal udefra i form af strømsensInd1 som er et analogt

	signal, blokken har til opgave at lave dette signal om til et digital signal som vi kan bruge i software
ADC_Sar_2	Denne blok skal modtage et signal udefra i form af strømsensInd2 som er et analogt signal, blokken har til opgave at lave dette signal om til et digital signal som vi kan bruge i software
PWM_1	Denne blok har til opgave at sende et PWM signal ud på en af PSoC'ens pins, i form af signalet mUd1 : PWM. Den styres af en clock der er forbundet til alle PWM-blokkene, og ADC_Timeren
PWM_2	Denne blok har til opgave at sende et PWM signal ud på en af PSoC'ens pins, i form af signalet mUd2 : PWM. Den styres af en clock der er forbundet til alle PWM-blokkene, og ADC_Timeren
PWM_3	Denne blok har til opgave at sende et PWM signal ud på en af PSoC'ens pins, i form af signalet mUd3 : PWM. Den styres af en clock der er forbundet til alle PWM-blokkene, og ADC_Timeren
PWM_4	Denne blok har til opgave at sende et PWM signal ud på en af PSoC'ens pins, i form af signalet mUd4 : PWM. Den styres af en clock der er forbundet til alle PWM-blokkene, og ADC_Timeren
PWM_5	Denne blok har til opgave at sende et PWM signal ud på en af PSoC'ens pins, i form af signalet mUd5 : PWM. Den styres af en clock der er forbundet til alle PWM-blokkene, og ADC_Timeren
PWM_6	Denne blok har til opgave at sende et PWM signal ud på en af PSoC'ens pins, i form af signalet mUd6 : PWM. Den styres af en clock der er forbundet til alle PWM-blokkene, og ADC_Timeren
hallInd1	Modtager et signal udefra i form af signalet

	hallInd1, som bruges i software.
hallInd2	Modtager et signal udefra i form af signalet hallInd2, som bruges i software.
optoInd	Modtager et signal udefra i form af signalet optoInd, som bruges i software.

Som det kan ses har hver eneste motor sin egen PWM-blok, hvortil de 6 PWM-blokker har en dertil hørende samlet clock. Dette skyldes at PSoC har en begrænsning på antallet af clock, hvilket vil sige at hvis man vil bruge en clock til noget andet, vil dette ikke være muligt da de alle er brugt til PWM-blokkene. Derudover ses også de 2 SAR convertere som hver modtager et input, som for den ene er basen (1. akse) og for den anden er hånden (6. akse). Dertil ses også I2C blokken som har udgange SDA og SCL, disse føres videre til en pin så de kan få et input udefra. Herudover ses også timeren som har en clock, samt et interrupt forbundet. Til sidst ses de 3 digitale pin PositionsSensor 1, PositionsSensor 2 og Optokobler.

PSoC's porte er fordelt som det kan ses i *tabel 11*.

Port på PSoC	Signal	Beskrivelse
P2_0	mUd1 : PWM	Sender et PWM signal til motoren på 1. akse.
P2_1	mUd2 : PWM	Sender et PWM signal til motoren på 2. akse.
P2_2	mUd3 : PWM	Sender et PWM signal til motoren på 3. akse.
P2_3	mUd4 : PWM	Sender et PWM signal til motoren på 4. akse.
P2_4	mUd5 : PWM	Sender et PWM signal til motoren på 5. akse.
P2_5	mUd6 : PWM	Sender et PWM signal til motoren på 6. akse.
P2_6	strømsensInd1	Modtager et analogt signal fra fra robottens 6. akse.

P2_7	strømsensInd2	Modtager et analogt signal fra robottens 1. akse.
P1_7	optoInd	Modtager et højt eller lavt signal fra optokobleren.
P1_6	hallInd1	Modtager et højt eller lavt signal fra 1. Hall sensor.
P1_5	hallInd2	Modtager et højt eller lavt signal fra 2. Hall sensor.
P12_0	psocI2cClkUd	Clocksignal til I2C kommunikation med RPi.
P12_1	psocI2cUd	Nyttesignal til I2C kommunikation med RPi.

Tabel 11 - Tabel over fordeling af PSoC'ens porte.

7 Kommunikationsprotokoller

I dette afsnit beskrives de protokoller, som benyttes til kommunikation mellem RoboPlay's forskellige delsystemer.

7.1 Protokol mellem MobilApp og RPiApp

Protokollen, der benyttes mellem MobilApp og RPiApp, bygger på TCP, og dataen, som sendes, består altid af 16 bit.

7.1.1 TCP klient/server

MobilApp'en opsættes som klient mens RPiApp fungerer som server. TCP protokollen, som blev udvalgt med baggrund i en undersøgelse, som ses beskrevet i et særskilt bilag¹, understøtter tovejskommunikation, men kun envejskommunikation fra MobilApp til RPiApp implementeres i udviklingen af RoboPlay.

¹ Bilag 17, Analyse af protokol mellem MobilApp og RPiApp

Forbindelsen mellem enhederne sker via et WIFI hotspot, som Raspberry Pi'en har ansvaret for at udbyde. Klienten forbinder til WIFI hotspotet, hvorefter TCP forbindelsen oprettes under følgende forudsætninger:

- Raspberry Pi'en skal have IP-adressen 10.9.8.2
- Den benyttede mobiltelefon tildeles en IP-adresse via DHCP
- RPiApp opsættes til at lytte efter klienter på port 4000

7.1.2 Protokol for udførelse af bevægelser

Jf. kravspecifikationen er det nødvendigt at styre robotten i 12 forskellige bevægelser, og hver af disse skal kunne både startes og stoppes.

Det implementeres således, at hver besked består af 16 bits. De 12 LSB'er er relateret til hver sin bevægelse således, at et 0 betyder en bevægelse skal stoppes, mens et 1-tal betyder, at en bevægelse skal påbegyndes.

I *tabel 12* ses en beskrivelse af betydningen af den enkelte bit.

Bit nr.	Beskrivelse
0 (LSB)	Starter/stopper bevægelsen "Åbn".
1	Starter/stopper bevægelsen "Luk".
2	Starter/stopper bevægelsen "Roter med uret".
3	Starter/stopper bevægelsen "Roter mod uret".
4	Starter/stopper bevægelsen "Tilt op".
5	Starter/stopper bevægelsen "Tilt ned".
6	Starter/stopper bevægelsen "Frem".
7	Starter/stopper bevægelsen "Tilbage".
8	Starter/stopper bevægelsen "Drej højre".
9	Starter/stopper bevægelsen "Drej venstre".
10	Starter/stopper bevægelsen "Op".

11	Starter/stopper bevægelsen "Ned".
12	Benyttes ikke.
13	Benyttes ikke.
14	Benyttes ikke.
15 (MSB)	Benyttes ikke.

Tabel 12 - Betydning for hver enkelt bit i protokollen mellem MobilApp og RPiApp. Et logisk "0" stopper en bevægelse, mens et logisk "1" starter en bevægelse.

I *tabel 13* ses eksempler på, hvordan bitmønstre for udvalgte beskeder ser ud.

Ønskede bevægelser	Bitmønster
Alle bevægelser stoppes	0000 0000 0000 0000
"Åbn".	0000 0000 0000 0001
"Frem" og "Op".	0000 0010 0010 0000

Tabel 13 - Eksempler på sammenhæng mellem den/de ønskede bevægelser og det tilhørende bitmønstre som udgør beskeden, der sendes fra MobilApp til RPiApp.

Visse kombinationer af kommandoer vil altid være ugyldige, idet det ikke giver mening fx at udføre bevægelserne "Åbn" og "Luk" samtidigt. Disse på hinanden modsat rettede kommandoer er grupperet i bitmønstret således, at bit nr. 0 og 1 ophæver hinanden, bit nr. 2 og 3 ophæver hinanden og så fremdeles.

Eksempler på ugyldige kommandoer ses i *tabel 14*.

Ønskede bevægelser	Bitmønster
"Åbn" og "Luk".	0000 0000 0000 0011
"Op" og "Ned".	0000 1100 0000 0000

Tabel 14 - Eksempler på ugyldige kommandoer, som ophæver hinanden. Bemærk hvordan de ugyldige kommandoer er grupperet to og to.

7.1.3 Protokol for indstilling af robot til startposition

For at indstille robotten til dens startposition som beskrevet i UC6 benyttes kommandoen, som ses i *tabel 15*.

Ønskede bevægelser	Bitmønster
Robotten indstilles til dens startposition	1111 1111 1111 1111

Tabel 15 - Kommandoen som indstiller alle robotens led til deres startpositioner.

7.2 Protokol mellem RPiApp og PSoCApp

Protokollen mellem RPiApp og PSoCApp benytter I2C-kommunikation², hvor RPiApp er master-enheden og PSoCApp er slave-enheden, hvilket betyder, at RPiApp altid iværksætter kommunikationen mellem applikationer ved enten at skrive til eller læse fra PSoCApp.

Når RPiApp skriver til eller læse fra PSoCApp, så består beskeden af syv bytes, hvoraf de seks første er data vedrørende de seks akser, som PSoCApp skal indstilles efter, mens den sidste og syvende er CRC. CRC benyttes til at forsikre data-integriteten. Antallet af logiske 1-taller i de første seks bytes tælles og udgør dermed den syvende byte. Det er op til modtageren af beskeden at kontrollere data-integriteten ud fra den syvende (CRC) byte.

Når der i denne protokol beskrives en bytes indhold af bits er det fra MSB mod LSB, hvor MSB er bit nr. 7 og LSB er bit nr. 0. Ved angivelsen af "X" i et binært mønster menes "don't care".

7.2.1 Opsætning

RPiApp og PSoCApp skal opsættes til at benytte I2C med en hastighed på 400 kbps og PSoCApp skal kommunikere på adressen 0x08.

7.2.2 Besked-struktur for skrivning

7.2.2.1 Byte #1-6

Fælles for beskedens seks data-bytes er, at de hver især kan indeholde et antal grader, som relaterer til en given akse. For eksempel kan første akse indstilles i intervallet mellem 0 og 180

² Bilag 18, Analyse af protokol mellem RPiApp og PSoCApp

grader, hvor 0 grader betegnes som “dreje til helt til venstre”, mens 180 grader betegnes som at “dreje helt til højre”. I *tabel 16* ses en oversigt over samtlige akser, deres yderpunkter i grader, samt hvordan yderpunkterne skal fortolkes i form af bevægelse. Det skal bemærkes, at graderne også kan indstilles i intervallet mellem de angivne yderpunkter.

Akse nr.	Start yderpunkt		Slut yderpunkt	
	<i>grader</i>	<i>bevægelse</i>	<i>grader</i>	<i>bevægelse</i>
1	0	Drej helt til venstre	180	Drej helt til højre
2	0	Formindsk vinkel til minimum	180	Forøg vinkel til maximum
3	0	Formindsk vinkel til minimum	180	Forøg vinkel til maximum
4	0	Tilt håndled helt ned	180	Tilt håndled helt op
5	0	Roter håndled helt i urets retning	180	Roter håndled helt mod urets retning
6	0	Luk hånden helt	180	Åbn hånden helt

Tabel 16 - Oversigt over aksernes yderpunkter og bevægelserne forbundet hermed

Til indstilling af første akse angives graderne i første byte, for anden akse i anden byte osv. I *tabel 17* ses en oversigt over valide værdier for de seks data-bytes.

Valid værdi(-interval) Binært	Handling
00000000 - 10110100	Indstil aksen til det angivende antal grader
11XXXXXX	Stop bevægelse

Tabel 17 - Valide værdier for hver af de seks data-bytes

7.2.2.2 Byte #7

Den sidste byte er en CRC-byte til kontrol af data-integriteten. Antallet af logiske 1-taller i beskedens første seks bytes tælles og udgør denne byte. I *tabel 18* ses et eksempel på en besked, hvor der tælles 23 1-taller i beskedens første seks bytes, hvorfor indholdet af syvende byte er en binær repræsentation af tallet 23.

Byte #1	Byte #2	Byte #3	Byte #4	Byte #5	Byte #6	Byte #7
10110100	10100001	00011011	01011010	10001111	01011010	00010111

Tabel 18 - Eksempel på korrekt brug af CRC

7.2.3 Besked-struktur for læsning

7.2.3.1 Byte #1-6

De første seks bytes data-bytes, som hver især respektivt indeholder det absolutte antal grader for første til sjette akse. Graderne er angivet i heltal af typen unsigned int.

Valid værdiinterval Binært	Information
00000000 - 10110100	Aksens aktuelle antal grader/hældning

Tabel 19 - Oversigt over protokolens byte 1-6 ved læsning

7.2.3.2 Byte #7

Den sidste byte er en CRC-byte til kontrol af data-integriteten og der henvises til afsnit "[7.2.2.7 - Byte #7](#)" i forrige afsnit.

8 Softwarearkitektur

I dette afsnit beskrives arkitekturen for den software, der kører på hvert enkelt delsystem. Dette gøres med såkaldte applikationsmodeller, hvorfor der i det følgende anvendes klasse- sekvens- og tilstandsdiagrammer i det omfang, det er nødvendigt.

På de forskellige diagrammer benyttes en række stereotyper som er kort forklaret nedenfor:

- <<data>> Angives på sekvensdiagrammer når data enten sendes mellem softwareapplikationer over eksempelvis TCP.
- <<create>> Angives på når et objekt af en klasse kalder constructoren i en anden klasse.
- <<event>> Angives på sekvensdiagrammer når en tråd vækkes af et event.

- <<databinding>> Angives når kommunikationen mellem to klasser sker via databinding som det kendes fra bl.a. C#³.

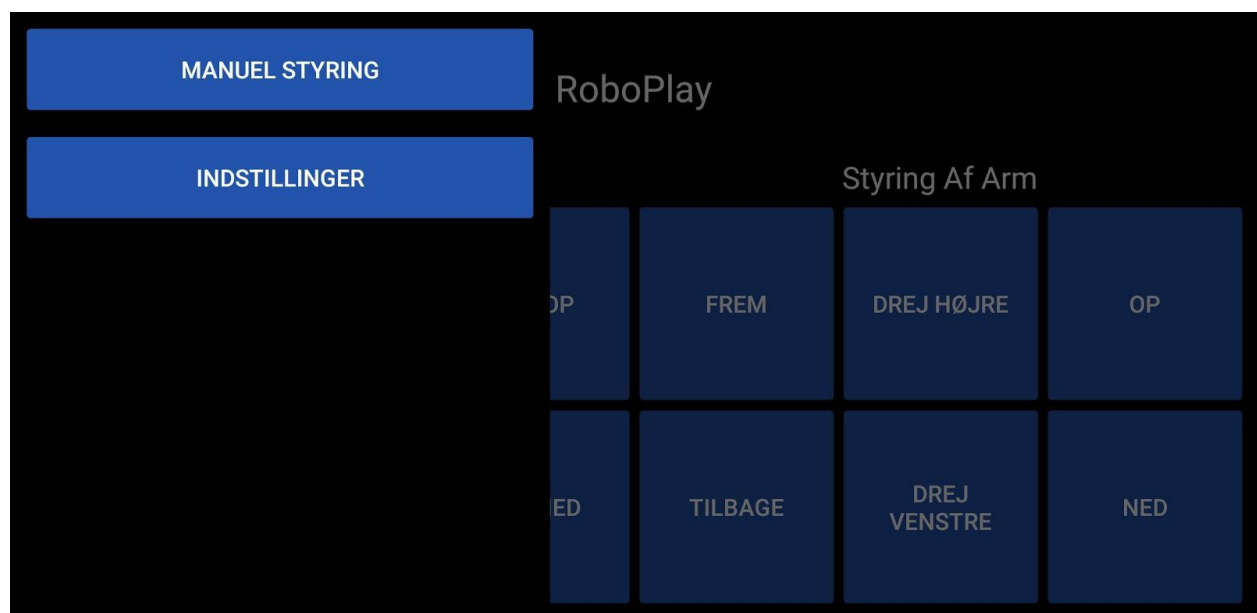
8.1 MobilApp

Applikationsmodellerne for MobilApp er lavet under den antagelse, at applikationen laves under brug af teknologien Xamarin.Forms og dermed programmeres i sproget C#. Som en naturlig følge heraf er den anvendte arkitektur MVVM⁴. Hvordan MVVM er benyttet er forklaret nærmere i bilaget *Software design*.

MobilApp består af følgende skærbilleder:

- Skærbilledet *Menu* som ses på figur 16
- Skærbilledet *Manuel styring* som ses på figur 17
- Skærbilledet *Indstillinger* som ses på figur 18

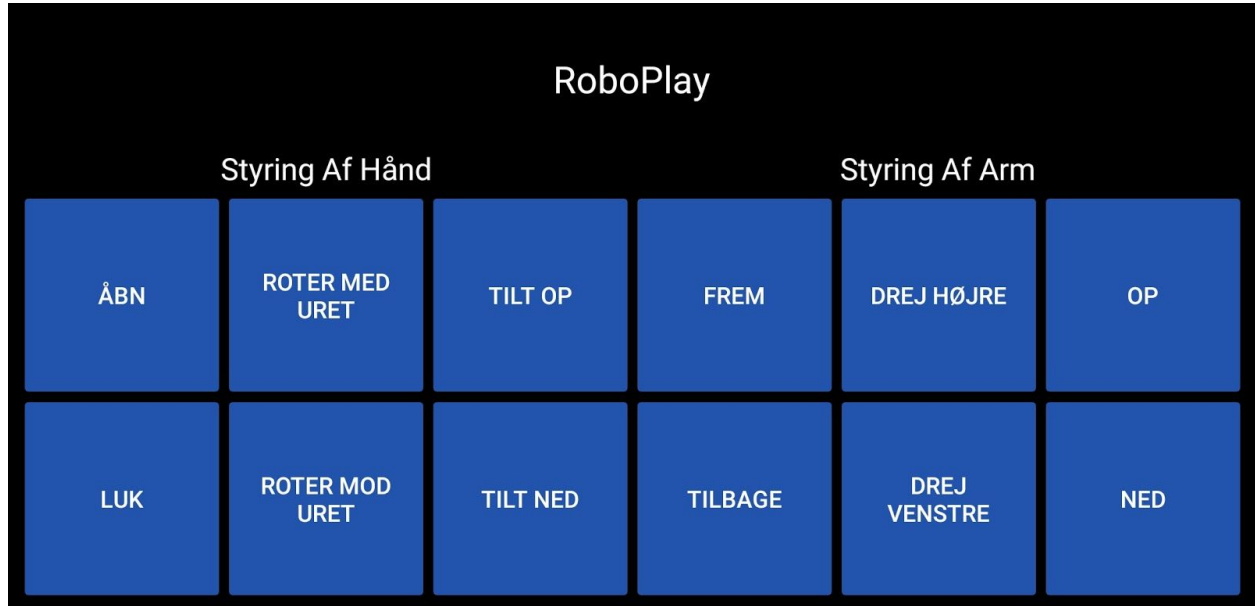
For en nærmere beskrivelse af de enkelte skærbilleder så henvises der til figurteksterne.



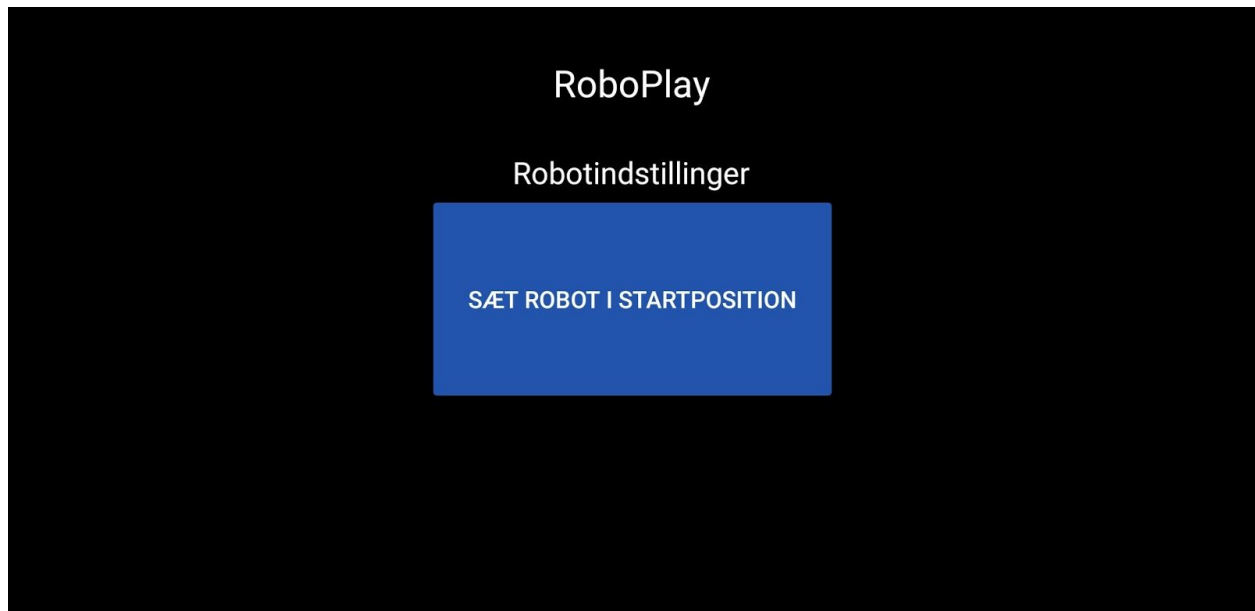
³ <https://docs.microsoft.com/en-us/dotnet/framework/wpf/data/data-binding-overview>

⁴ Forklaring af MVVM ses her: <https://www.wintellect.com/model-view-viewmodel-mvvm-explained/>

Figur 16 - Skærbilledet Menu. En menu hvorfra der kan navigeres til applikationens øvrige skærbilleder. Denne menu er tilgængelig fra samtlige skærbilleder ved med en finger at trække fra kanten af skærmen og ind mod midten af skærmen. Det senest viste skærbillede vil ses i baggrunden og når et skærbilledet vælges i menuen kollapseder menuen således det nye skærbillede vises.



Figur 17 - Skærbilledet Manuel styring. Det skærbillede der vises, når applikationen opstartes. Knapperne i skærbilledet benyttes til at styre robotten, hvorved UC1: Udfør bevægelse og UC7: Grib om objekt udføres herfra. Når en knap holdes nede udføres en given bevægelse - ellers er bevægelsen stoppet.



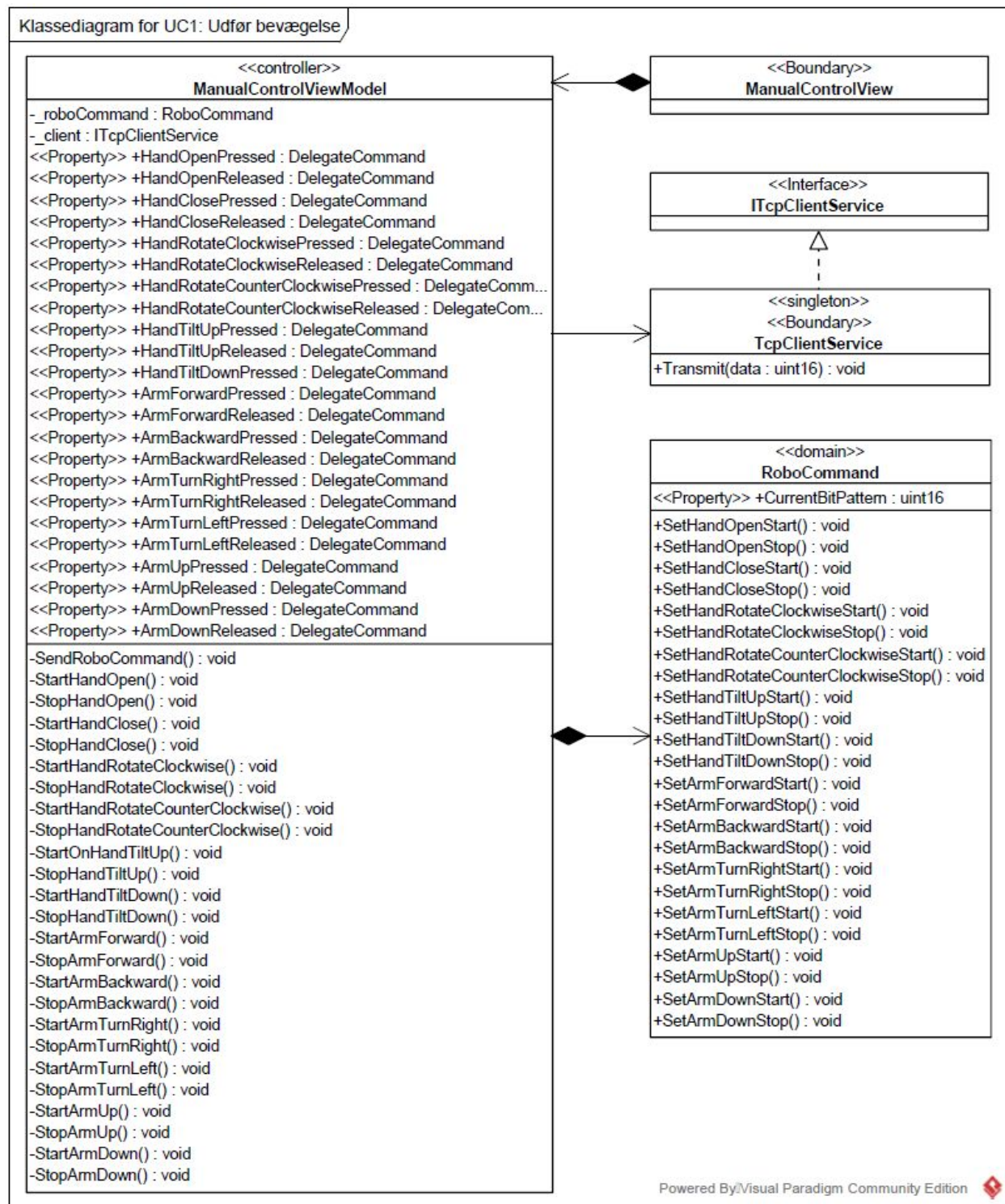
Figur 18 - Skærbilledet Indstillinger. Kun UC6: Sæt robot i startposition udføres fra dette skærbillede.

8.1.1 Applikationsmodel for UC1 og UC7

Denne applikationsmodel viser den underliggende arkitektur for skærbilledet, som ses på *figur 17*, hvorved applikationsmodellen både omfatter UC1 og UC7. Dette skyldes, at de to use cases set fra et implementeringsmæssigt synspunkt er næsten ens.

8.1.1.1 Klassediagram

Klassediagrammet for UC1 og UC7 ses på *figur 19*. En kort forklaring af de viste klasser ses i forlængelse af klassediagrammet.



Figur 19 - Klassediagrammet for UC1 og UC7's applikationsmodel.

ManualControlView

Er selve viewet som ses på *figur 17*. Viewet er knyttet til en view model i form af ManualControlViewModel, hvorved databinding kan ske til denne view models properties.

ManualControlViewModel

Fungerer som controller for use casen. Alt logik omhandlende de forskellige knapper, der ses i ManualControlView, håndteres altså her.

Klassens properties af typen RelayCommand databindes knapperne i viewet til. Hver knap databindes altså til to RelayCommands - en når knappen trykkes ned og en når knappen slippes igen. Når et objekt af typen RelayCommand aktiveres, kaldes en tilhørende metode.

Typen RelayCommand er en del af frameworket Prism, som kort fortalt er et framework der letter implementering af MVVM. Prism omtales nærmere i bilaget *Softwaredesign*.

TcpClientService

Håndterer kommunikationen med RPiApp over TCP protokollen via et objekt af .NET klassen kaldet TcpClient. Implementeret som en singleton hvorved kun ét objekt kan oprettes og dermed oprettes der kun én forbindelse til RPiApp. Implementerer interfacet ITcpClientService, idet det under udfærdigelsen af unit tests vil være nødvendigt at stubbe klassen ud.

RoboCommand

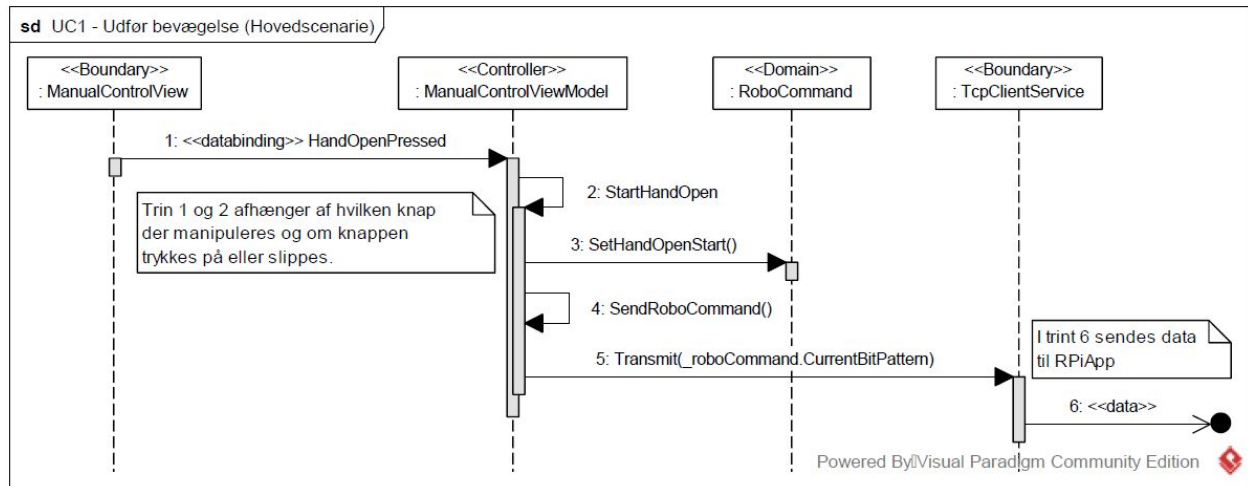
Denne klasse har en property kaldet "CurrentBitPattern", som er et bitmønster der sættes alt efter hvilken kommando der skal sendes til RPiApp. Se afsnit ["7.1 Protokol mellem MobilApp og RPiApp"](#) for yderligere forklaring om kommandoernes opbygning.

ManualControlViewModel har en reference til et objekt af RoboCommand og dette objekt benyttes til at sætte bitmønsteret alt efter hvilke knapper, som brugeren holder nede på brugergrænsefladen. RoboCommand har en række metoder, som benyttes til at ændre bitmønsteret, når knapperne trykkes ned eller slippes.

8.1.1.2 Sekvensdiagram for hovedscenariet

På *figur 20* ses sekvensdiagrammet for UC1 og UC7's hovedscenarier. Sekvensdiagrammet er meget simpelt, da der ikke er to-vejskommunikation mellem MobilApp og RPiApp. Netop fordi

der ikke er to-vejskommunikation, involverer UC1 og UC7's extensions, hvor eksempelvis en forhindring standser en bevægelse, ikke MobilApp.



Figur 20 - Sekvensdiagram for UC1's hovedscenarie. På eksemplet her er det bevægelsen "Åbn" der udføres.

Af diagrammet ses følgende:

- I trin 1 holder brugeren en knap nede eller har sluppet en knap. I eksemplet ovenfor er knappen "Åbn" holdt nede. Via databinding notificeres ManualViewModel om det pressed-event som opstår som følge af, at en knap trykkes ned.
- I trin 2 eksekveres den metode, som RelayCommand'en, der databindes til, peger på.
- I trin 3 sætter controlleren bitmønstret jf. den vedtagne protokol, som er beskrevet tidligere i dette dokument.
- I trin 4 og 5 kaldes metoden Transmit(...) med bitmønstret som parameter på TcpClientService-objektet.
- I trin 6 sendes dataen til RPiApp.

8.1.2 Applikationsmodel for UC2

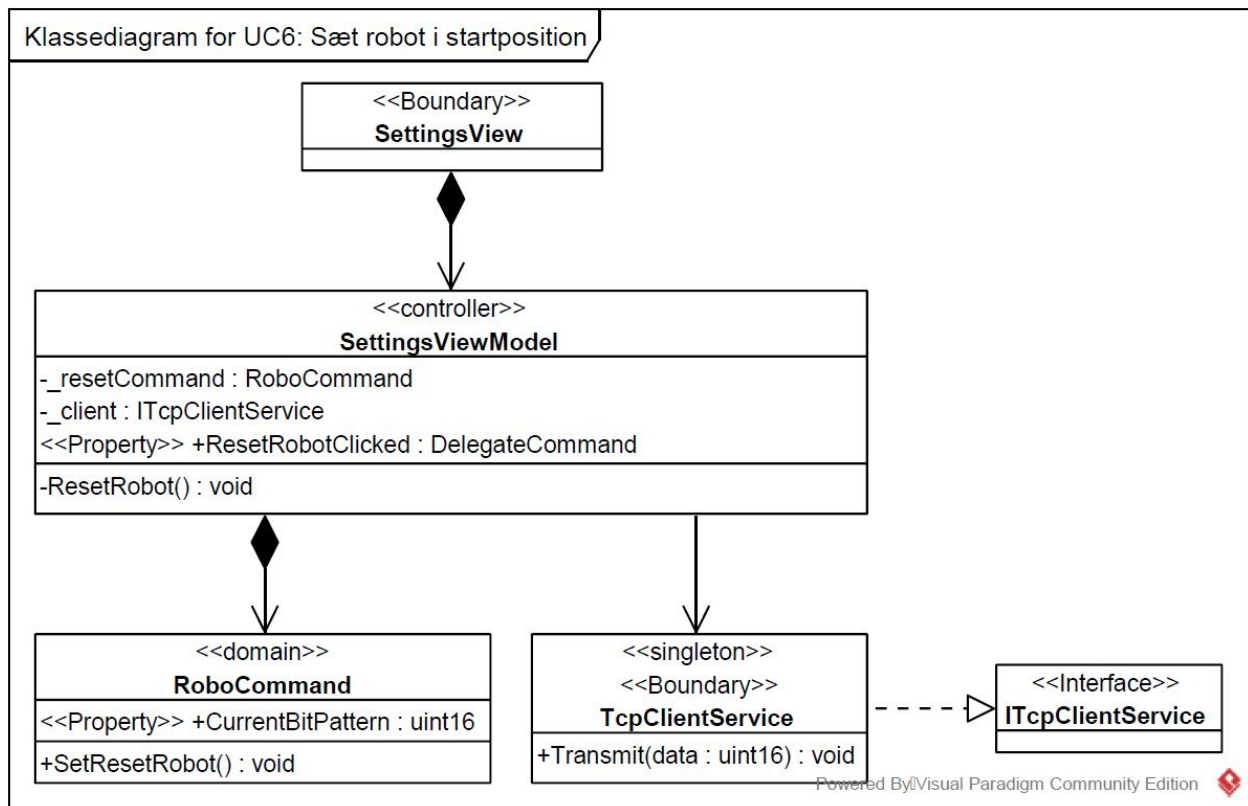
En applikationsmodel er ikke relevant, idet denne use case ikke involverer MobilApp.

8.1.3 Applikationsmodel for UC6

Denne applikationsmodel viser den underliggende arkitektur for UC6 og dermed for skærmbilledet, som ses på *figur 18*.

8.1.3.1 Klassediagram

Klassediagrammet for UC6 ses på *figur 21*. En kort forklaring af de viste klasser følger efter diagrammet.



Figur 21 - Klassediagrammet for UC6's applikationsmodel.

SettingsView

Er selve viewet som ses på *figur 18*. Viewet er knyttet til en view model i form af SettingsViewModel, hvorved databinding kan ske til denne view models properties.

SettingsViewModel

Fungerer som controller for use casen. Alt logik omhandlende den knap, der ses i SettingsView, håndteres altså her.

Klassens property `ResetRobotClicked`, databindes til knappen i viewet, hvorved robotten kan indstilles til dens startposition.

TcpClientService

Håndterer kommunikationen med RPiApp over TCP protokollen via et objekt af .NET klassen kaldet `TcpClient`. Implementeret som en singleton hvorved kun ét objekt kan oprettes og dermed oprettes der kun én forbindelse til RPiApp. Implementerer interfacet `ITcpClientService`, idet det under udfærdigelsen af unit tests vil være nødvendigt at stubbe klassen ud.

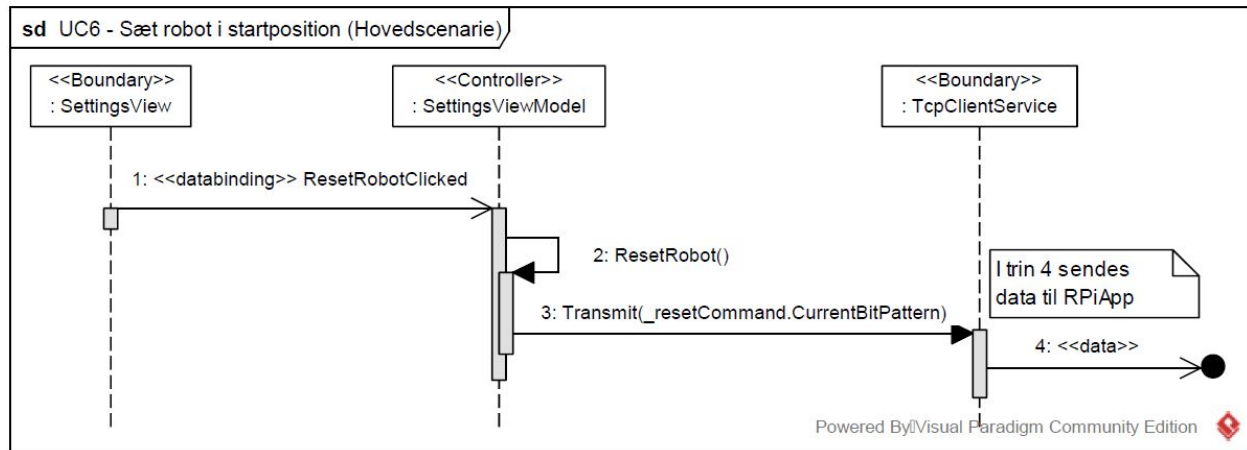
RoboCommand

Samme klasse som beskrevet under applikationsmodellen for UC1 og UC7. Involveres dog ikke direkte i sekvensdiagrammet vist på *figur 22*. Dette skyldes at `SettingsViewModel` instantiere et objekt af denne klasse i sin constructor. Objektets property `CurrentBitPattern` sættes straks via metoden `SetResetRobot`, således objektet er klar til at blive sendt når robotten skal indstilles til sin startposition. Det er således ikke nødvendigt at ændre på objektet senere hen.

8.1.3.2 Sekvensdiagram for hovedscenariet

På *figur 22* ses sekvensdiagrammet for UC6's hovedscenarie. Sekvensdiagrammet adskiller sig fra UC1 og UC7's på den måde, at det ses, at et andet view og en anden view model er i brug.

Det ses også, at domæneklassen *RoboCommand* ikke involveres direkte. Et objekt af denne klasse instantieres i `SettingsViewModels'` constructor, hvorfor objektet's property `CurrentBitPattern` blot kan sendes når brugeren trykker på skærmen for at indstille robotten til dens startposition.



Figur 22 - Sekvensdiagram for UC6's hovedscenarie.

8.2 RPiApp

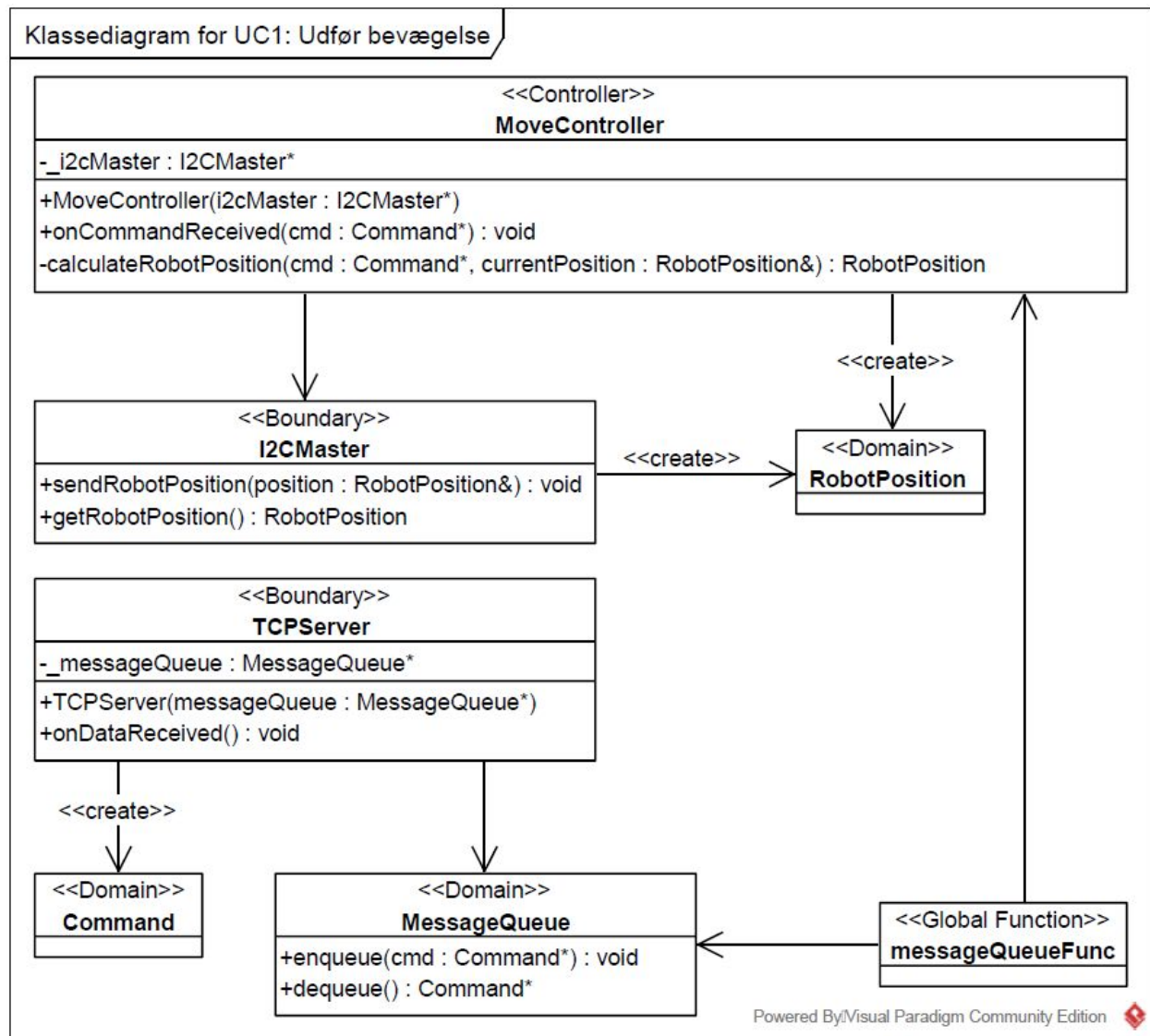
Applikationsmodellen for RPiApp er lavet under den antagelse, at det anvendte programmeringssprog er C++. Fordi RPiApp ikke involveres i nogle af de extensions, som de forskellige use cases har, så er disse selvsagt ikke beskrevet i applikationsmodellerne.

8.2.1 Applikationsmodel for UC1

Applikationsmodellen for RPiApp for UC1 vises i de følgende underafsnit som hhv. klassediagram og sekvensdiagram.

8.2.1.1 Klassediagram

Klassediagrammet for UC1 ses på figur 27. Samme klassediagram benyttes endvidere også for UC6 og UC7. En kort forklaring af de vigtigste klasser ses i forlængelse af diagrammet.



Figur 23 - Klassediagram for UC1's applikationsmodel. Fungerer også som klassediagram for UC6 og UC7.

MoveController

Fungerer som controller for use casen. Klassens onCommandReceived-metode bliver kaldt når der modtages data fra MobilApp. På baggrund af det modtagne data beregner controlleren en RobotPosition, som sendes til PSoCApp via I2CMaster.

I2CMaster

I2CMaster er en boundary-klasse, som benyttes til kommunikation mellem RPiApp og

PSoCApp. Når f.eks. MoveController har en RobotPosition, som skal sendes til PSoCApp, så sendes den via I2CMaster's metode sendRobotPosition(). I2CMaster's metode getRobotPosition() rekvirerer robottens nuværende position fra PSoCApp.

TCPServer

TCPServer er en boundary-klasse, som benyttes til kommunikationen mellem MobilApp og RPiApp. TCPServer modtager data fra MobilApp, som klassen opretter som en Command og efterfølgende lægger i MessageQueue.

Command

Command er en domæne-klasse. Instanser af denne klasse oprettes af instanser af klassen TCPServer på baggrund af det data, som modtages fra MobilApp. Command's lægges i MessageQueue, hvorfra den globale funktion messageQueueFunc trækker dem ud og uddelegerer dem til de respektive controller-klasser.

MessageQueue

MessageQueue er en domæne-klasse, som fungerer som bindeled mellem TCPServer og MoveController (samt øvrige controller-klasser). Command's bliver af TCPServer lagt i MessageQueue, hvorfra Dispatcher trækker dem ud og uddelegerer dem til de respektive controller-klasser.

messageQueueFunc

Her er der tale om en global funktion som kører i en særskilt tråd i en uendelig løkke. Her lyttes der på om der er nogle Command's i kø i MessageQueue og hvis det er tilfældet piller messageQueueFunc Command's ud en efter en og uddelegerer disse til den relevante controller-klasse.

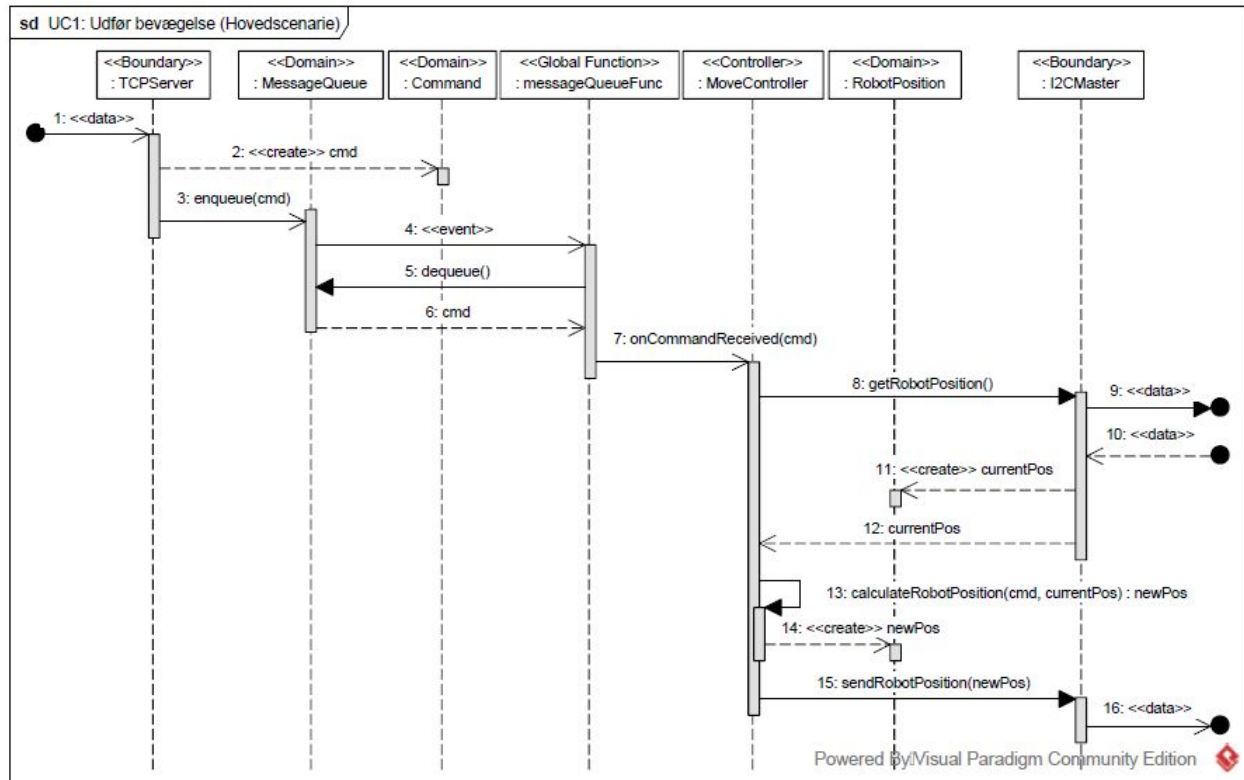
RobotPosition

RobotPosition er en domæne-klasse, som indeholder data om indstilling af robottens motorer. Klassen instantieres af MoveController og sendes til PSoCApp via I2CMaster.

8.2.1.2 Sekvensdiagram for hovedscenariet

På *figur 24* ses sekvensdiagrammet for RPiApp's afvikling af hovedscenariet. I trin 1 modtages data fra MobilApp og i trin 11, 12 og 18 sendes/modtages data til/fra PSoCApp. I trin 3 oprettes

et objekt af klassen Command kaldet cmd. Dette objekt sendes efterfølgende rundt mellem klasserne. På samme vis oprettes objekter af klassen RobotPosition i trin 13 og 16. I trin 4 betyder stereotypen <<event>>, at messageQueueFunc notificeres om, at der findes en Command, der er lagt i kø i MessageQueue, hvorefter Command'en pilles ud af køen og videresendes til MoveController.



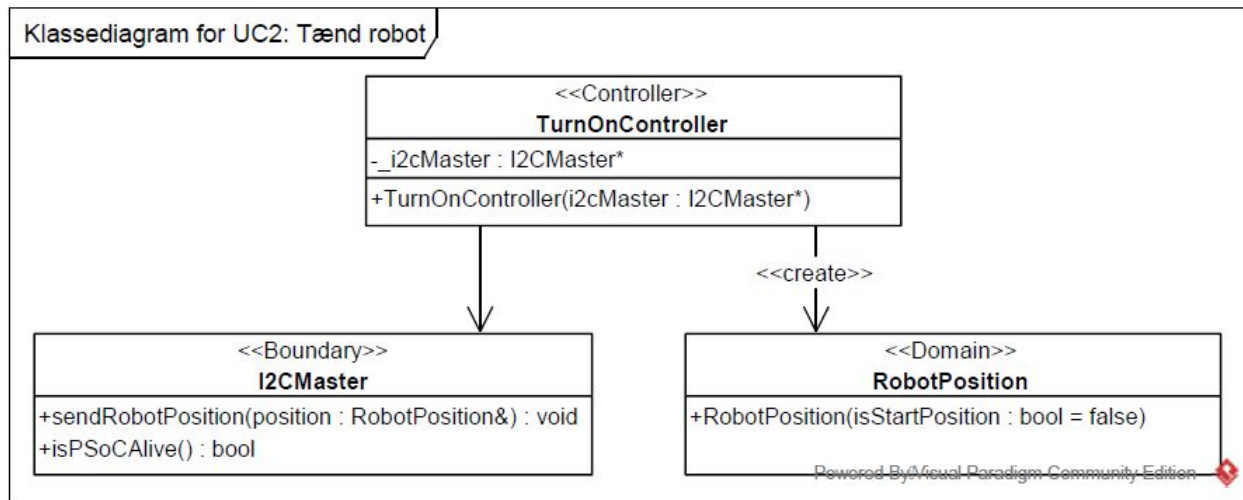
Figur 24 - Sekvensdiagram for UC1's applikationsmodel.

8.2.2 Applikationsmodel for UC2

Applikationsmodellen for UC2 er relativt simpel, idet MobilApp ikke involveres i use casen.

8.2.2.1 Klassediagram

Klassediagrammet for UC2 ses på figur 25. I forlængelse af diagrammet knyttes nogle kommentarer til de enkelte klasser.



Figur 25 - Klassediagram for UC2's applikationsmodel.

TurnOnController

Fungerer som controller for UC2. Klassen har ansvaret for at oprette forbindelse til PSoCApp og for at sende robottens startpositioner til PSoCApp ved opstart af systemet.

I2CMaster

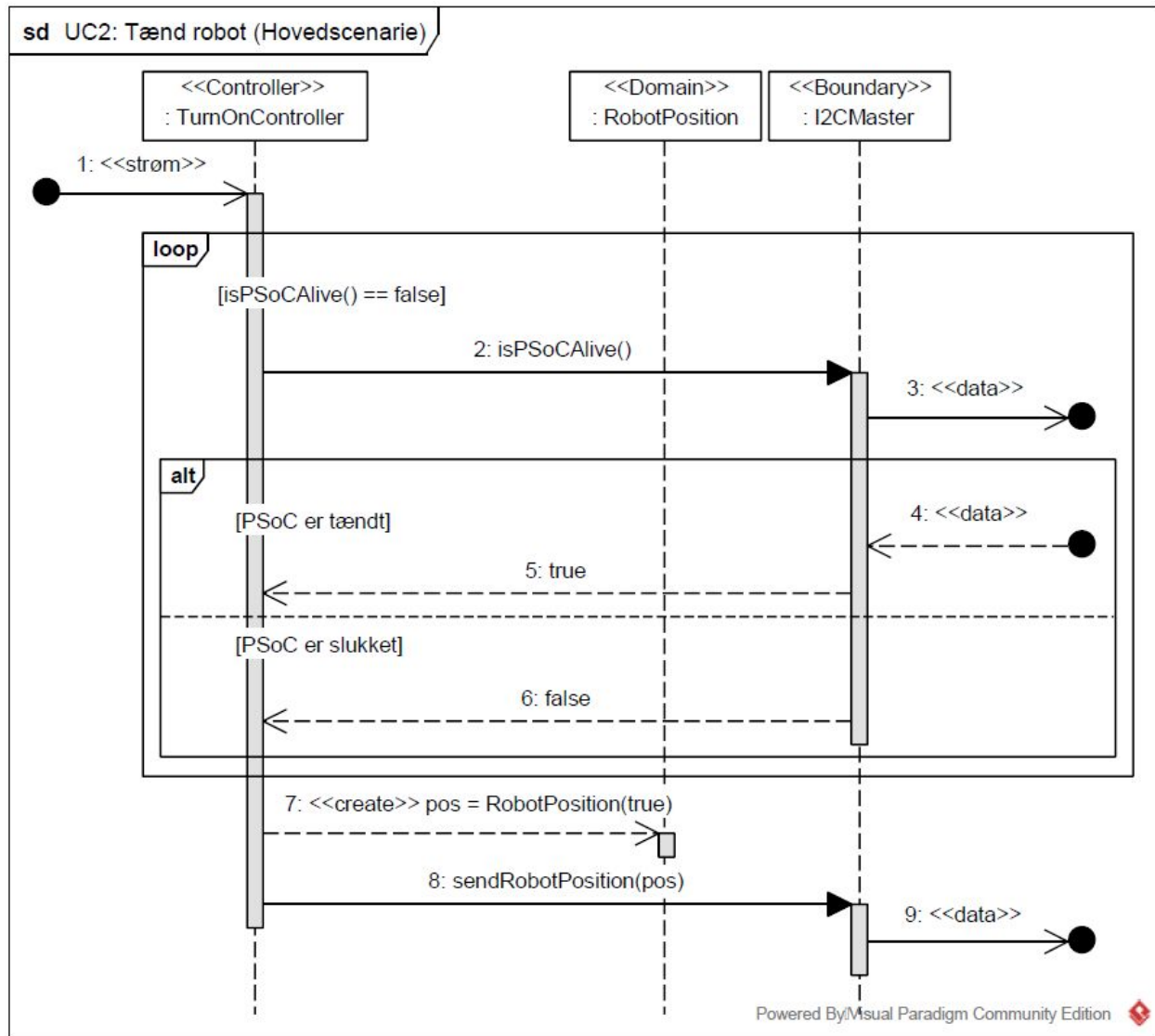
Denne klasse benyttes ligesom i UC1 til at kommunikere med PSoCApp. En metode benævnt isPSoCAlive() benyttes af TurnOnController til at verificere, at PSoCApp er startet op og klar til at modtage kommandoer.

RobotPosition

I modsætning til UC1 så er det i denne use case ligegyldigt hvilken position robottens akser befinder sig i. I stedet kaldes denne klasses constructor med parameteren "true". Dette constructor-kald returnerer et objekt der er initialiseret således, at robottens akser indstilles til deres startpositioner, når det sendes til PSoCApp, jf. protokollen, der er beskrevet i et tidligere afsnit.

8.2.2.2 Sekvensdiagram for hovedscenariet

Sekvensdiagrammet for hovedscenariet ses på *figur 26*.



Figur 26 - Sekvensdiagram for UC2's hovedscenarie. Når RPiApp starter op spørger RPiApp kontinuerligt PSoCApp om den er klar til at modtage kommandoer. Når PSoCApp svarer sendes robottens startposition til PSoCApp'en.

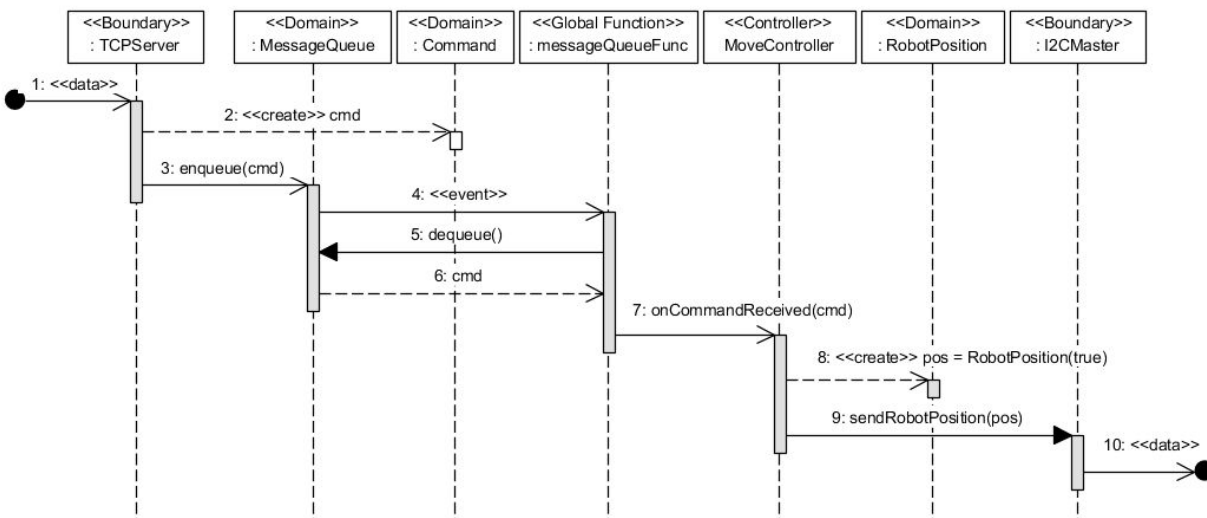
8.2.3 Applikationsmodel for UC6

8.2.3.1 Klassediagram

Klassediagrammet for UC6 er identisk med klassediagrammet for UC1, som ses på figur 27.

8.2.3.2 Sekvensdiagram for hovedscenariet

På *figur 28* ses sekvensdiagrammet for UC6's hovedscenarie. Diagrammet minder om sekvensdiagrammet for UC1. Forskellen er dog, at RPiApp ikke henter robotens akser aktuelle positioner ligesom ingen egentlige beregninger foretages. Robotens startpositioner sendes blot til PSoCApp.



Figur 28 - Sekvensdiagram for UC6's hovedscenarie. I trin 1 modtages data fra MobilApp og i trin 12 sendes data til PSoCApp. I trin 4 notificeres messageQueueFunc om, at køen ikke er tom, hvorfor dequeue() kaldes.

8.2.4 Applikationsmodel for UC7

For RPiApp er denne use case set fra et implementeringsmæssigt perspektiv fuldstændig identisk med UC1. Klassen "MoveController" har således også ansvar for afvikling af denne use case og der henvises derfor til klassediagrammet på *figur 27* og sekvensdiagrammet på *figur 24*.

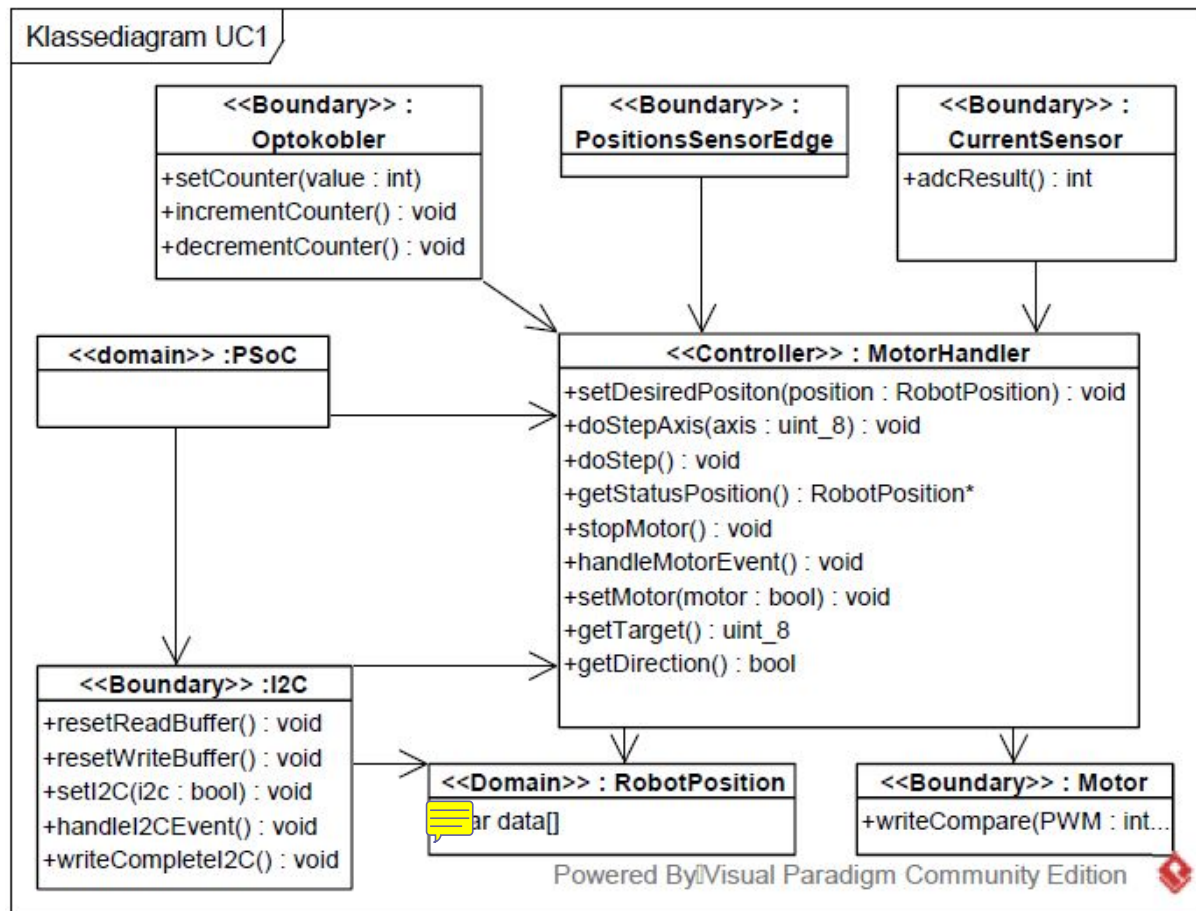
8.3 PSoCApp

Applikationsmodellerne for PSoC vil fremvises først med klassediagrammer, hvori de enkelte klassers formål vil beskrives og derefter anvendes i sekvensdiagrammerne. Det bemærkes at da PSoC anvender C, vil klasserne implementeres som moduler.

8.3.1 Applikationsmodel for UC1

UC1 beskriver scenariet for PSoC App, hvor det vil indstille sig til en given position. Se figur 29.

8.3.1.1 Klassediagram



Figur 29 - Dette er et overordnet klassediagram for systemet, hvor klassen MotorHandler agere som den primære controller. PSoCen primære formål er at dispatche events til systemet.

Som der ses på klassediagrammeret, påvirker langt de fleste moduler MotorHandler. MotorHandler har kun en boundary klasse motor, som den anvender til at sende ud af systemet, som er Motor. De resterende agere som handlers for input i systemet. Disse vil senere beskrives under diverse sekvenser diagrammer.

PSoC

Dette er main løkken i programmet, som kontinuerligt vil kalde `handleMotorEvent` og `handleI2CEvent`.



MotorHandler

MotorHandler står for primært for at styre systemet, ved at den graduelt stepper alle tilsluttet motorer imod en ønskede position. Den modtaget position er modtaget fra RPi. Dette sker ved at PSoC kalder `doStep()`, som stepper hvert enkelt akse 1 grad tættere imod den ønskede position.



I2C

Denne står for at modtage og behandle data fra RPi, valideret jævnført protokollen, og kommunikere disse kommandoer til MotorHandler.

Motor

Motor klassen interagere direkte med det fysiske system, og styre PWM og dermed vinklerne.

RobotPosition

Dette er en data struktur, som indeholder et sæt af 6 vinkler fra 0-180 grader, og en byte af validation.

CurrentSensor

Dette er en sensor modul, der regelmæssigt validere at systemet ikke trækker for meget strøm. Hvis dette er tilfældet, stoppes motoren.

PositionSensorEdge

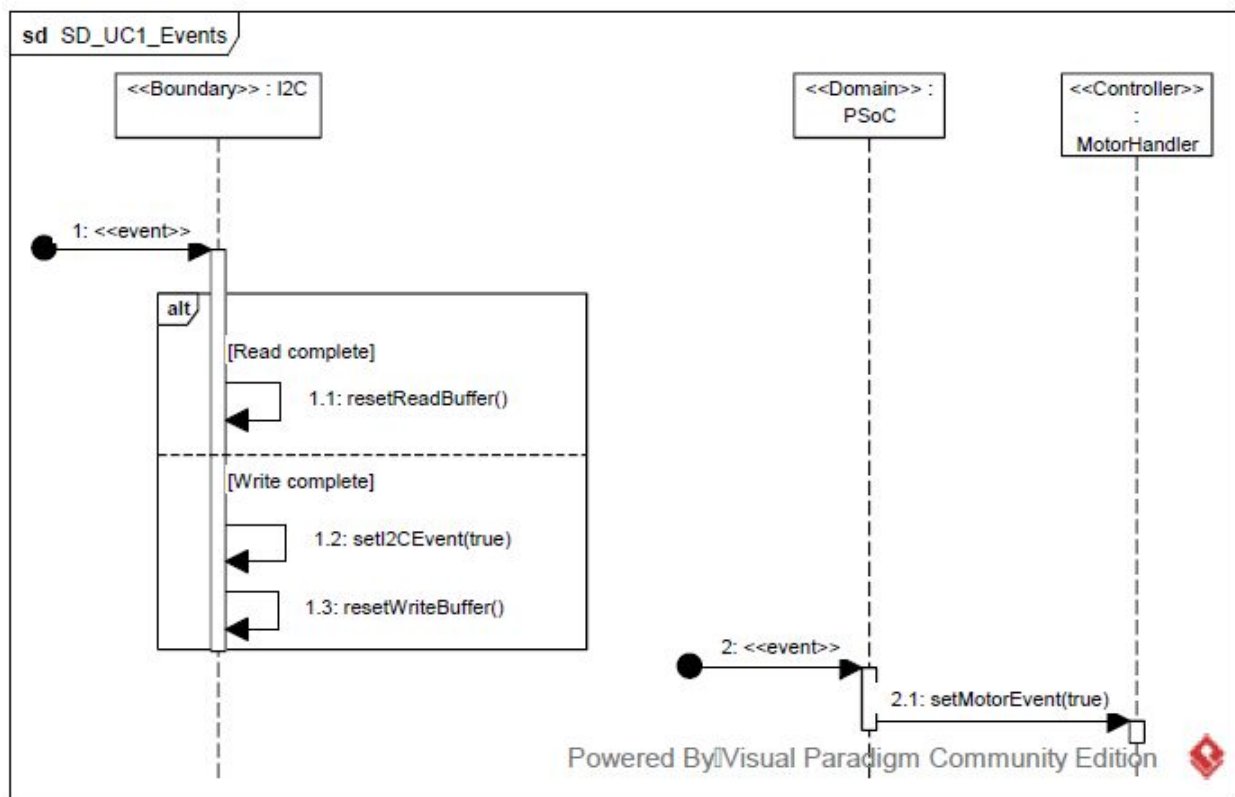
Dette modul står for at detektere om robotten har nået yderpunkterne på 1. Akse, og hvis dette er tilfældet stoppe den.

Optokobler

Dette modul står for at tælle en værdi op eller ned, når denne bliver interruptet. Om den tælles op eller ned afhænger af hvilken vej robotarmen rotere på 1. Akse. Dette gøres forl at vide i hvilken vinkel robotarmen er i.

8.3.1.2 Sekvensdiagram for hovedscenariet

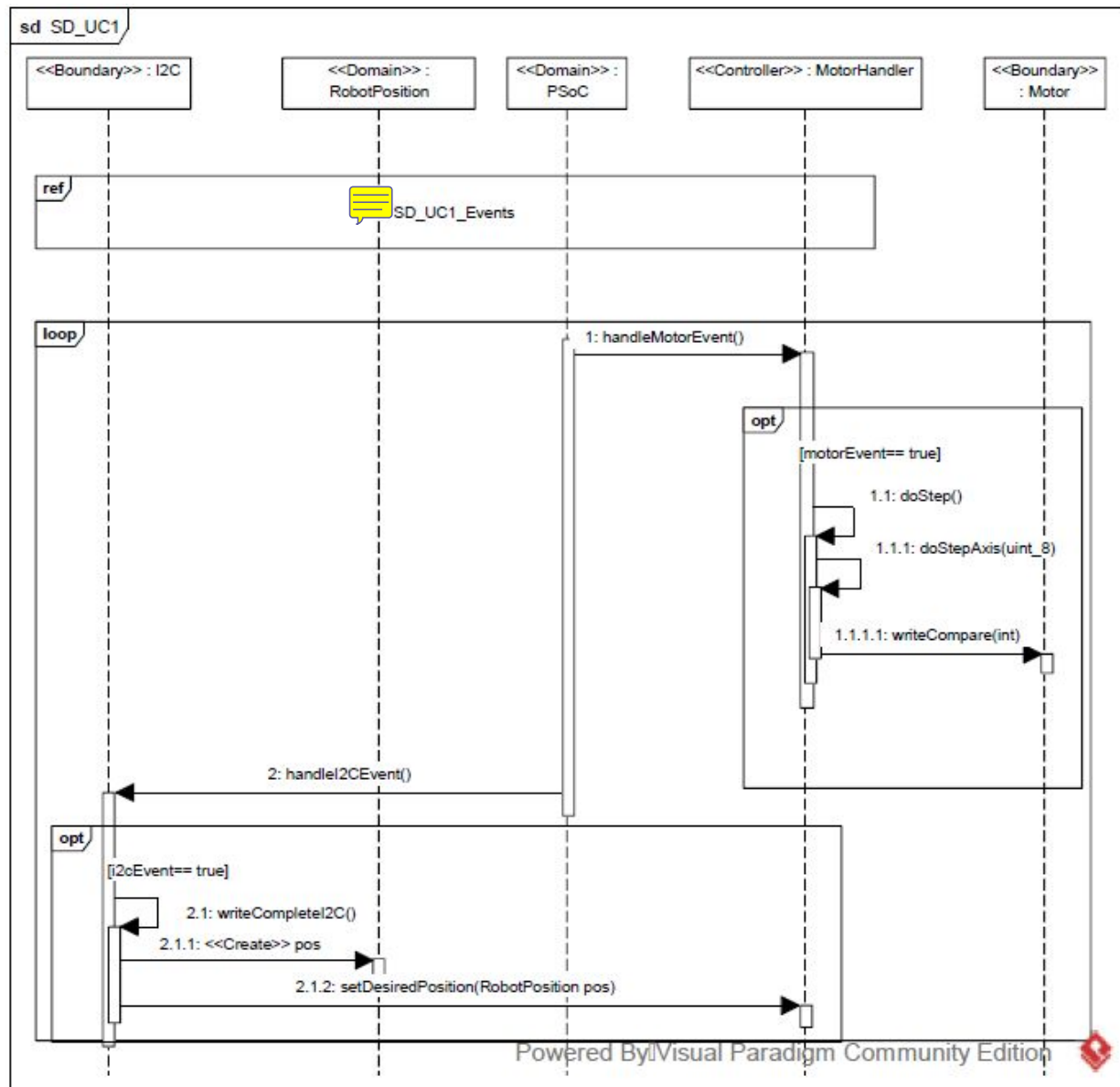
På nedestående figur ses der sekvensdiagrammer for UC1. Denne er opdelt i 2 elementer, events og deres håndtering. På figur *figur 30* ses der hvordan et event kan påvirke systemet. Der findes 2 primære type hændelser, en timer som interrupter 50 gange i sekundet på PSoC, og et interrupt som kaldes når der modtages data fra I2C'en. Disse interrupts eksekvere kun det essentielle kode, og sætter et flag som gør at main eksekvere det nødvendige kode.



Figur 30 - Her ses hvordan flowet er i systemet, og hvornår systemet sætter flag baseret på de modtagne events.

Dette leder videre over i det flow der sker når disse events afvikles.

På denne nedenstående figur kan der ses hvordan flowet i hændelserne bliver håndteret. Se klassebeskrivelser for beskrivelser af hvad formålet er med de individuelle handlinger.



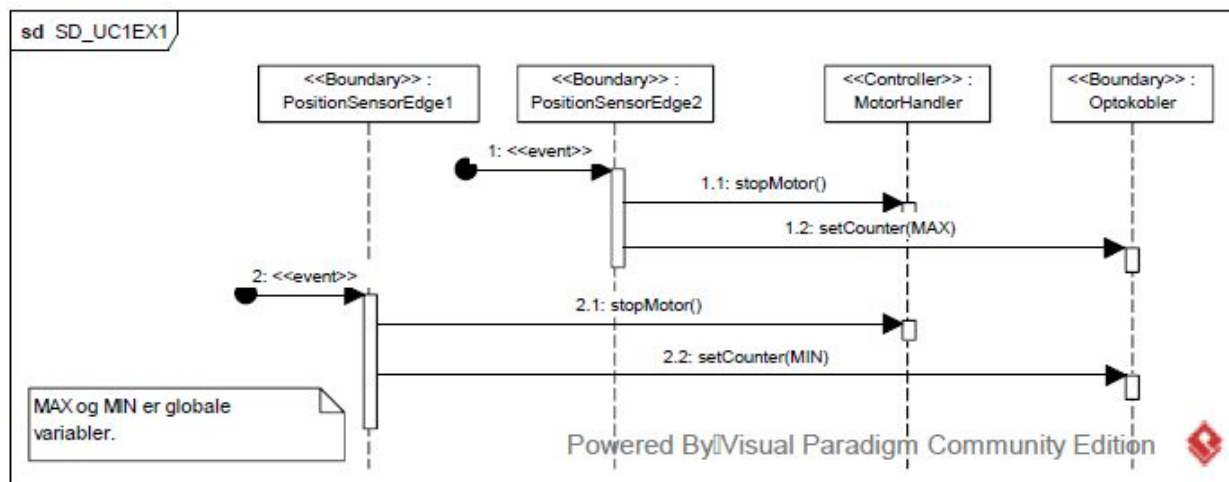
Figur 31 - Dette sekvensdiagram beskriver hvordan systemet håndtere satte flag, og dermed skaber handling i systemet. Efter hver udført "event", resettes flaget.

Alle de følgende events og sekvenser diagrammer til denne use case, beskriver hvordan systemet kan påvirkes af eksterne genstande eller yderpunktet, som kan få en eller flere motorer til at stoppe deres aktive bevægelse.

8.3.1.3 Sekvensdiagram for extension 1(Step Motor)

De følgende diagrammer tager udgangspunkt i at i systemet havde anvendt en stepmotor, og dermed er der behov for at man kan finde ud af hvilken position af motoren. Ved at finde 2 yderpunkter, kan vi beskrive et arbejdsrum som stepmotoren kan bevæge sig indenfor, og dette arbejdsrum kan beskrives i steps af Optokoblerens counter værdi. Hermed kan man i sammenhæng med en calibration, der bliver udført i UC2, koble counter værdien til en aktuel vinkel, således at motoren, kan stoppes når en ønskede vinkel er opnået. Det følgende sekvens diagram viser hvordan systemet kan blive påvirket af positionssensoren.

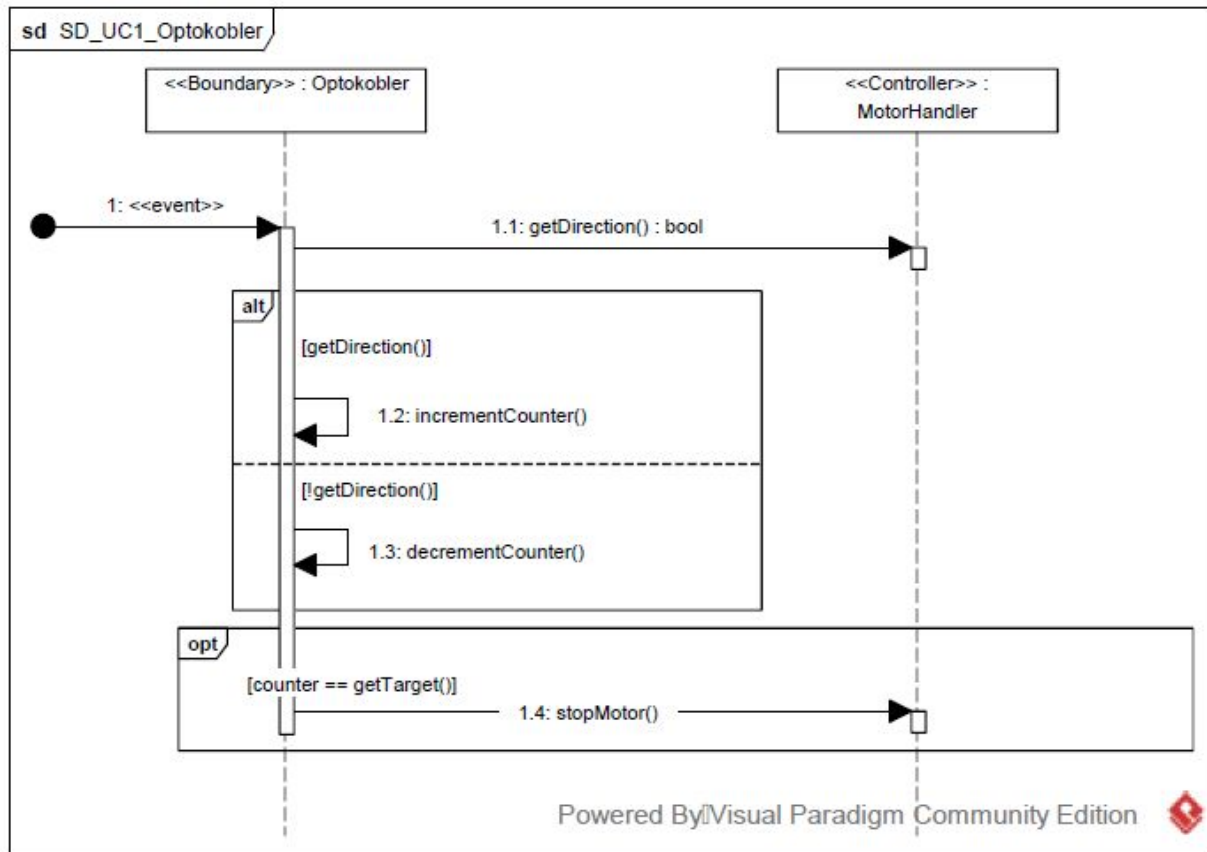
Når positionssensor interruptes betyder dette at en grænseflade er nået, og dermed skal motoren stoppe. Dermed kaldes stopMotor, og Optokoblerens sættes til henholdsvis enten dens max værdi eller min værdi, afhængigt af hvilken Sensor der interruptes.



Figur 32 - Dette sekvensdiagram beskriver hvordan den simuleret stepmotor ville være påvirket af positionssensorer.

Tilhørende til dette diagram er der diagrammet for optokobleren.

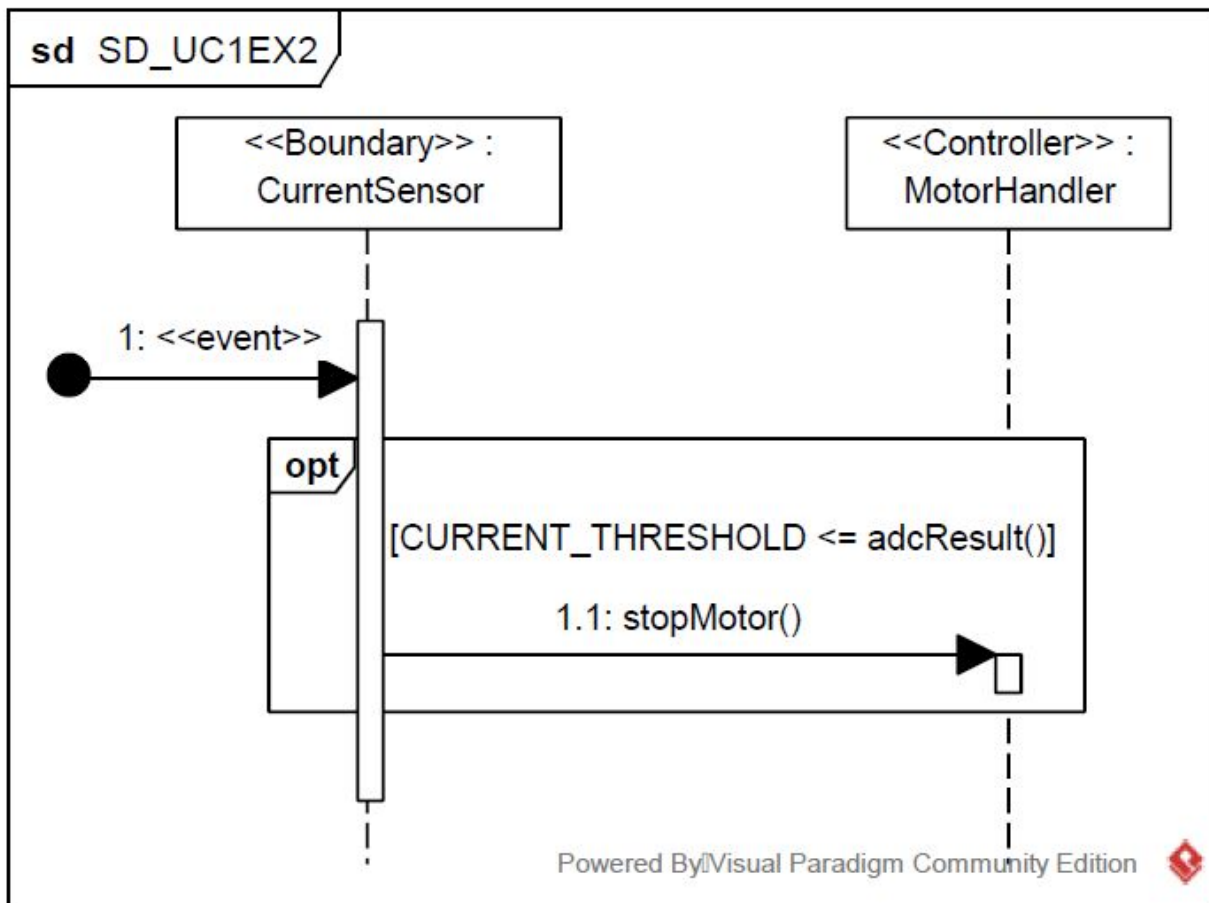
Denne interruptes når 1. Akse rotere, og tæller op hvis den rotere med uret, og ned hvis den rotere mod uret. Hvis counteren stemmer overens med den ønskede position, som getTarget beskriver, værdi, så stoppes første akse.



Figur 33 - Dette sekvensdiagram beskriver hvordan den simuleret stepmotor ville være påvirket af Optokobler.

8.3.1.4 Sekvensdiagram for extension 2

Nedestående sekvensdiagram, *figur 34*, interruptes af færdige ADC SAR konverteringer, hvor resultatet sammenlignes op imod et threshold for at vurdere om en motor skal stoppes.

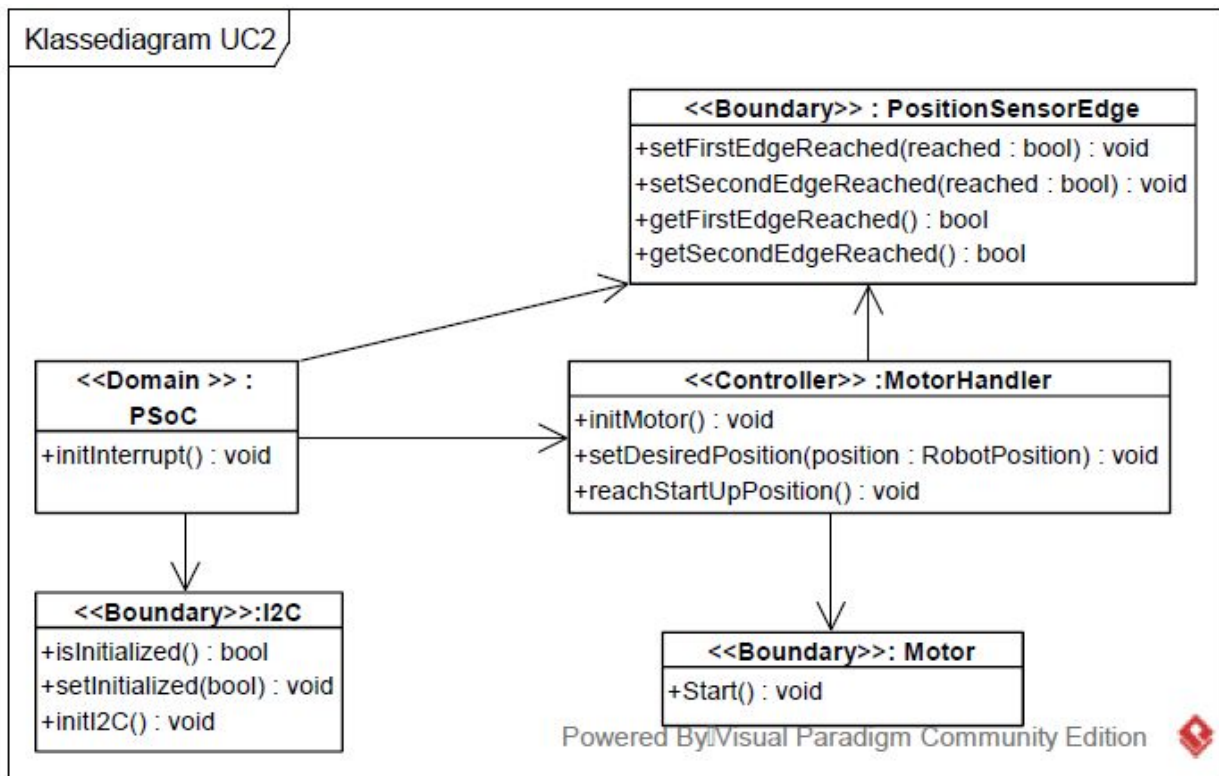


Figur 34 - Dette sekvensdiagram beskriver hvordan systemet stoppes under belastning.

8.3.2 Applikationsmodel for UC2

Denne use case beskriver startup af systemet, og for PSoC hvordan default positionen opnåes, samt kalibration.

8.3.2.1 Klassediagram



Figur 35 - Klassediagrammet for UC2. Beskriver hvordan forskellige moduler er forbundet.

PSoC

Denne klasse står for at initialisere forskellige interrupts og timers, hvilket skaber garanti for at systemet er funktionelt, når I2C kommunikation startes.

MotorHandler

Denne klasse sørger for at alle motorene er indstillet til en default position, 90 grader, og holder denne position til en ny kommando modtages som per use case 1. Herudover afvikler denne også en kalibration for 1. Motor.

PositionSensorEdge

Denne anvendes under kalibrerings sekvensdiagrammet, til at detektere når yderpunkterne er nået.

I2C

Dette modul står for at starte I2C kommunikation, samt at holde resten af systemet i en afventende status til at systemet bliver pinget af RPi App, hvorefter UC1 vil påbegynde.

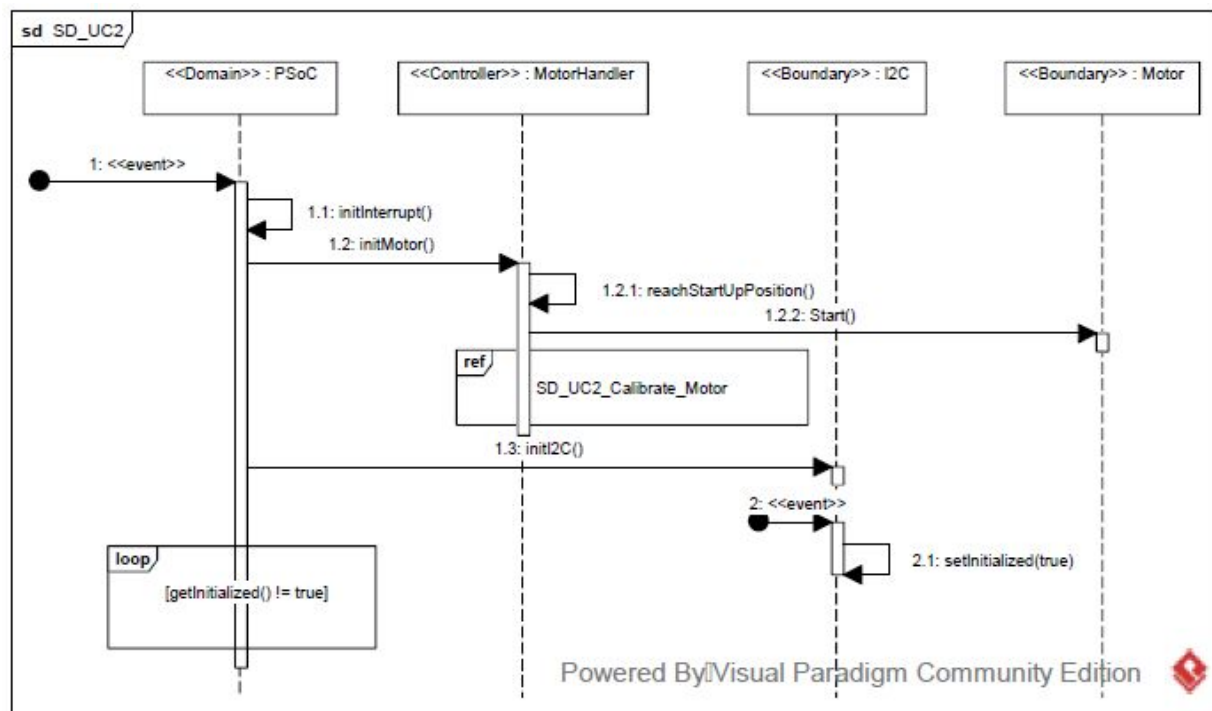
Motor

Dette modul står for at tænde PWM modulerne der styrer motorene. Dette er autogenereret.

8.3.2.2 Sekvensdiagram

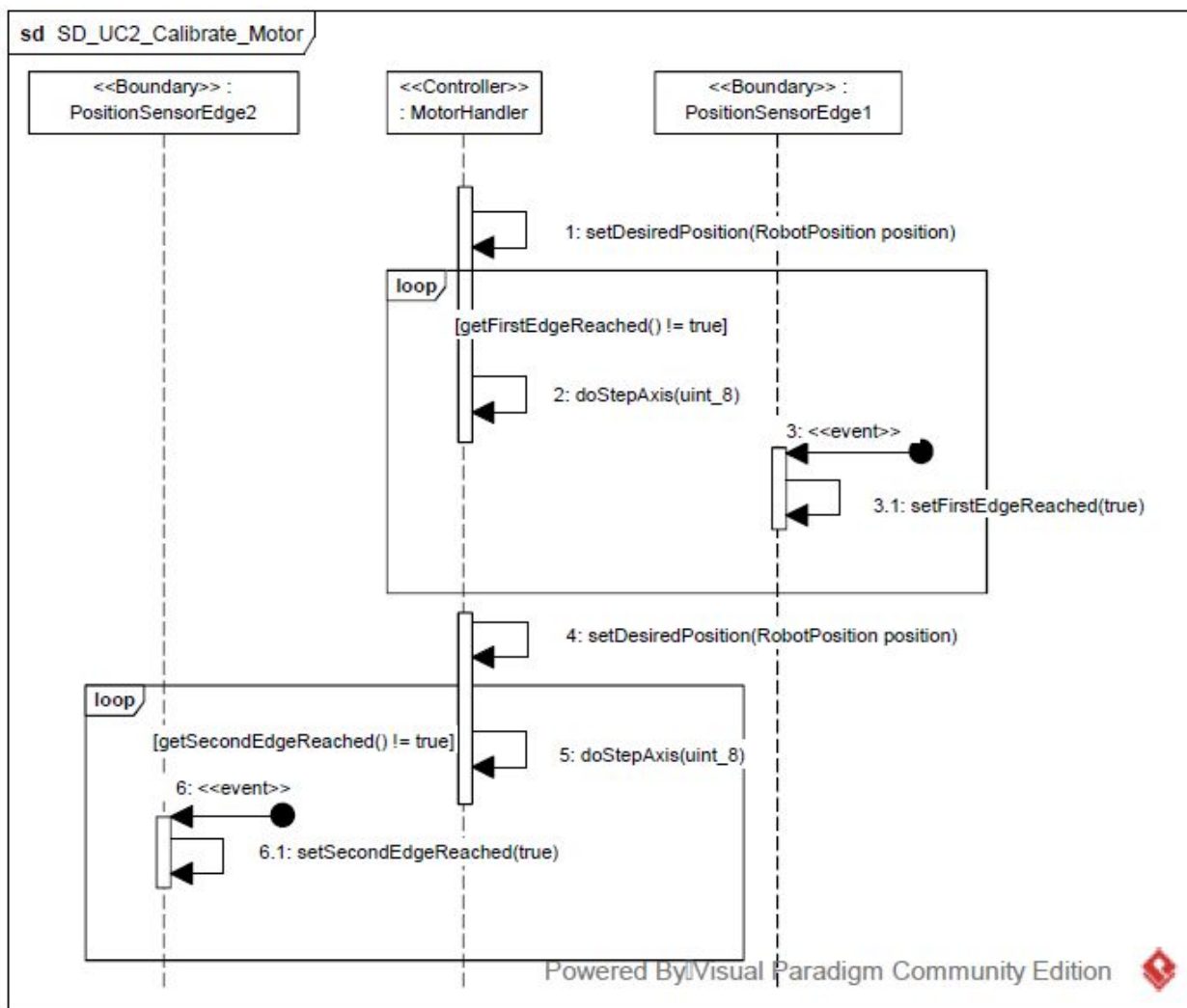
Nedestående sekvens diagram beskriver hvordan systemet bliver initialiseret. Her skal de bemærkes, at denne use case først afsluttes når systemet er blevet pinget af RPi App, igennem det event der er ved punkt 2. Se *figur 38* for sekvens diagrammet der beskriver kalibration.

Herudover består sekvensdiagrammet primært af initialisering af diverse timers og interrupts.



Figur 36 Sekvensdiagram over initialiserings processen af systemet. Her er rækkefølgen vigtig for at brugeren ikke skal afvente at systemet er færdigopstartet.

Det følgende sekvensdiagram kalibrer optokobleren i forhold til MotorHandler. Se *figur 37*, for sekvensdiagrammet der beskriver anvendelsen af dette. Dette gøres ved at først bevæge sig til den ene kant, og derefter tælle hvor langt der er over til den anden kant. Dermed har man et arbejdsrum, som systemet kan bevæge sig indenfor.



Figur 38 - Beskriver hvordan 1. Akse bliver kalibreret, og det er implicit at de fremfundne værdier under kalibrationen gemmes i variabler.

8.3.3 Applikationsmodel for UC6

For PSoCen er denne use case set fra et implementeringsmæssigt perspektiv fuldstændig identisk med UC1, da dette er en almindelig bevægelse. Controller klassen har således også ansvar for afvikling af denne use case og der henvises derfor til sekvensdiagrammet på *figur 31*.

8.3.4 Applikationsmodel for UC7

Tilsvarende for UC7, gælder det samme som for UC6. Her bemærkes dog at hovedscenariet bliver termineret af UC1 extension 2, som registrerer når systemet har grebet fat. Se *figur 34*.