NTNU
Det skapende universitet

NTNU-Cyborg: Oppsøking av sosiale situasjoner

Stine Lilleborge

Master i kybernetikk og robotikk
Innlevert: juni 2015
Hovedveileder: Sverre Hendseth, ITK

Norges teknisk-naturvitenskapelige universitet
Institutt for teknisk kybernetikk

Masteroppgave

Student:	Stine Lilleborge
Fag:	Teknisk kybernetikk, masteroppgave
Norsk tittel:	NTNU-Cyborg: Oppsøking av sosiale situasjoner
Engelsk tittel:	NTNU-Cyborg: Seeking social situations
Veileder:	Sverre Hendseth

Beskrivelse av oppgaven

NTNU-Cyborgprosjektet ble startet høsten 2014, og har som mål å kombinere kloke hoder fra ulike fagfelt på NTNU, for å sammen skape en kyborg (kybernetisk organisme). Kyborgen skal bestå av en robotteknisk kropp med en organisk hjerne/samling nerveceller.

Kyborgen skal opptre sosialt, og vil i starten være mye basert på kunstig intelligens. Etter hvert er det tenkt at nervecellekulturen skal kunne ta over deler av beslutningsstrategien.

Kyborgen skal danne en lærings- og forskningsplatform i forbindelse med blant annet koblingen mellom nevroner og elektronikk.

Motivasjon for oppgaven

Oppsøking av sosiale situasjoner står svært sentralt i forbindelse med sosialisering. Denne funksjonaliteten anses dermed som essensiell for kyborgen, og vil danne et grunnlag for videre utvikling.

Oppgavens omfang

Denne oppgaven tar for seg utformingen av et system som skal gjøre det mulig for kyborgen å oppsøke sosiale situasjoner. Den skal kunne oppdage personer omkring den, velge ut en av dem å bevege seg mot, samt produsere utdata som tilsier hvor den valgte personen befinner seg i bildet. Utdata skal videre kunne benyttes i forbindelse med utvidelse av systemet. Da særlig i forbindelse med utvikling av en posisjonskontroller for kyborgen.

Oppgaven innebærer blant annet:

1. Innføring i Kinect-programmering, samt hvilke muligheter denne programvaren byr på
2. Formulering av kravspesifikasjon og tester for systemet
3. Presentasjon av systemdesign som oppfyller kravspesifikasjonen
4. Implementasjon av systemet
5. Dokumentasjon av det resulterende systemet, slik at arbeidet kan videreføres

Forord

Denne rapporten oppsummerer arbeidet jeg har gjort i forbindelse med min masteroppgave ved Institutt for teknisk kybernetikk ved Norges teknisk-naturvitenskapelige universitet.

Det er veldig rart å tenke på at dette er min siste innlevering som student. Tenk; nå er jeg ferdig! De fem årene har gått så utrolig fort, men de har også vært så utrolig bra. Mye tid har gått med til å være studerende oppe på Gløshaugen, men mye tid har også gått med til å være student nede på Studentersamfundet. Jeg er kjempestolt over å ha fått ta del i en så fantastisk flott studenttradisjon!

Dette siste semesteret har vært sterkt preget av arbeid med masteroppgave, studentassistentstilling, regneøvinger, styreverv i Akademisk Radioklubb, jobbsøking og -intervjuer. Det har med andre ord vært nok å ta seg til, men selv om det har vært slitsomt til tider, har det vært både lærerikt og givende å holde på med alt sammen.

Jeg har trivdes veldig godt med å jobbe med masteroppgaven min, og jeg har lært masse. Jeg har lest meg opp på utviklingsmetoder, eventer og eventhandlere. Jeg har fått prøvd meg på Kinectprogrammering i C#. Jeg har tegnet diagrammer, og øvd meg på å skrive kodedokumentasjon. Utfordringene har vært mange, men de har bare gjort meg enda mer sikker på at det er dette jeg vil jobbe med, og bli enda bedre på!

Avslutningsvis vil jeg først og fremst få rette en stor takk til min veileder, Sverre Hendseth, for masse god hjelp, og ikke minst god tålmodighet. Takk for alle tips og underholdende veiledningsmøter!

Jeg vil også takke Akademisk Radioklubb, for å ha vært et godt sted for avkobling og sosialisering med verdens triveligste folk. En stor takk må også rettes for semesterets lisenskurs og -prøve. Masse 88 de LB0IG!

Sist, men ikke minst, må jeg få takke Trondheim for å være verdens beste studentby. Jeg har blitt så utrolig glad i denne byen, og det er med tungt hjerte jeg må forlate den. Når jeg nå pakker tingene mine for å dra videre nordover mot Bodø, fylles vesker og bager til randen av gode minner, høyt prissatte vennskap, samt masse lærdom og erfaringer. Vi ses igjen, Trondheim. Det lover jeg!

Trondheim, 1. juni 2015
Stine Lilleborge

Sammendrag

Denne rapporten tar for seg arbeidet som er utført i forbindelse med utviklingen av et system som gjør det mulig for en robot å oppdage personer omkring den, og velge ut en av dem som den ønsker å bevege seg mot. Det er til dette benyttet en Microsoft Kinect v2.

Oppgaven innebærer formulering av kravspesifikasjon og konkret kravsliste, utvikling av systemdesign og tester, implementasjon av systemet, og dokumentering av systemet.

Utviklingsmetodikken som er benyttet er en kombinasjon av fossefallsmetoden og iterative metoder. Den iterative delen av metoden kommer frem gjennom at implementasjon og design foregår iterativt og at endringer i kravspesifikasjonen og testing underveis i utviklingen er tillatt.

Det endelige systemet er delt inn i tre moduler; Distribusjonsmodulen, Beslutningsmodulen og Brukergrensesnittmodulen. Distribusjonsmodulen leser bildeinformasjon fra Kinecten, og videreformidler posisjonsinformasjon, fargebilder, dybdebilder og infrarødbilder til henvendende moduler.

Både Beslutningsmodulen og Brukergrensesnittmodulen mottar data fra Distribusjonsmodulen. Beslutningsmodulen mottar posisjonsdata fra Distribusjonsmodulen, som tilsier hvor i bildet en person befinner seg. Dersom det finnes en eller flere personer i bildet, vil Beslutningsmodulen velge ut en av dem, og videreformidle denne personens posisjon til en potensiell posisjonskontroller.

Brukergrensesnittmodulen mottar både posisjonsdata og fargebilder fra Distribusjonsmodulen. Fargebildet vises for brukeren, sammen med et antall firkanter som markerer eventuelle hoder på personer som befinner seg i bildet. Disse firkantene lages ut i fra posisjonsdataene modulen mottar. Det skal da være mulig å gjøre et trykk i en slik firkant for å aktivere valg av person å bevege seg mot.

For det resulterende systemet er automatisk valg av person å bevege seg mot via Beslutningsmodulen ferdig implementert. Manuelt valg via Brukergrensesnittmodulen er derimot ikke fullført. Årsaken til dette er at det oppstod problemer i forbindelse med å motta to typer av informasjon parallelt, samt visning av fargebilder.

Grunnet høy tidsbruk på disse problemene, ble det ikke nok tid til å fullføre hele modulen. Funksjonalitet for å velge person å bevege seg mot er ikke blitt implementert. Videre arbeid går dermed ut på å ferdigstille Brukergrensesnittmodulen, både med tanke på problemene som oppstod og manglende funksjonalitet.

Abstract

This report focuses on documenting the work done during the development of a system for detecting people around a robot, and choosing one of them for the robot to move towards. A Microsoft Kinect v2 has been used in the development.

The assignment that this report summarises includes formulating a requirements specification, developing a design for the system, formulating tests, implementing the system, testing and documentation.

The method used for the system development is a combination of the waterfall method and iterative methods. The method used is iterative in the sense that the design and implementation phases go through several iterations, and also that changes in the system requirements and testing during development is allowed.

The final system is divided into three modules; the Distribution module, the Decision module and the User interface module. The Distribution module reads information from the Kinect and supplies requesting modules with information about the positions of people in the Kinects view, in addition to color, depth and infrared frames.

Both the Decision module and the User interface module receives information from the Distribution module. The Decision module receives information about the positions of people. If there is one or more people in the picture, the Decision module will choose one of them, and send its position information to a potential position controller.

The User interface module receives both information about the positions of people and color frames. The color frame is shown to the user together with a number of squares surrounding the heads of the people in the picture. The positions of the squares are given by the position information that the module receives from the Distribution module. A click inside one of the squares should result in the module sending the position information of the person inside that square to a potential position controller.

For the resulting system the automatic choice of person to move towards is fully implemented. Manual choice, on the other hand, is not completed. Some complications regarding receiving multiple types of information at the same time arose. Showing the color frame also turned out to be harder than expected, while all the attempted solutions resulted in memory leakage.

Future work will focus on finishing the User interface module, while some of the planned functionality is still missing.

Innhold

1 Introduksjon

1.1 Bakgrunn for rapporten

Denne rapporten er et produkt av en masteroppgave utført ved Institutt for teknisk kybernetikk (ITK) ved Norges teknisk-naturvitenskapelige universitet (NTNU) våren 2015.

Oppgaven rapporten presenterer er en del av et større prosjekt, kalt NTNU-Cyborg. Dette er et bioteknologisk satsningsprosjekt, som kombinerer flere fagfelt ved NTNU.

Formålet med NTNU-Cyborgprosjektet er å lage en kybernetisk organisme (kyborg), som skal bestå av en kybernetisk robotkropp og en organisk cellekultur som hjerne. Kyborgen skal opptre sosialt, og interaktere med studenter og ansatte i et bygg på campus Gløshaugen.

ITK har fått ansvaret for å utvikle robotdelen av prosjektet, og startet arbeidet med å planlegge utviklingen høsten 2014 [Lilleborge, 2014] [Nævra, 2014].

1.2 Rapportens omfang

Første trinn i en sosialiseringsprosess, er å oppsøke en sosial situasjon. I henhold til [Lilleborge, 2014], er det dermed blitt bestemt at oppsøking av sosiale situasjoner er en viktig funksjonalitet for en sosial robot. Det er dermed en implementasjon av dette som er formålet med denne oppgaven.

Rapporten presenterer arbeidet som er blitt gjort i forbindelse med utvikling av et system som skal gjøre en sosoial robot i stand til å oppdage mennesker rundt den, samt å velge ut en av dem å bevege seg mot. Systemet skal produsere utdata som tilsier hvor personen som følges befinner seg.

Oppgaven går ut på å formulere kravspesifikasjon for systemet, bestemme systemdesign, samt å implementere, teste og dokumentere det resulterende systemet.

1.3 Disponering av rapporten

Rapporten starter med en introduksjon til tidligere utført arbeid på NTNU-Cyborgprosjektet, samt en innføring i rammebetingelsene for prosjektet.

Videre gir rapporten leseren et innblikk i ulike teoretiske tema det blir aktuelt å ha kjennskap til videre i rapporten, blant annet eventer og eventhandlere, samt en introduksjon til programmering med Microsoft Kinect v2

i C#.

Etter dette følger en presentasjon av arbeidsmetodikken som er benyttet i arbeidet med denne rapporten. Her ligger begrunnelse for valg av utviklingsmetodikk, illustrasjon som viser tidsplanlegging, samt valg av programmeringsspråk.

Selve systemdokumentasjonen starter med utforming av kravspesifikasjonen. Her finnes blant annet systembeskrivelse med illustrasjon av systemet, rammekrav, funksjonelle egenskaper, samt en nummerert liste over konkrete krav som stilles til systemet.

Systemdokumentasjonen fortsetter så med en formulering av systemdesign, inndelt etter systemmodulene.

Deretter følger kommentarer til implementasjonen, der det gjøres rede for eventuelle hindre som har oppstått.

Videre gis det en oversikt over systemtestene, som skal kjøres i verifiseringsavsnittet. Dette etterfølges av nettopp verifiseringsavsnittet, som går gjennom kravene formulert i kravspesifikasjonen.

Mot slutten følger diskusjon, der eventuelle forbedringer, både med tanke på arbeidsprosess og selve utviklingen, trekkes frem.

Rapporten avsluttes med en konklusjon, som tar for seg videre arbeid.

2 Bakgrunn

2.1 Tidligere utført arbeid

Høsten 2014 ble det gjennomført to prosjektoppgaver på NTNU-Cyborgprosjektet ved ITK.

[Lilleborge, 2014] tar for seg utviklingen av en funksjonsspesifikasjon for en sosial robot. Ulike funksjoner blir her vurdert og rangert relativt til hverandre med tanke på tre vurderingskriterier;

- implementasjonshardhet: hvor vanskelig er det å implementere?
- nyttegrad: hvor nyttig er funksjonen i seg selv? Vil den komme til nytte på egenhånd, eller er den avhengig av andre funksjoner for å kunne utnyttes maksimalt?
- knaggpotensiale: i hvilken grad fungerer denne funksjonen som en "knagg" for andre funksjoner? Er andre funksjoner avhengig av denne funksjonen?

Resultatet er en prioritetsliste over funksjoner rangert etter foretrukket implementasjonsrekkefølge. Listen er gjengitt i Tabell 1

[Nævra, 2014] ser på mulige forflytningsmåter for en sosial robot, og er en studie av ulike robotbaser som eksisterer på markedet. Rapporten konkluderer med å gå til innkjøp av en Pioneer LX Research Platform [Mobilerobots, 2015] fra Adept, Figur 1.

2.2 Overordnet systemstruktur

Systemet som skal utvikles i denne oppgaven skal være del av et større modulbasert system. Totalsystemet kan dermed skisseres som vist i Figur 2. Hver modul kjører her, teoretisk sett, på hver sin IP-adresse, og kommuniserer med hverandre og robotbasen via TCP.

Kinecten er koblet direkte til robotbasen, som mottar informasjon fra Kinecten og videreformidler relevant informasjon til de ulike modulene.

Det som her omtales som robotbasen, vil være selve robotbasen medregnet eksterne maskiner som plasseres på den. Systemet som planlegges i denne

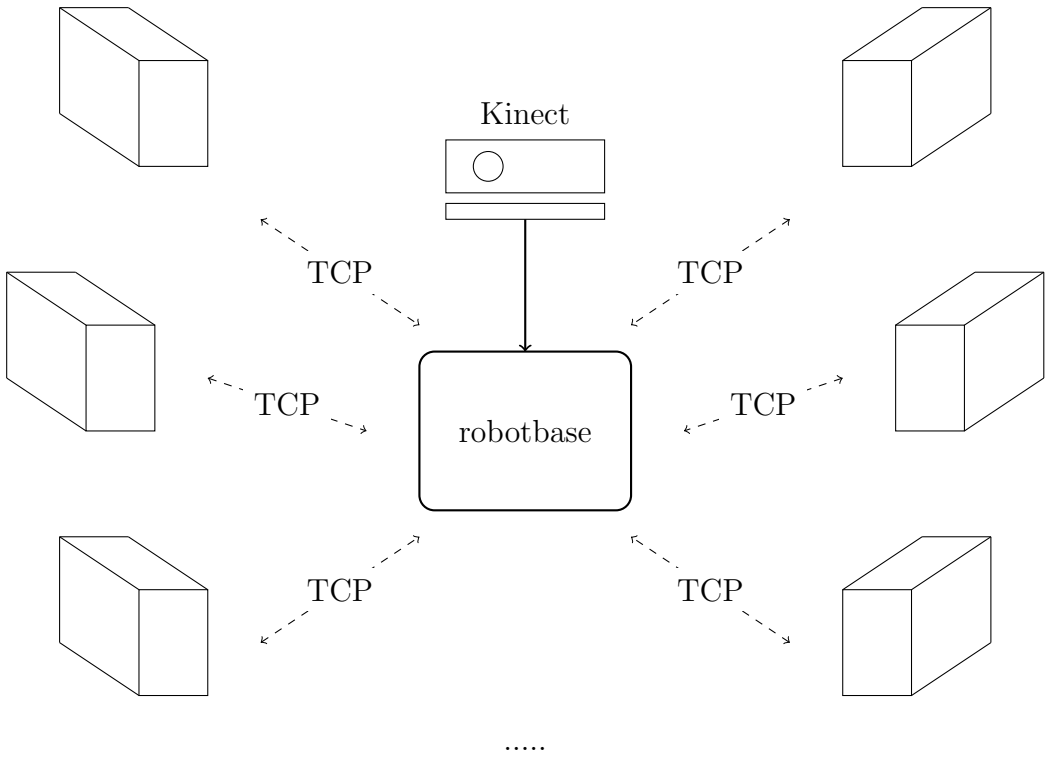
Tabell 1: Prioritetsliste over funksjonalitet for sosial robot [Lilleborge, 2014].

#	Funksjonalitet
1	Forflytning
2	Ansiktsgjenkjenning
3	Bevege seg mot folk
4	Persongjenkjenning
5	Talesyntese
6	Vitsedatabase
7	Sosiale medier
8	Selfies
9	Talegjenkjenning
10	Verbal kommunikasjon
11	Applikasjoner
12	Spørsmålsbesvaring
13	Humørgjenkjenning
14	Måle respons

Figur 1: Pioneer LX – Roboten det er valgt å gå til innkjøp av [Mobilerobots, 2015].

rapporten, er representert ved en av modulblokkene omkring robotbasen vist i Figur 2.

Det at systemet er modulbasert skal gjøre det enklere å erstatte/oppdatere



Figur 2: Systemskjema for totalsystemet.

moduler etterhvert som prosjektet videreutvilkes.

2.3 Rammebetingelser

For å gjøre det mulig for roboten å se omgivelsene sine, trengs det et kamera. Denne oppgaven kan egentlig utføres av et hvilket som helst kamera, for eksempel et ordinært web-kamera. Da dette ville medført uhensiktsmessig mye *overhead*, er det blitt besluttet å benytte en Microsoft Kinect v2, som med egen SDK inneholder mye innebygd funksjonalitet.

3 Teori

3.1 Utviklingsmodeller

De etterfølgende underkapitlene er basert på[Ribu, 2004] og [Hansen and Hjertø, 2006].

3.1.1 Fossefallsmetoden

Fossefallsmetoden baserer seg på en sekvensiell designprosess i fem trinn [WordPress & Atahualpa, 2015] :

1. Formulering av krav
2. Design
3. Implementasjon
4. Verifisering
5. Vedlikehold

Først når det foregående trinnet er fullført, kan designprosessen gå videre til det etterfølgende trinnet. Når prosessen først har gått et trinn videre, kan den ikke gå tilbake.

Den første fasen tar for seg formulering av krav, og resulterer i en kravspesifikasjon. Kravformuleringsfasen etterfølges av designfasen. Eventuelle mangler eller feil i kravspesifikasjonen som oppdages i denne fasen kan etter modellens oppsett ikke rettes opp i. Det vil si at det, dersom modellen skal følges nøyaktig, vil være vanskelig å rette opp i feil som oppdages utover i utviklingsprosessen. Det er altså ingen vei tilbake når det først er tatt et skritt videre i prosessen.

Verifiseringsfasen kommer etter implementasjonsfasen, hvilket vil si at systemet ikke blir testet før hele implementasjonen er fullført. Feil som oppdages i verifiseringsfasen, vil det dermed ikke være mulig å få gjort noe med før i vedlikeholdsfasen.

Det kreves derfor god kontakt mellom kunde og utvikler under formulering av kravspesifikasjon, slik at den blir godt formulert, og det resulterende systemet blir nøyaktig slik kunden ønsker.

Det kan være vanskelig å få formulert en fullstendig og feilfri kravspesifikasjon ved første forsøk. Ofte vil det kunne dukke opp uforutsette problemer

underveis i både design- og implementasjonsfasen, og valg som må tas underveis i utviklingsprosessen. For store prosjekter kan det derfor være risikabelt å følge fossefallsmodellen.

Av dette kan det konkluderes at fossefallsmetoden egner seg best for små, kortvarige prosjekt, der det er god kontakt mellom kunde og utvikler. For slike prosesser er det mer sannsynlig at alle krav kan settes satt ved starten av prosjektet, selv om det uansett er risikabelt.

3.1.2 Iterative metoder

Et alternativ til fossefallsmetoden, er såkalte iterative metoder. Inndelingen i faser er de samme som for fossefallsmodellen, men, som navnet tilsier, går systemutviklingen gjennom hele designprosessen gjentatte ganger. Hver gjennomgang kalles da for en iterasjon. På denne måten vil eventuelle feil som måtte oppstå underveis i en iterasjon kunne bli tatt hensyn til ved neste iterasjon.

Kravspesifikasjonen formuleres fortsatt ved oppstart av prosjektet, men det er mulig å gjøre endringer i den ved begynnelsen av hver iterasjon.

Resultatet fra en iterasjon vil alltid være en velfungerende, kjørbar versjon av systemet, der hver iterasjon tilegner systemet nye funksjoner.

Iterative metoder er fordelaktige ved at de gjør det mulig å rette opp i feil som oppstår underveis. Samtidig gjør de det mulig å kjøre tester på systemet underveis i utviklingen, slik at kunden kan gi tilbakemelding på om systemet blir som forventet.

3.1.3 Evolusjonære metoder

Evolusjonære metoder er enda mer fleksible enn iterative metoder. I tillegg til å gjøre design og implementasjon i flere iterasjoner, blir også formuleringen av kravspesifikasjonen gjort i flere iterasjoner. Det kan dermed sies at hele totalsystemet evolverer utover i utviklingsprosessen.

Fordelen med dette er at det er enda mindre fokus på å formulere kravspesifikasjonen ved oppstart av prosjektet. Iterative metoder tillater at kravspesifikasjonen modifiseres underveis i utviklingsprosessen, mens evolusjonære metoder tilsier at den skal endres/videreutvikles underveis.

3.1.4 Smidig utvikling

I smidig utvikling er det fokus på kundesamarbeid og fungerende kode fremfor omfattende dokumentasjon. Fokuset ligger på å levere velfungerende kode til kunden, og at koden som leveres svarer til kundens ønsker. Endringer er derfor velkomment gjennom hele utviklingsprosessen.

3.2 SINTEFs standard for kravspesifikasjon

SINTEF deler kravspesifikasjonen inn i ti deler [Mikalsen, 1999]:

1. Innledning

Inneholder en presentasjon av systemet som skal utvikles, samt en introduksjon til kravspesifikasjonens oppbygning.

2. Overordnet systembeskrivelse

Beskriver hensikten med systemet. Dersom systemet som skal utvikles er en forbedring av et allerede eksisterende system, skal eventuelle endringer som innføres i det nye systemet presenteres her.

En oversiktsfigur som viser oppdeling i delsystemer og kommunikasjonen mellom dem kan også presenteres her.

3. Rammekrav

Presentasjon av rammekrav for totalsystemet:

- tilgjengelighet – åpningstiden for systemet, maksimaltid for driftsstans, behov for helgekjøring.
- sikring mot tap eller ødeleggelse av data.
- sikring mot tyveri eller misbruk av data – adgangskontroll, personverntiltak.
- kapasitet.
- fremtidig utvidelse av systemet.
- tidsfaktorer. Svartider, oppdateringstider, overførings- og kommunikasjonstider, gjennomløpstider m.v.
- brukervennlighet.
- arbeidslokale og miljø. Støysvakt utstyr, skal kunne utplasseres i kontormiljø.
- nøyaktighet.

Listen er tatt direkte fra [Mikalsen, 1999].

4. Systemets funksjonelle egenskaper

For hvert delsystem skal det gis en beskrivelse av hva det skal gjøre, samt en beskrivelse av inn- og utdata til/fra delsystemet.

5. Logisk datamodell

"Beskriv den logiske sammenhengen mellom datasettene ved hjelp av datastrukturdiagrammer" [Mikalsen, 1999].

6. Krav til systemkonstruksjon

Dersom det finnes begrensninger i allerede eksisterende maskin- eller programvare som må tas hensyn til under systemutviklingen, skal dette presiseres her.

7. Krav til dokumentasjon
Inneholder en beskrivelse av hva systemdokumentasjonen skal inneholde.

8. Krav til manuelle funksjoner
Beskrivelse av delsystemer som innebærer full eller delvis styring av en operatør. I tillegg til en innføring i hva delsystemet skal gjøre, skal det også blant annet gis informasjon om eventuelle inn- og utdata til/fra delsystemet.

9. Krav til opplæring i bruk av systemet
Beskriver opplæring som må gis til eventuelle operatører av systemet.

10. Regler for godkjenningsprøve
Inneholder formulering av tester, både for system og dokumentasjon, som må godkjennes før utviklingen kan anses å være fullført.

3.3 Eventer og EventHandlere

Dersom det er ønskelig at hendelser som oppstår på en plass i koden skal håndteres på en annen plass i koden, kan *eventer* og *event handlere* være et nyttig verktøy.

Hensikten med eventer er å videreformidle fra et sted i koden til et annet at en hendelse har inntruffet. Når et spesielt event inntreffer, er det ofte ønskelig å håndtere det ved å kjøre egendefinert kode. Event handlere muliggjør dette.

3.3.1 Hvordan fungerer det?

I forbindelse med å forklare hvordan event-håndtering fungerer, står begrepet *delegater* svært sentralt. Som forklart i [StackOverflow, 2009], er et delegat en peker til en funksjon. Et delegat har en tilhørende signatur, bestående av returtype og parameterliste, og denne må samsvare med eventuelle funksjoner delegatet skal peke til.

Et event inneholder en liste over delegater, som peker til funksjoner. Når et event av en gitt type blir opprettet, går det gjennom lista med eventets delegater, og kaller på alle funksjonene [StackOverflow, 2009].

3.3.2 Forhåndsdefinerte eventer i C#

Det finnes en rekke forhåndsdefinerte event i C#. Sentrale eksempler er vist i Tabell 2.

Tabell 2: Event med tilhørende argumenter og objekter de oppstår i.

Event	Argument	Oppstår i
MouseLeftButtonDown	MouseButtonEventArgs	Image
FrameArrived	ColorFrameArrivedEventArgs	ColorFrameReader
FrameArrived	DepthFrameArrivedEventArgs	DepthFrameReader
FrameArrived	InfraredFrameArrivedEventArgs	InfraredFrameReader
FrameArrived	BodyFrameArrivedEventArgs	BodyFrameReader

For å knytte en event handler opp mot et event, må det opprettes et objekt av typen eventet tilhører. Det kan så legges til en event handler til dette eventobjektet, med et tilhørende delegat. Når eventet inntreffer, vil funksjonen delegatet peker til bli kalt.

Et eksempel for et FrameArrived-event på en ColorFrameReader er vist i Figur 3.

```
ColorFrameReader colorReader = sensor.ColorFrameSource.OpenReader();
colorReader.FrameArrived += Reader_ColorFrameArrived;

void Reader_ColorFrameArrived(object sender, ColorFrameArrivedEventArgs args)
{
    //kode som ønskes kjørt
}
```

Figur 3: Eksempel for håndtering av FrameArrived-event for et ColorFrameReader-objekt.

3.3.3 Egendefinerte eventer i C#

C# gir også mulighet for å lage egne eventer. Dette kan være nyttig i forbindelse med informasjonsøverføring mellom klasser i/deler av et program.

For å lettere kunne illustrere hvordan egendefinerte eventer kan opprettes, er det valgt å ta utgangspunkt i et eksempelprogram. Eksempelprogrammet

tar for seg arbeid med og innlevering av en masteroppgave. Det finnes to klasser i programmet; en for arbeid med oppgaven (*MasterThesis*) og en for innlevering av oppgaven (*ThesisSubmitter*). I *MasterThesis* finnes det en datovariabel for innleveringsfrist, samt en funksjon for å gjøre arbeid på oppgaven. Når arbeidet med oppgaven har løpt såpass lenge at innleveringsfristen er nådd, settes det opp et event av typen *deadlineReached* i *MasterThesis*. Dette fanges opp av klassen *ThesisSubmitter*, som så leverer inn oppgaven. Eksempelet kan ses i sin helhet i elektronisk vedlegg som "eventeksempelkode.cs".

Et nytt event deklareres inne i et *interface* [Microsoft, 2015c]. Et interface kan inneholder signaturer for blant annet funksjoner og eventer, men ikke selve implementasjonen av dem. Et interface kan sammenlignes med det som i C++ kalles for en *abstakt* klasse. En abstrakt klasse fungerer som en slags baseklasse, med abstrakte medlemsfunksjoner. Abstrakte funksjoner **må** implementeres for alle klasser som arver fra den abstrakte klassen. Det samme gjelder altså for en klasse som *implementerer* et interface; alle funksjonene / eventene som deklareres i interfacet må implementeres i klassen.

Interfacet inneholder også deklarasjon av eventuelle eventargumenter som gjelder spesielt for det deklarerte eventet. I forbindelse med eksempelprogrammet vil koden i interfacet bli som vist i Figur 4.

```
public Interface IReportDeadline
{
  event EventHandler OnDeadlineReached;

  class DeadlineReachedEventArgs : EventArgs
  {
    //include classvariable(s) and constructor(s)
  }

}
```

Figur 4: Interfacet IReportDeadline inneholder deklarasjon av event handler og eventargumenter for eventet.

Deklarasjonen av eventet gjøres altså i et interface, mens implementasjonen gjøres i klassen som skal sette opp eventet når det inntreffer. Implementasjonen innebærer å lage et delegat, samt å sette opp en lås for eventet. Dette gjøres ved hjelp av et objekt av variabeltypen *object*. I forbindelse med eksempelprogrammet vil koden bli som vist i Figur 5.

```
event EventHandler IReportDeadline
{
  add
  {
    lock(deadlineLock)
    {
      deadlineReached += value;
    }
  }
  remove
  {
    lock(deadlineLock)
    {
      deadlineReached -= value;
    }
  }
}
```

Figur 5: Lås og delegat for event handleren i IReportDeadline-interfacet. En funksjon (representert ved value) legges til delegatet ved +=, og fjernes fra delegatet ved -=.

Etter dette gjenstår det kun å sette i gang eventet på ønsket sted i koden. I forbindelse med eksempelprogrammet vil koden bli som vist i Figur 6. Eventet kan fanges opp av andre klasser på samme måte som vist i Figur 2 i Delkapittel 3.3.2.

```
EventHandler deadlineHandler = deadlineReached;

if(deadlineHandler != null)
{
  deadlineHandler(this, new EventArgs());
}
```

Figur 6: Illustrasjon av hvordan et event settes igang.

3.4 Mean-shift algoritmen

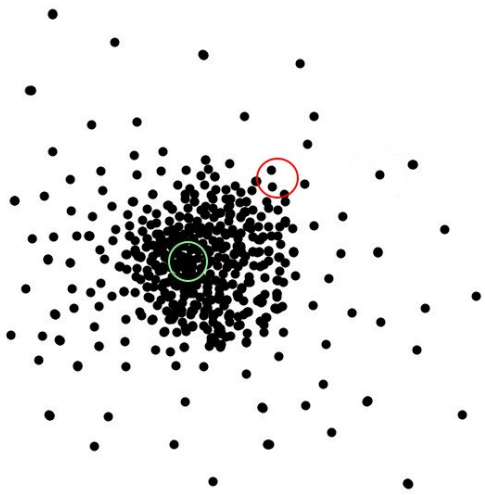
Mean-shift algoritmen er en iterativ metode for lokalisering av maksimum i en tetthetsfunksjon
[Thirumuruganathan, 2010]. Algoritmen ser på datapunkter i det aktuelle dataområdet som en sannsynlighetstetthetsfunksjon. Områder med høy tett-

hetsgrad av datapunkter ses dermed på som maksimum for funksjonen, mens områder med lav tetthetsgrad av datapunkter anses som minimum.

Metoden baserer seg på å velge ut et tilfeldig initialpunkt som aktivt punkt, for så å utføre tre enkle operasjoner gjentatte ganger inntil det oppnås konvergens:

1. definer et område rundt datapunktet
2. kalkuler maksimum internt i området
3. bytt aktivt punkt til det interne maksimumet

Figur 7 illustrerer algoritmens start- og sluttpunkt.



Figur 7: Den røde sirkelen markerer startpunktet for mean-shift algoritmen, mens den grønne sirkelen markerer resultatet.

Mean-shift algoritmen brukes blant annet i datasyn. Algoritmen kan brukes til for eksempel å bestemme hvilke punkter som inngår i et *cluster* [1]. Alle datapunkter som leder til ett felles maksimum, tilhører det samme *clusteret* [Collins, 2006].

I tillegg kan mean-shift algoritmen brukes i forbindelse med sporing. Sporingen gjøres på grunnlag av et histogram for fargefordeling i et objekt. Ved

[1] Samling av datapunker.

å ta utgangspunkt i objektets posisjon i det nåværende bildet, kan det så gjøres et søk i det omkringliggende området i det etterfølgende bildet. Meanshift brukes da for å finne området med samsvarende fargefordeling som i histogrammet [Collins, 2006].

3.5 Kinect v2 sensor

Dersom annet ikke er angitt, er informasjonen som er gjengitt i dette kapittelet basert på det som står på Microsofts nettsider om utvikling med Kinect v2 i Windows [Microsoft, 2015d].

Microsofts Kinect er originalt et tilbehør til Xbox som benyttes i forbindelse med kroppsstyring i spill. Da for eksempel idretts- eller dansespill. Kinecten inneholder, i tillegg til et HD videokamera, en infrarødsender og en mikrofonrekke.

Den første versjonen av Kinect beregnet for utvikling i Windows kom på markedet i februar 2012 [Eisler, 2012]. Kinect v2 fulgte relativt raskt etter, og ble utgitt i juli 2014 [Pradeep, 2014].

3.5.1 Forbedringer fra v1 til v2

Kinect v2 har en del oppgraderinger i forhold til Kinect v1. En del av oppgraderingene finnes oppsummert i Tabell 3.

Tabell 3: Forskjeller mellom Kinect v1 og v2. Data om oppløsning er hentet fra [Smeenk, 2014], mens resten av dataene er hentet fra [Microsoft, 2015e].

	v1	v2
Oppløsning(video)	640x480	1920x1080
Oppløsning(dybde)	320x240	512x424
Sporing	6	6
Skjelettsporing	2	6
Ledd	20	25
Tilkobling	USB2.0	USB3.0
Avstand	1.2 m - 3.0 m	0.5 m - 4.5 m

En annen ganske stor forandring, og fordel, i forbindelse med v2, er at den gir flere applikasjoner mulighet til å lese informasjon fra Kinecten på

likt. I v1 måtte distribusjonslogikken settes opp av utvikleren, mens det nå i v2 går via SDKen [Microsoft, 2015e].

3.5.2 Hvordan fungerer den?

Det er det Apple Inc-eide selskapet PrimeSense som står bak maskinvaren i Kinect [Miller, 2011].

Som sagt består Kinecten av et HD videokamera, en infrarødsender og en mikrofonrekke. Disse kan brukes sammen for å hente ut mye nyttig informasjon fra et bilde.

Avstandsmåling

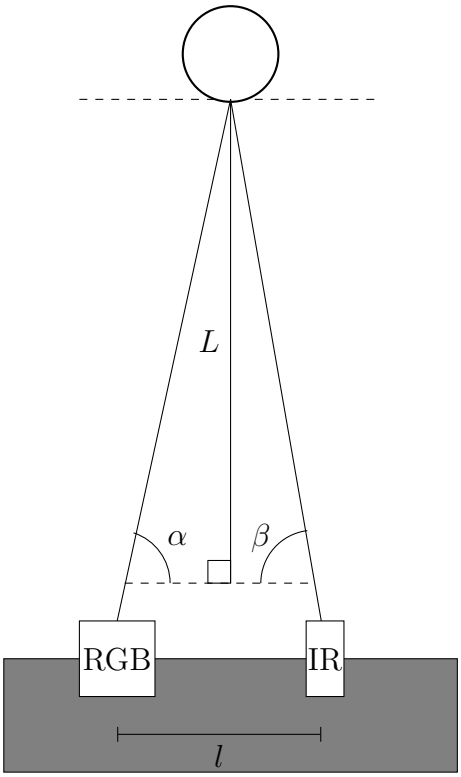
Avstandsmålingen som gjøres av en Kinect er basert på PrimeSenses lyskodingsteknologi [The Chaos Computer Club, 2011]. Denne teknologien baserer seg på at infrarødsenderen projekterer et rutenett av grupper med små hvite dotter i bildet som videokameraet ser. Disse utgjør da et slags koordinatsystem, og objekters posisjon i bildet kan angis relativt til dette. Dette gjør det mulig å ved hjelp av videokameraet og infrarødsenderen kunne bestemme objekters avstand fra Kinecten, se Figur 8.

Skjelettsporing

I følge [MacCormick,] detekterer Kinecten kropper og kroppspositurer ved hjelp av skjelettvisning ved å kombinere dybdebildet med maskinlærte algoritmer [2]. Omgjøringsprosessen fra et dybdebilde til skjelettvisning er, i følge samme kilde, en todelt prosess:

1. Oversette dybdebildet til kroppsdelsbilder
2. Oversette fra kroppsdelsbilder til ledd-/skjelettvisning

[2] Algoritmer datamaskinen har lært seg gjennom en rekke tester.



Figur 8: Infrarødsenderen lager et rutenett i bildet videokameraet ser, slik at de har et felles koordinatsystem å organisere objekter i. Ut i fra den konstante avstanden mellom infrarødsenderen og RGB-kameraet (l), samt vinklene mellom dem og et objekt (α og β), kan så avstanden til objektet (L) beregnes.

Kinecten har gjennom et treningssett på 100000 dybdebilder med kjent skjelett, lært seg et sett med valgtrær for deteksjon av kropper i dybdebilder. Disse settene har den så satt sammen til en tilfeldig valgskog [3]. Denne valgskogen benyttes når et dybdebilde skal oversettes til et kroppsdelsbilde. For hvert dybdebilde blir det valgt et tilfeldig valgtre, som så følges for å beslutte hva i bildet som er menneskekropper.

For så å til slutt kunne oversette kroppsdelsbildet til selve skjelettvisningen, benyttes *mean-shift*-algoritmen for å beregne kroppsdeler/*clustere*.

3.5.3 Utviklingsmuligheter med Microsoft Kinect

Microsoft har i tillegg til selve Kinecten gitt ut adapter som gjør det mulig å koble Kinecten til en PC, og SDK som gjør det mulig å utvikle Windows-applikasjoner.

For å programmere med Kinect, stilles det en del krav til maskinvaren den skal benyttes med. Kravene finnes oppsummert i Tabell 4.

Tabell 4: Maskinvarekrav i forbindelse med Kinect v2 [Microsoft, 2015f].

Maskinvare	Minstekrav
Prosessor	64bit
Klokkefrekvens	3.1 GHz
Tilkobling	USB3.0
Tilkoblingsdriver	Intel
RAM	4GB
Grafikkort	støtter DirectX
Operativsystem	Windows 8 / 8.1

For utvikling med Kinect, kan kode skrives i enten C++, C#, Visual Basic, eller et annet .NET-språk [Microsoft, 2015g].

Mulighetene for utvikling med Kinect er mange, og den er blitt svært populær blant teknologi- og programmeringsentusiaster. Dette kan sees ut i fra at det er opprettet opptil flere *communities* for Kinectentusiaster. For eksempel `http://www.kinecthacks.com/` og `http://developkinect.com/`.

[3] Samling av valgtrær, der et tilfeldig valgtre velges hver gang. Mer info: [MacCormick,]

Det vil her presenteres tre eksempler på applikasjoner som er blitt utviklet for Kinect.

1. Kontroller for Portal 2

Portal 2 er et spill til Xbox som kom ut i 2011 [Narcisse, 2011], og er etterfølger til spillet Portal som kom i 2007 [Narcisse, 2011]. Målet med spillet er å komme seg helskinnet gjennom en rekke løyper med innlagte hindere/puslespill som må løses ved hjelp av ulike våpen eller hjelpemidler som dukker opp på veien.

Intel Perceptual Computing [Hollister, 2013] er en Kinectapplikasjon som gjør det mulig å styre spillet vha håndbevegelser. Kinecten kan registrere åpning og lukking av hender, samt bevegelse i både x-, y- og z-retning. Dette er benyttet i spillstyringen, slik at lukking/åpning av hånden resulterer i at objekter blir henholdsvis plukket opp/sluppet. Tilting av hånden resulterer i at objektet roteres, mens det å bevege hånden fra/mot Kinecten, resulterer i at objektet beveger seg henholdsvis lengre ut av skjermen/inn i skjermen.

2. Selvspillende musikkinstrument

Daniel Iglesia har utviklet programvaren Rengeinet, som produserer musikk ved hjelp av Kinect. Kinecten leser fra høyre til venstre gjennom bildet den ser gjentatte ganger, og produserer for hvert utsnitt en tone. Hvilken tone som spilles av, er avhengig av størrelse, posisjon og orientering på objektet i utsnittet. Gjennom programmets forløp produseres det flere toner, og disse utgjør til sammen et Kinectkomponert musikkstykke [Iglesia, 2013].

3. Virtuelt prøverom

Adam Horvath har brukt Kinect for å lage et virtuelt prøverom [Hacks, 2011]. Brukeren står fremfor Kinecten og velger ut antrekk fra en meny, og så tegner programmet det valgte antrekket på brukeren.

3.5.4 Programmering med Kinect i C#

Under følger en introduksjon til programmering med Kinect ved bruk av programmeringsspråket C#. Inspirasjon er hentet fra [Pterneas, 2014].

For å kunne benytte Kinectens funksjonalitet, trengs det først og fremst et objekt som representerer Kinecten. I C# heter denne klassetypen for *KinectSensor*. Før den kan benyttes, må *KinectSensoren* åpnes ved å kalle på medlemsfunksjonen *Open()*. For å lukke den igjen, benyttes medlemsfunksjonen *Close()*. Dette er illustrert i Figur 9.

```
KinectSensor sensor = KinectSensor.GetDefault();

sensor.Open();

//do something with your Kinect

sensor.Close();
```

Figur 9: En Kinectsensor representeres i C# ved et objekt av klassetypen KinectSensor. Denne åpnes ved å kalle på medlemsfunksjonen Open(), og lukkes ved å kalle på medlemsfunksjonen Close().

For å hente ut informasjon fra Kinecten, benyttes såkalte *FrameReader*e. Fem av disse er:

1. ColorFrameReader – gir tilgang til fargebilder
2. DepthFrameReader – gir tilgang til dybdebilder
3. InfraredFrameReader – gir tilgang til infrarødbilder
4. BodyFrameReader – gir tilgang til informasjon om eventuelle skjelett som befinner seg i bildet
5. MultiSourceFrameReader – gir tilgang til alle de ovenfornevnte bildene og skjelett

Hver av *FrameReaderne* har sitt eget event som settes opp hver gang en ny frame av tilsvarende type er tilgjengelig. For å oppnå kontinuerlig oppdatering av bildeinformasjonen, bør det dermed settes opp event handlere for

disse eventene. Dette gjøres for alle via *FrameReader*ens *FrameArrived*-event. Et eksempel for en *ColorFrameReader* er vist i Figur 10 [Microsoft, 2015a].

```
ColorFrameReader colorReader = sensor.ColorFrameSource.OpenReader();

colorReader.OnFrameArrived += new EventHandler(Reader_ColorFrameArrived);

void Reader_ColorFrameArrived(object o, ColorFrameArrivedEventArgs args)
{
      //Handle event
}
```

Figur 10: ColorFrameReaderen henter ut sin fargebildekilde via KinectSensoren (her sensor). Når eventet OnFrameArrived inntreffer, blir funksjonen Reader_ColorFrameArrived() kjørt.

Bildet som kommer fra Kinectens *FrameReadere* blir sendt til eventet via eventargumentet, og hentes ut derfra via funksjonen *AcquireFrame()*. Bildet som returneres av *AcquireFrame()* er i et format tilsvarende bildet *FrameReader*en tilhører, for eksempel *ColorFrame* for *ColorFrameReader*.

For å kunne vise bildet i en applikasjon, må denne *Frame*n oversettes til, for eksempel, en *BitmapSource*, som kan vises i et XAML-Image. Oversettelsen gjøres via et byte array, som illustrert i Figur 11.

```
void Reader_ColorFrameArrived(object o, ColorFrameArrivedEventArgs args)
{
  /*get color frame from event args*/

  /*create a byte-array to loading the image into,
    length of array depends on size (width and height)
    of the image*/

  /*copy data from frame to byte-array*/

  /*create bitmapsource from byte-array*/

  /*load image with the bitmapsource*/
}
```

Figur 11: Oversettelsen fra en ColorFrame til en BitmapSource foregår via en byte-tabell.

Når en *BitmapSource* er fylt med data fra en *Frame*, kan den legges direkte inn i et XAML-Image sin Source-egenskap, og dermed vises i applikasjonen.

For visning av infrarød- og dybdebilder gjelder en ganske liknende fremgangsmåte.

For fremvisning av skjelett, er prosessen derimot noe annerledes. Selve *BodyFrame*n hentes ut på samme vis som en *ColorFrame*, altså via *AcquireFrame()*. Forandringen kommer under tegning av skjelett.

Dataene som kan hentes ut fra *BodyFrameReader*ens *BodyFrameArrivedEventArgs* inneholder informasjon blant annet informasjon om posisjonen til 25 kroppsledd for opptil seks personer. Disse tegnes individuelt inn i et objekt av *Canvas*-typen, som er koblet sammen med et XAML-Image.

Dataene fra *BodyFrameArrivedEventArgs* legges først inn i en tabell av *Body*-objekter. *Body*-klassen inneholder medlemsvariabler og funksjoner for å få tak i, samt å behandle, informasjon om skjelettene som befinner seg i bildet. For å kopiere over informasjon fra eventargumentet til *Body*-tabellen, benyttes *BodyFrame*-klassens medlemsfunksjon *GetAndRefreshBodyData()*, som vist i Figur 12.

```
void Reader_BodyFrameArrived(object o, BodyFrameArrivedEventArgs args)
{
  /*get body frame from event args*/

  /*update body data in class body-array*/
}
```

Figur 12: Event handleren for ankomst av ny BodyFrame. Vektoren bodies oppdateres med fersk data via BodyFrameArrivedEventArgs.

Figur 13 viser hvordan *Body*-objektet kan brukes til å bla gjennom alle de tilgjengelige leddene for å kunne kalle en funksjon for å tegne dem.

```
foreach(JointType joint in body.Joints.Keys)
{
  //Do something with joint
}
```

Figur 13: DrawBody-funksjonen blar seg gjennom alle leddene som er tilgjengelig i Body-elementet, og kaller på en funksjon for å tegne dem [Pterneas, 2014].

Figur 14 viser hvordan hvert ledd kan tegnes i et *Canvas*.

```
void DrawJoint(Joint joint)
{
  /*Create ellipse with appropriate size and shape*/

  /*Add the ellipse to a Cancas at the joint-position*/
}
```

Figur 14: DrawJoint tar inn argumenter som angir hvilket ledd som skal tegnes, hvor stort det skal være, om det skal tegnes fylt eller ikke, tykkelse på omrisset og farge. Dette brukes for å konstruere en ellipse, som tegnes inn i et felles Canvas for alle leddene [Pterneas, 2014].

3.5.5 Applikasjonseksempel: skjelettsporing

Den resulterende applikasjonen skal vise Kinectens videostrøm, samt at personer som befinner seg i bildet skal markeres med fullt skjelett. Hele eksempelet finnes i elektronisk vedlegg som "BodyTracking.cs".

Denne oppgaven er todelt. Første del går på å vise fargebilder fra Kinectens videokamera, mens andre del av implementeringen er skjelettsporingen.

For å vise bilde og skjelettene, trengs det et XAML-dokument med et Image-objekt for fargebilder, og et tilhørende *Canvas* for skjelett. Image-objektet og *Canvas*-objektet plasseres sammen i et Grid, som vist i Figur 15.

```
<Grid>
    <Image Name="camera" MouseLeftButtonDown="Mouse_ClickEvent"/>
    <Canvas Name="headsCanvas"/>
</Grid>
```

Figur 15: For å få Canvas-objektet til å overlappe Image-objektet, plasseres de sammen i et Grid.

For å vise fargebildet i Image-objektet, behøves det først og fremst en *ColorFrameReader*. Det må så settes opp en event handler til *ColorFrameReader*ens event *FrameArrived*, slik at bildet kan oppdateres ved hver nye ankomne fargebilde. Event handleren må ta seg av oversettelse fra *ColorFrame* til *BitmapSource*, slik at bildet til slutt kan legges inn i Image-objektet.

For å tegne skjelett, må det først hentes ut informasjon om plassering av personer og deres ledd i Kinectens bilde. Dette gjøres via en *BodyFrameReader*. *BodyFrameReader*ens event *FrameArrived* må kobles opp til en event handler, og denne må for alle personene som befinner seg i bildet gå gjennom

```
namespace BodyTracking
{
  public class PersonDisplayer
  {
    private KinectSensor sensor;

    private ColorFrameReader colorReader;
    private BodyFrameReader bodyReader;

    private Body[] bodies;

    public PersonDisplayer()
    {
      InitializeComponent();

      sensor = KinectSensor.GetDefault();
      sensor.Open();

      colorReader = sensor.ColorFrameSource.OpenReader();
      colorReader.FrameArrived += Reader_ColorFrameArrived;

      bodyReader = sensor.BodyFrameSource.OpenReader();
      bodyReader.FrameArrived += Reader_BodyFrameArrived;

      bodies = new Body[6];
    }
```

alle ledd som er registrert for dem, og tegne dem inn i det felles *Canvas*-objektet.

Implementasjonen kan sees i Figur 14.

```
void Reader_ColorFrameArrived(object sender,
                      ColorFrameArrivedEventArgs args)
{
  var frame = args.FrameReference.AcquireFrame();

  if (frame != null)
  {
    camera.Source = ToBitmap(frame);
  }
}
private ImageSource ToBitmap(ColorFrame frame)
{
  int width = frame.FrameDescription.Width;
  int height = frame.FrameDescription.Height;

  byte[] pixels = new byte[/*size of picture*/];

  if (frame.RawColorImageFormat == ColorImageFormat.Bgra)
  {
    frame.CopyRawFrameDataToArray(pixels);
  }
  else
  {
    frame.CopyConvertedFrameDataToArray(pixels,
                                        ColorImageFormat.Bgra);
  }

  /*create bitmapsource and return it*/
}
```

```
    private void DrawBody(Body body)
    {
      foreach (JointType joint in body.Joints.Keys)
      {
        DrawJoint(body.Joints[joint]);
      }
    }

    private void DrawJoint(Joint joint)
    {
      /*get joint position*/

      Ellipse ellipse = new Ellipse();
      ellipse.Fill = Brushes.RoyalBlue;
      ellipse.Stroke = Brushes.White;
      ellipse.StrokeThickness = borderWidth;
      ellipse.Width = 15;
      ellipse.Height = 2;

      /*position ellipse at joint position*/

      headsCanvas.Children.Add(ellipse);
    }
  }
}
```

Figur 14: Implementasjon av applikasjon som viser fargebilde med påtegnede skjelett for personer i bildet.

4 Arbeidsmetodikk

4.1 Valg av utviklingsmetodikk

Da dette er et forholdsvis kortvarig prosjekt med veldefinerte krav, er det valgt å ta utgangspunkt i en kombinasjon av fossefallmetoden og iterative metoder under systemutviklingen. Dette for å sette klare rammer for utviklingsprosessen, samt å sikre fremgang.

Utviklingsprosessen vil gå gjennom fasene som ble introdusert i Delkapittel 3.1.1. Fase 1, 4 og 5 er alle i hovedsak tenkt å skulle gjennomføres kun én gang. Den iterative delen av utviklingsmetodikken slår inn mellom fase 2 og 3. Disse fasene vil bli gjennomgått i flere iterasjoner for å optimalisere systemdesignet.

Selv om fase 1 er tenkt å kun gjennomføres en gang, vil det, som i iterative metoder, være tillatt å modifisere kravspesifikasjonen ettersom eventuelle feil eller forbedringer oppdages. Kompilering og testing er også velkomment underveis i implementeringen.

Utviklingsmetodikken som benyttes i dette prosjektet kan ses på som enten en adaptiv [4] versjon av fossefallsmodellen, eller en inkrementell metode med redusert iterasjonsomfang.

Selv om selve utviklingsmetodikken er en blanding av fossefallmetoden og iterative metoder, vil presentasjonen av systemutviklingen være strukturert etter fossefallmetoden. Hver fase for seg, og i den nummererte rekkefølgen gitt i Delkapittel 3.1.1.

4.2 Tidsplanlegging

Utviklingen av systemet, samt dokumentering og rapportskriving, skal alt foregå i løpet av de 20 ukene som er tildelt arbeidet med masteroppgaven. Derfor er det viktig å strukturere tiden riktig og effektivt.

Prosessen som skal gjennomgås i løpet av dette tidsrommet er delt opp i følgende deloppgaver med beskrivelse:

[4] Aksepterer at endringer er normal [Hansen and Hjertø, 2006]

Prosjektplanlegging: overordnet planleggingsprosess i forbindelse med selve masteroppgaven. Inndeling i deloppgaver, og utforming av tidsplan (Gantt-diagram). Planlegge oppsett av rapport, og bestemme forstudie.

Forstudie: Finne informasjon om, samt skrive, teoridelen av rapporten.

Kravspesifikasjon: Formulering av kravspesifikasjon for systemet som skal utvikles.

Testformulering: Planlegging av tester som systemet skal oppfylle, basert på kravspesifikasjon.

Systemdesign: Designe et system som vil kunne oppfylle kravspesifikasjonen og testene.

Implementasjon: Implementering av systemet som er designet.

Verifisering: Teste at systemet fungerer som ønsket, ved å kjøre testene.

Dokumentering: Dokumentere systemet.

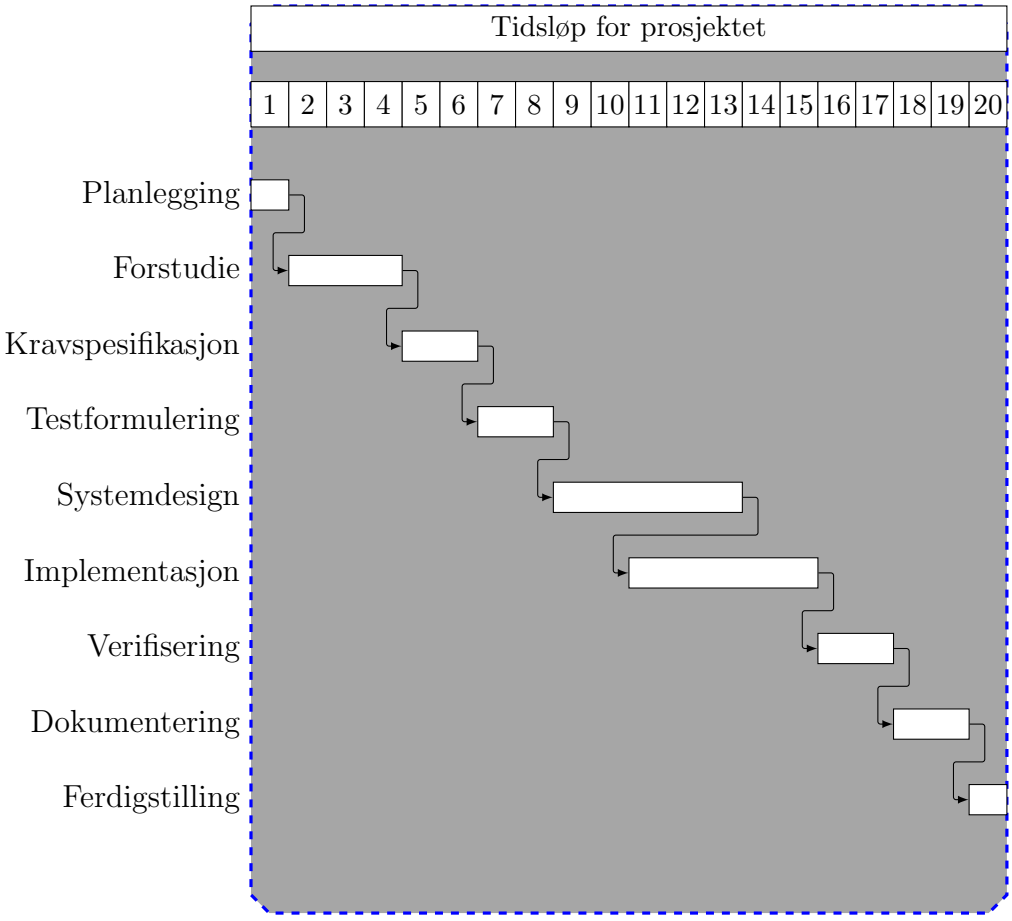
Ferdigstilling: Fullføring av rapport.

Planlagt starttidspunkt, samt varighet for hver av deloppgavene, er vist i Figur 15. Da systemdesign og implementasjon vil foregå i flere iterasjoner, er disse fasene overlappende.

4.3 Valg av programmeringsspråk

Av programmeringsspråkene som er tilgjengelig for utvikling med Kinect, stod valget i hovedsak mellom C++ og C#.

C++ virker fordelaktig, med tanke på allerede tilegnede erfaringer med språket. Ulempen med språket, og det som dermed taler i mot å skrive i C++, er minnehåndtering. I C++ er programmereren selv herre over minneallokering og frigjøring, hvilket kan ses på som både en fordel og en ulempe. I relativt store prosjekter kan det oppstå feilmeldinger det kan være vanskelig å oppdage roten til.



Figur 15: Gantt-diagram for systemutviklingen.

Grunnet dette, og lysten til å lære noe nytt, var det aktuelt å utforske andre muligheter.

C# har syntaksmessige likheter med Java, som forfatteren også har brukt tidligere, og C++. Minnehåndtering i C# fungerer på samme vis som i Java, og blir i hovedsak ordnet automagisk. I tillegg har det kommet forfatteren for øret at C# har diverse fordeler i forbindelse med nett- og socketprogrammering, da det tilhører .NET-rammeverket. Dette kommer godt med, da denne systemutviklingen innebærer nettprogrammering.

C# er altså både ukjent og kjent på samme tid, og byr dermed på mye nyttig funksjonalitet kombinert med nogenlunde kjent syntaks. Det er dermed besluttet å benytte C# i denne oppgaven.

5 Utforming av kravspesifikasjon

Kravspesifikasjonen som presenteres i dette kapitlet er formulert av utvikleren av systemet, Stine Lilleborge. Da hun har god kjennskap til systemet som skal planlegges, etter å tidligere ha formulert funksjonsspesifikasjon for roboten systemet skal være en del av [Lilleborge, 2014], regnes dette som hensiktsmessig.

For dokumentering av kravspesifikasjon er det tatt utgangspunkt i SINTEFs standard for kravspesifikasjon [Bræk, 1982]. Del 5, 9 og 10 er utelatt, da de ikke anses som relevante for dette systemet. I tillegg følger en punktliste over endelige krav i Delkapittel 5.8.

5.1 Innledning

Systemet som skal designes er en del av NTNUs bioteknologiske satsningsprosjekt NTNU-Cyborg, som ble startet høsten 2014. Systemet som beskrives i denne kravspesifikasjonen skal utgjøre kyborgens øyne, og danne et godt utgangspunkt for å gjøre kyborgen i stand til å oppsøke sosiale situasjoner.

Kravspesifikasjonen er delt inn i 7 deler, inkludert innledningen, som regnes som del 1. Del 2 inneholder en overordnet beskrivelse av systemet. Her gis det en innføring i hva systemet skal gjøre, samt at en inndeling i delsystemer presenteres grafisk.

Del 3 forteller om systemets sammenheng med resten av NTNU-Cyborgprosjektet, og en beskrivelse av den kommende kyborgens arbeidsområde.

Del 4 dykker dypere ned, og gir funksjonsbeskrivelse for hvert delsystem/modul i systemet.

Del 5 spesifiserer begrensninger som oppstår som funksjon av allerede vedtatte maskinvarebeslutninger.

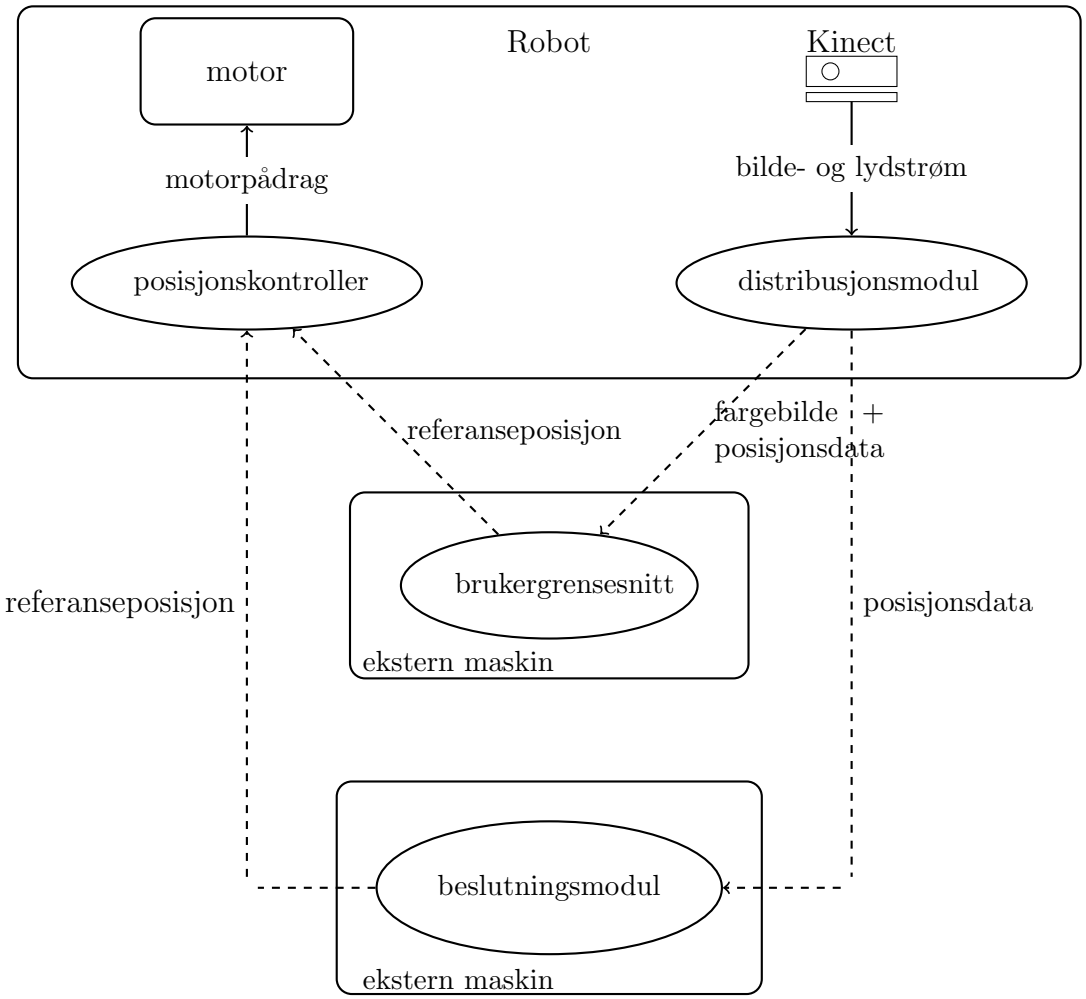
Del 6 gir en beskrivelse av hva systemdokumentasjonen skal inneholde, mens del 7 beskriver brukergrensesnittmodulens virkemåte.

5.2 Overordnet systembeskrivelse

Hensikten med systemet er å velge ut en person innen Kinectens synsvidde, og følge denne, samt beregne dens posisjon.

Systemet bør deles inn i flere moduler, som skal fungere uavhengig av hverandre og kommunisere via trådløst nettverk. Systemet skal kunne fungere automatisk, uten ytre påvirkninger, og dermed på egenhånd oppdage

personer omkring den, velge ut en av dem, og produsere utdata som forteller hvor i rommet personen befinner seg. Det skal i tillegg være mulig å overstyre dette systemet, ved å manuelt velge ut personen som skal følges. Figur 16 viser en oversiktsfigur for systemet.



Figur 16: Når systemet fungerer som normalt, går informasjonsflyten fra Kinect til motor via distribusjonsmodul, beslutningsmodul og posisjonskontroller. Ved manuell styring av systemet, går informasjon fra distribusjonsmodulen til en modul med et tilhørende brukergrensesnitt ut mot bruker, som muliggjør manuelt valg av person roboten skal følge. Informasjonen som når posisjonskontrolleren kommer da fra brukergrensesnittmodulen.

Alle kommunikasjonskanaler skal kobles opp ved oppstart av systemet, og brytes når systemet slås av. Dersom det oppstår kommunikasjonssvikt/nettverksbrudd, skal systemet skru seg av etter 6 sekunders forsøk på å gjenopprette kommunikasjonskanalene.

5.3 Rammekrav

Systemet er del av et større system, som skal utvides trinnvis modul for modul. Denne første modulen gir roboten evnen til å se, mens det etterhvert også skal legges til moduler for å tilføye funksjonalitet for hørsel, tale og andre livnære funksjoner [Lilleborge, 2014].

Blant utvidelsene finnes en posisjonskontroller. Denne vil motta posisjonsinformasjon fra dette systemet, og passe på at kyborgen beveger seg mot referanseposisjonen.

Roboten skal bevege seg i et rom kalt Glassgården i Elektrobygget på campus Gløshaugen, tilhørende NTNU. Dette er et lysrikt rom med mye passerende mennesker. Roboten kommer til å ha et relativt stort og potensielt støyfult område å bevege seg rundt på, og vil komme i nær kontakt med både studenter og ansatte ved NTNU.

Distribusjonsmodulen vil etter videre utvidelse av systemet kunne motta forespørsler om å sende andre bildedata enn de som brukes i dette delsystemet. Den må derfor være i stand til å håndtere forespørsler etter alle bildeformatene Kinecten tilbyr.

5.4 Systemets funksjonelle egenskaper

Distribusjonsmodul: leser data fra Kinecten, og videreformidler etterspurt informasjon til henvendende moduler. Inndata vil da være bildestrøm av både farge-, dybde- og infrarødbilder. Utdata vil være bildeinformasjon i farge-, dybde eller infrarødformat eller posisjonsdata for personer i bildet. Distribusjonsmodulen vil sende posisjonsdata til Beslutningsmodulen, mens både farge-bilde og posisjonsdata sendes til Brukergrensesnittmodulen.

Beslutningsmodul: skal bestemme hvem av personene som er blitt funnet i bildet kyborgen skal bevege seg mot. Inndata er ansiktsposisjoner. Basert på dette, skal det kunne konstrueres ulike heuristiske funksjoner som bestemmer hvilken person som er den mest "optimale" å bevege

seg mot. Utdata fra modulen er en referanseposisjon det er ønskelig at roboten beveger seg mot.

For alle modulene gjelder det at informasjon sendes og mottas med 1-2 sekunders mellomrom, og at de kan anses som inaktive etter å ha vært utilgjengelige i 6 sekunder.

5.5 Krav til systemkonstruksjonen

Utstyr som på forhånd er bestemt at skal benyttes er

→ Microsoft Kinect v2

→ Pioneer LX robotbase fra Adept Mobilerobots

Kinecten krever Windows 8 og har tilkoblingsmulighet via USB3.0. Programvare tilknyttet Kinecten direkte må programmeres i enten C++, C#, Visual Basic, eller et annet .NET-språk [Microsoft, 2015g].

Robotbasen har en integrert datamaskin som kjører Windows 7 og har 2 USB2.0-porter [Mobilerobots, 2015]. Det må dermed innføres et ledd mellom Kinecten og robotbasen, dersom distribusjonsmodulen skal kjøres direkte på datamaskina i robotbasen.

Kommunikasjonssystemet skal benytte seg av TCP-koblinger, for å simplifisere langvarig kommunikasjon mellom modulene.

5.6 Krav til dokumentasjon

Det skal produseres en brukermanual for systemet. Denne skal inneholde en beskrivelse av hva slags informasjon som kan mottas fra systemet, samt en innføring i hvordan en applikasjon kan koble seg opp mot systemet for å kunne motta denne informasjonen.

5.7 Krav til manuelle funksjoner

Kun en av modulene i systemet gir mulighet for manuell påvirkning.

Denne modulen har et fargebilde som inndata. Bildet skal vises i applikasjonen tilhørende modulen, og ansiktene i bildet skal markeres med grønne firkanter, se Figur 17a. Det skal så være mulig for brukeren å klikke hvor som helst inni en av disse firkantene, for å aktivere følging av den personen

(a) Grafisk brukergrensesnitt før valg av person.

(b) Grafisk brukergrensesnitt etter valg av person.

Figur 17: Brukergrensesnitt for manuelt valg av person roboten skal følge.

firkanten tilhører. Når en person er blitt valgt ved hjelp av denne prosedyren, skal boksen tilhørende personen skifte farge til orange, se Figur 17b.

For å velge en annen person å følge, skal det bare være å velge en ny person på samme måte som den første ble valgt. For å annulere valg av person, skal det kunne trykkes på et hvilket som helst punkt inne i den orange ruta rundt personens ansikt. Da skal boksen rundt personens ansikt bytte farge tilbake til grønn.

Så lenge en person er valgt på manuelt vis ved hjelp av denne modulen, kan ikke roboten selv velge person å følge. Det manuelle valget er gyldig inntil det annuleres, eller til applikasjonen avsluttes. Først når en av disse hendelsene inntreffer skal roboten igjen kunne gjøre egne valg.

Utdata fra denne modulen, er ønsket posisjon for roboten, altså posisjonen til den personen roboten skal bevege seg mot.

5.8 Liste over krav

5.8.1 Distribusjonsmodul

1. inn- og utdata for modulen er på byte[]-format
2. forespørsler om tilkobling fra nye klienter skal mottas kontinuerlig
3. kun klienter med forespørsel lik ”color”, ”body”, ”depth” eller ”infra-red” skal godkjennes

4. hver nye tilkobling skal håndteres av en egen prosess
5. prosessen avsluttes når forbindelsen går tapt
6. når en ny klient får koble seg til, skal den legges til i klientlisten
7. når en klient kobles fra/mister forbindelsen med modulen, skal den fjernes fra klientlisten
8. når modulen mottar en forespørsel med melding "body", skal den med ett sekunds mellomrom sende ut oppdatert posisjonsinformasjon til klienten forespørselen kom fra
9. når modulen mottar en forespørsel med melding "color", skal den med ett sekunds mellomrom sende ut et nytt fargebilde til klienten forespørselen kom fra
10. når modulen mottar en forespørsel med melding "depth", skal den med ett sekunds mellomrom sende ut et nytt dybdebilde til klienten forespørselen kom fra
11. når modulen mottar en forespørsel med melding "infrared", skal den med ett sekunds mellomrom sende ut et nytt infrarødbilde til klienten forespørselen kom fra
12. lengden på bildemeldingene som går ut fra modulen er på $(bildebredde) \times (bildehøyde) \times (bitperpiksel) + 1$ byte
13. lengden på posisjonsmeldingene er på $(antallmuligepersoneribildet) \times (antalldatafelt) + 1$ byte

5.8.2 Beslutningsmodul

1. inn- og utdata for modulen er på byte[]-format
2. lengden på inndata er på $(antallmuligepersoneribildet) \times (antalldatafelt) +$ 1 byte, og inneholder posisjonsdata
3. lengden på utdata er $(antallkoordinaterforposisjon)$ byte, og inneholder informasjon om personen som følges
4. utdata sendes med ett sekunds mellomrom

5. inndata skal mottas i egen prosess
6. utdata skal sendes i egen prosess
7. posisjonsdata skal lagres i modulen
8. når nye meldinger mottas, skal posisjonsdata i modulen oppdateres
9. tilgjengelige personer representeres med tallverdier større enn null
10. manglende personer representeres med 0'ere
11. kun én person skal følges av gangen
12. så lenge det er personer i bildet, skal det være valgt en person å følge
13. dersom det ikke er registrert personer i bildet , blir utdata fylt med 0'ere
14. dersom tilkoblingen til en av de andre modulene brytes, skal det bli forsøkt å opprette ny tilkobling i 6 sekunder før kontakten anses for å være tapt
15. når kontakten med en av modulene anses å være tapt, skal kontakten med den andre modulen også brytes, og applikasjonen avsluttes

5.8.3 Brukergrensesnittmodul

1. inn- og utdata for modulen er på byte[]-format
2. modulen skal kunne ta imot to ulike inndata parallelt
3. den ene inndataen skal være posisjonsdata
4. den andre inndataen skal være fargebilder
5. inputene skal håndteres individuelt
6. når nye data ankommer modulen, skal de lagres i modulen
7. når dataene i modulen oppdateres, skal de vises for brukeren
8. posisjonsdata skal resultere i firkantmarkeringer rundt personer hode

9. firkantene skal kunne ha to ulike farger
10. grønn firkant skal representere en person som ikke følges
11. orange firkant skal representere en person som følges
12. kun én person skal følges av gangen
13. trykk i grønn firkant skal aktivere følging av personen firkanten tilhører
14. trykk i orange rute skal deaktivere følging av personen firkanten tilhører
15. ved aktivering av ny person å følge, skal tidligere aktivert følging av en person deaktiveres før ny følging aktiveres
16. dersom person å følge er satt, skal utdata settes til x-, y- og z-posisjon for denne personen
17. dersom person å følge ikke er satt, skal utdata fylles med 0'ere
18. dersom tilkoblingen til en av de andre modulene brytes, skal det bli forsøkt å opprette ny tilkobling i 6 sekunder før kontakten anses for å være tapt
19. når kontakten med en av modulene anses å være tapt, skal kontakten med den andre modulen også brytes, og applikasjonen avsluttes

6 Systemdesign

Dette kapitlet presenterer valgt systemdesign med klassediagram og aktivitetsdiagram for alle delsystemene. Formatet som er brukt på diagrammene er hentet fra [Fowler, 2004].

6.1 Distribusjonsmodul

Distribusjonsmodulens funksjonalitet er tenkt fordelt på to klasser:

- FrameSupplier
- KinectFrameServer

Klassen *FrameSupplier* skal lese informasjon fra Kinecten og behandle denne, mens *KinectFrameServer*-klassen skal oppnå kontakt med henvendende moduler og videreformidle ønsket informasjon til dem.

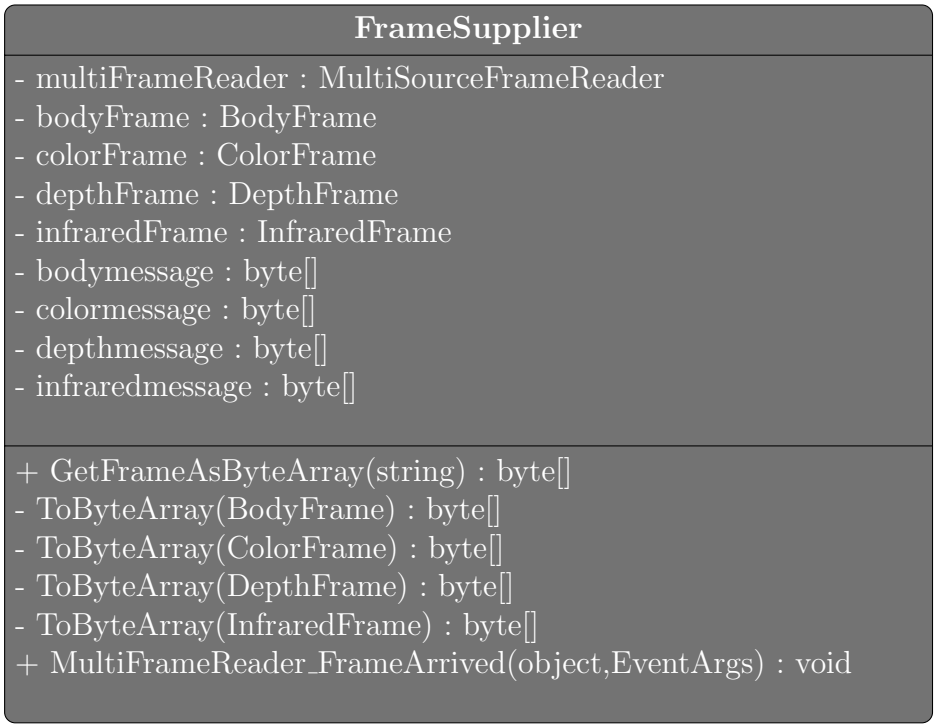
FrameSupplier

Planlagt klassediagram for *FrameSupplier*-klassen er vist i Figur 18.

Klassen skal inneholde medlemsvariabler for å lagre posisjons-, fargebilde-, dybdebilde- og infrarødbildeinformasjon, da henholdsvis i variablene *bodyFrame*, *colorFrame*, *depthFrame* og *infraredFrame*. Disse skal oppdateres hver gang det er nye data tilgjengelig, altså hver gang det inntreffer et nytt *FrameArrived*-event (enten *BodyFrameArrived*, *ColorFrameArrived*, *DepthFrameArrived*, *InfraredFrameArrived*).

For å fange opp disse eventene, er det planlagt å benytte en *MultiSourceFrameReader*. Denne kan fange opp eventer av alle de nevnte typene, og er dermed fordelaktig i forhold til å ha en *FrameReader* for hver av dem.

FrameSupplier-klassen skal også inneholde funksjonalitet for å oversette hver av de nevnte variablene til byte-tabeller, da dette er formatet informasjon blir sendt til andre moduler på. Disse byte-tabellene skal ligge lagret i klassen, slik at de er klare når de skal sendes.



Figur 18: Klassediagram FrameSupplier-klassen i Distribusjonsmodulen.

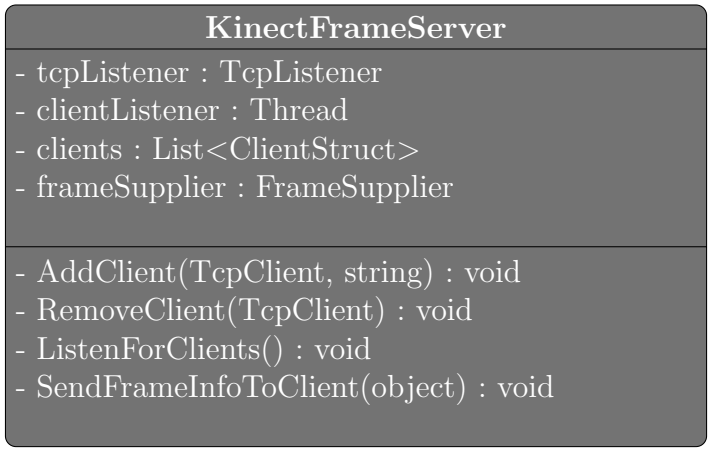
KinectFrameServer

Klassediagram for *KinectFrameServer*-klassen er vist i Figur 19.

Denne klassen skal inneholde et objekt av typen *FrameSupplier*, som den henter informasjon i byte[]-format ut fra. Disse byte-tabellene brukes direkte når informasjon skal videreformidles til Distribusjonsmodulens klienter.

For å kunne lytte etter klienter, er det tenkt å benytte en *TcpListener*. Denne skal hele tiden lytte etter nye henvendelser, via funksjonen *ListenForClients()*. Denne funksjonen skal, for hver nye henvendelse, kontrollere om tilsendt beskjed overensstemmer med kriteriet for godkjenning, gitt av kravlista for Distribusjonsmodulen i Delkapittel 5.8.1. Aktivitetsdiagram for *ListenForClients()* er vist i Figur 20.

Dersom klientens henvendelse godkjennes, legges klienten til i klassens liste over klienter vha *AddClient(TcpClient)*. I tillegg skal det opprettes en ny prosess for kommunikasjon med klienten. Funksjonen som tar seg av kom-



Figur 19: Klassediagram for KinectFrameServer i Distribusjonsmodulen.

munikasjon med klientene, er *SendFrameInfoToClient()*. Denne tar inn et objekt av TcpClient-typen, og sender med ett sekunds mellomrom oppdatert versjon av den forespurte informasjonen til klienten. Aktivitetsdiagram for *SendFrameInfoToClient()* er vist i Figur 21.

Dersom kontakten med en klient forsvinner, vil klienten fjernes fra klient-listen vha *RemoveClient(TcpClient)-funksjonen*, og prosessen som tok seg av kommunikasjon med klienten avsluttes.

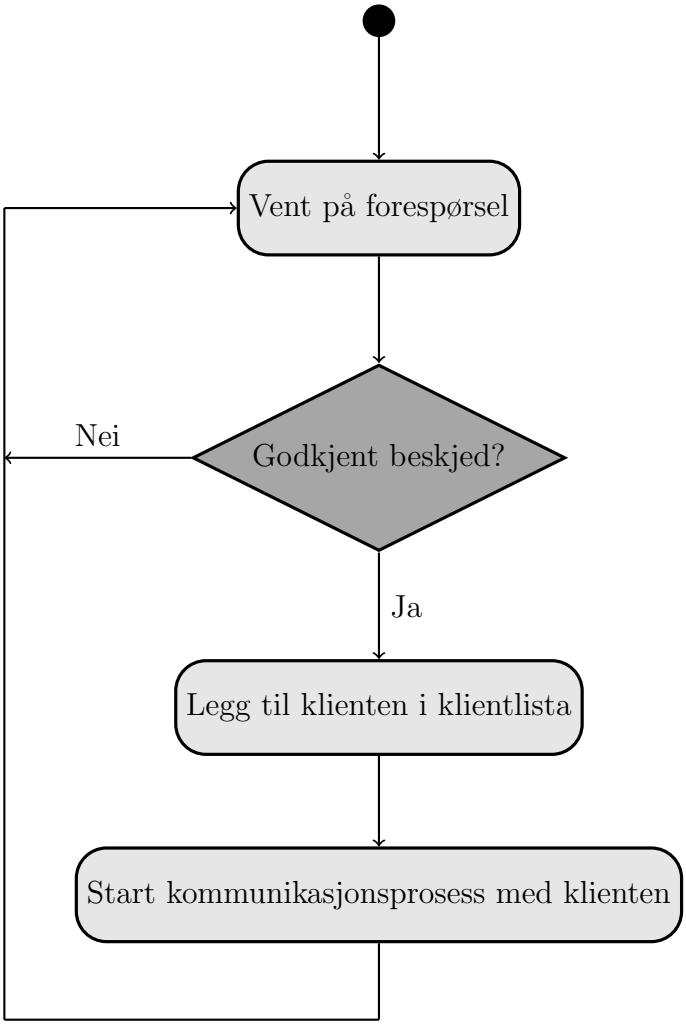
Forespørsler som godkjennes av modulen er vist sammen med format for resulterende output i Tabell 5.

Tabell 5: Inndata og resulterende format på utdata for Distribusjonsmodulen.

input	resulterende output
"body"	byte$[6 \times 4 + 1]$
"color"	byte$[1920 \times 1080 \times (bitsperpixel) + 1]$
"depth"	byte$[512 \times 424 \times (bitsperpixel) + 1]$
"infrared"	byte$[512 \times 424 \times (bitsperpixel) + 1]$

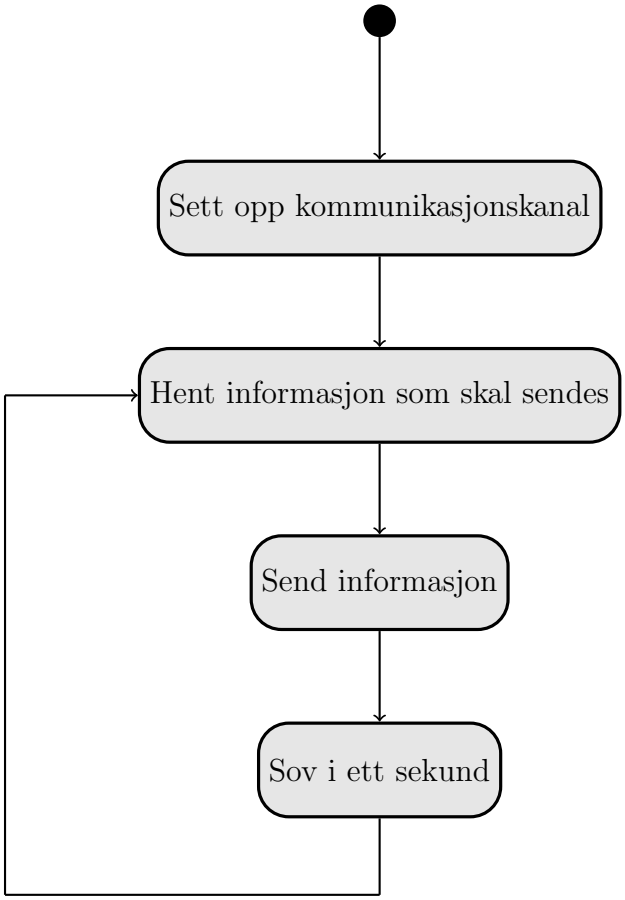
Første tegn i den utsendte byte-tabellen, forteller hva slags data som ligger i byte-tabellen. Tegnet som sendes er en enum av typen *FrameType*. Gyldige verdier for *FrameType* er:

- body (0)



Figur 20: Aktivitetsdiagram for *ListenForClients()*. tilhørende KinectFrameServer-klassen i Distribusjonsmodulen

- color (1)
- depth (2)
- infrared (3)



Figur 21: Aktivitetsdiagram for *SendFrameInfoToClient()* tilhørende KinectFrameServer-klassen i Distribusjonsmodulen.

6.2 Beslutningsmodul

Beslutningsmodulens funksjonalitet er fordelt på tre klasser:

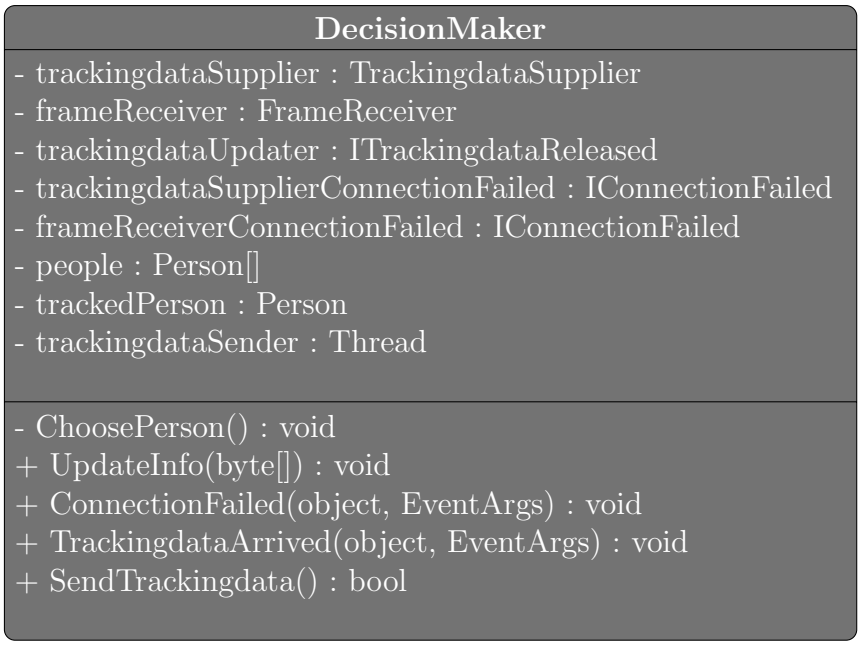
- DecisionMaker
- FrameReceiver
- TrackingdataServer

Klassen *DecisionMaker* står bak modulens hovedfunksjonalitet. Denne holder og behandler posisjonsdataene som modulen mottar. Klassen *FrameReceiver* er koblet opp mot foregående modul, Distribusjonsmodulen, og

mottar posisjonsdata fra denne. Dataen som *FrameReceiver* mottar, videreformidles til *DecisionMaker*-klassen. Klassen *TrackingdataServer* videreformidler posisjonsdata for valgt person å følge videre til henvendende moduler.

DecisionMaker

Planlagt klassediagrammet for *DecisionMaker* er vist i Figur 22.



Figur 22: Klassesdiagram for DecisionMaker i Beslutningsmodulen.

Klassen skal lagre posisjonsdata for personer i variabelen *people*. Denne består av en tabell av variabler av typen *Person*. Dette er en struct som igjen består av fire variabler; en ulong for å holde en persons ID, og tre integers for å holde informasjon om personens posisjon i x-, y- og z-koordinater.

people oppdateres hver gang *DecisionMaker*-klassen mottar nye posisjonsdata. Dette registreres gjennom *positionDataUpdater*, som er en event handler for å fange opp det egenkomponerte eventet *OnNewPositionDataArrived*. Dette eventet settes opp av *FrameReceiver*-klassen når nye posisjonsdata er tilgjengelig. Når dette eventet oppstår, kjøres funksjonen *NewPositionDataArrived()*. Denne funksjonen oppdaterer posisjonsdataen som ligger i *people*.

Når *people* inneholder informasjon om en eller flere personers posisjon, skal en av dem velges ut, og plasseres i variabelen *trackedPerson*. Dette er informasjonen som videreformidles fra *Beslutningsmodulen*. Funksjonen *SendTrackingdata()* tar seg av valg av person, samt å komponere en byte-tabell som inneholder informasjonen som skal videreformidles. Aktivitetsdiagram for *SendTrackingdata()* er vist i Figur 23.

FrameReceiver

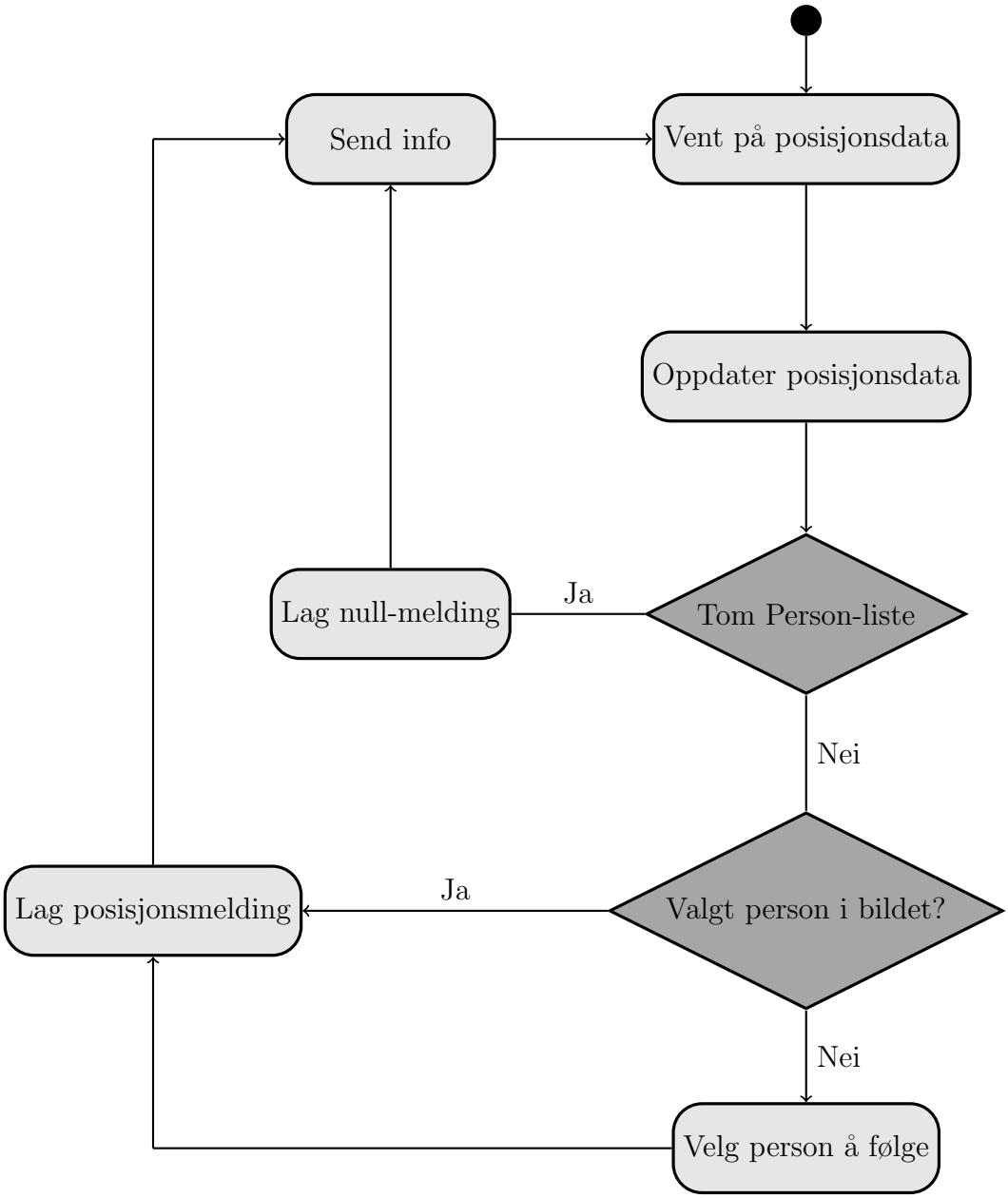
Klassediagram for *FrameReceiver* er vist i Figur 24.

FrameReceiver-klassen skal inneholde en TcpClient som ved oppstart søker å koble seg opp til Distribusjonsmodulen ved å sende teksten "body". Når forespørselen om tilkobling er blitt godkjent, startes funksjonen *ReceiveBodyFrame()* i en egen prosess via klassens *frameReceiver*-tråd. *ReceiveBodyFrame()*-funksjonen venter på nye posisjonsdata fra Distribusjonsmodulen, og setter opp et *newPositionInfoEvent* når nye data er tilgjengelig. Aktivitetsdiagram for ReceiveBodyFrame() er vist i Figur 25.

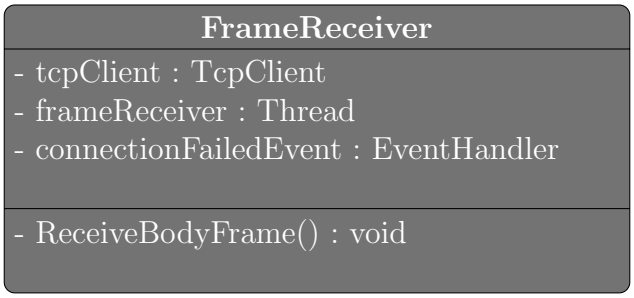
TrackingdataServer

Klassediagram for *TrackingdataServer* er vist i Figur 26.

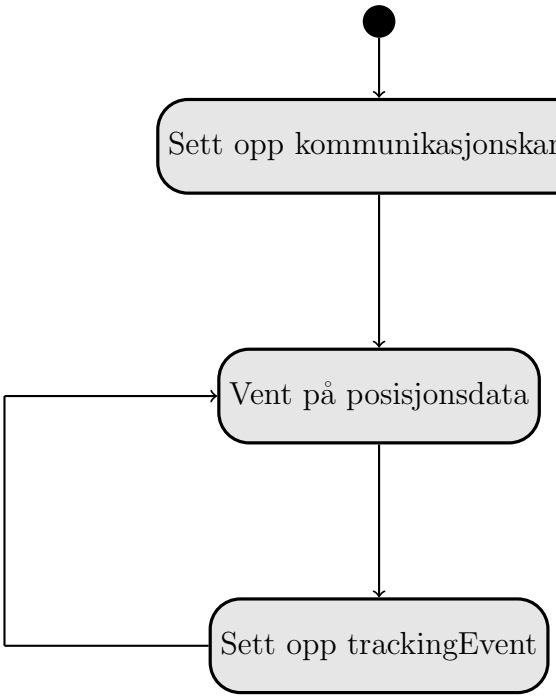
TrackingdataSupplier-klassen skal inneholde funksjonalitet for å sende en byte-tabell med lengde 3 til en potensiell posisjonskontroller. Ved oppstart kobles klassens *tcpClient* seg opp mot en mottager. Når klassens medlemsfunksjon *SendTrackingdata()* kalles, blir byte-tabellen i parameterlista sendt til mottageren.



Figur 23: Aktivitetsdiagram for SendTrackingdata() tilhørende DecisionMaker-klassen i Beslutningsmodulen.



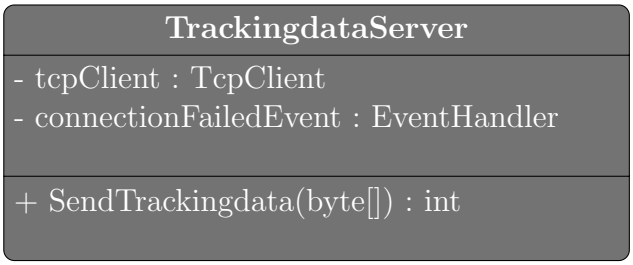
Figur 24: Klassesdiagram for FrameReceiver i Beslutningsmodulen.



Figur 25: Aktivitetsdiagram for ReceiveBodyFrame() tilhørende FrameReceiver-klassen i Beslutningsmodulen.

Feilhåndtering

Dersom kommunikasjonskanalen til en av klassene blir brutt, skal kontakten forsøkes gjenopptatt. Dersom det etter 6 sekunder fortsatt ikke er oppnådd kontakt, skal forsøket avsluttes, og et *connectionFailedEvent* set-



Figur 26: Klassesdiagram for TrackingdataServer i Beslutningsmodulen.

tes opp. Dette eventet skal så fanges opp av *DecisionMaker*-klassen, som avslutter hele Beslutningsmodulen.

6.3 Brukergrensesnittmodul

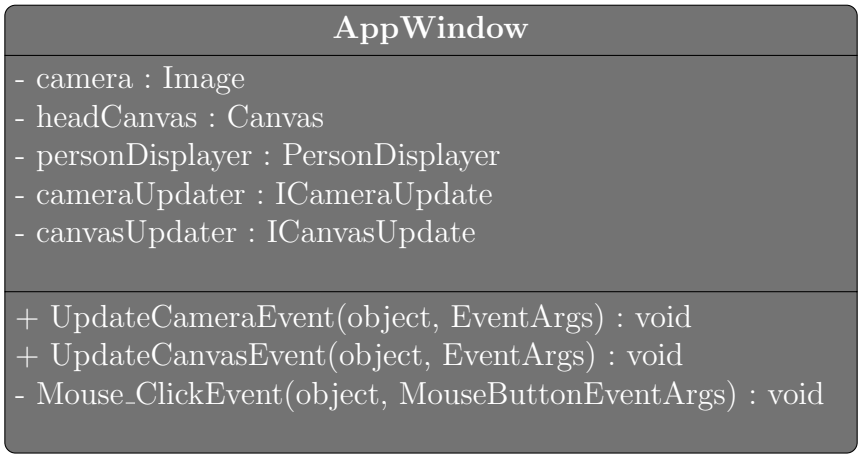
Brukergrensesnittmodulens funksjonalitet skal deles inn i fire klasser

- AppWindow
- FrameReceiver
- PersonDisplayer
- TrackingdataServer

AppWindow er en underklasse av C#s Windows-klasse, og skal vise fargebilder og firkantmarkerte personer i bildet for brukeren. Klassen *FrameReceiver* skal motta informasjon fra Distribusjonsmodulen, både fargebilder og posisjonsdata. *PersonDisplayer* skal lagre og behandle posisjonsdataene som mottas av *FrameReceiver*-klassen, slik at ansikter i fargebildet kan markeres med firkanter. *TrackingdataServer* skal videreformidle posisjonsdata for personen som spores.

AppWindow

Klassediagram for *AppWindow* er vist i Figur 27.



Figur 27: Klassediagram for AppWindow i Brukergrensesnittmodulen.

Denne klassen inneholder et Image kalt *camera* som fargebilder fra Distribusjonsmodulen skal vises i når de blir tilgjengelige. Dette gjøres ved hjelp av funksjonen *UpdateCameraEvent*, som kjøres hver gang et event av typen *OnFrameArrived* registreres.

Firkantene som markerer personers hoder skal tegnes i *headCanvas*. *headCanvas* skal oppdateres ved hjelp av funksjonen *UpdateCanvasEvent()*, som kjører hver gang et event av typen *OnCanvasDataArrived* registreres.

Fargen på firkantene skal kunne endres ved museklikk. Funksjonen *Mouse_ClickEvent()* skal kjøres ved museklikkevent. Funksjonen skal først sjekke om museklikket ble gjort innenfor en firkant. Dersom det ble det, skal det finnes ut hvilken av tre mulige tilstander applikasjonsvinduet befinner seg i:

1. Klikk i grønn rute, ingen orange rute eksisterer
2. Klikk i grønn rute, en orange rute eksisterer
3. Klikk i orange rute

Hva som skal gjøres videre, vil være avhengig av denne tilstanden, pluss hvor i bildet klikket ble gjort. Aktivitetsdiagram for *Mouse_ClickEvent()* er vist i Figur 28.

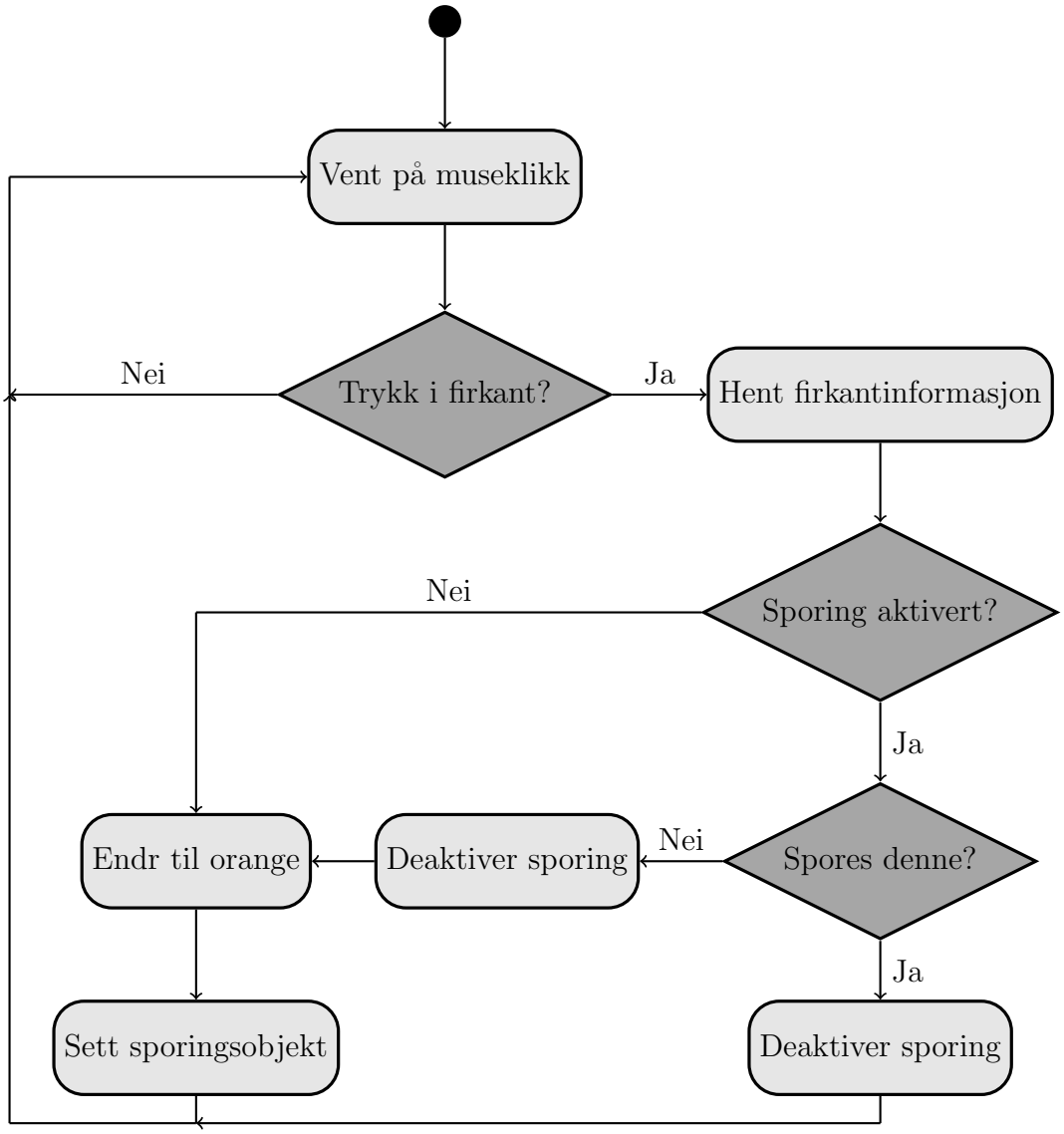
PersonDisplayer

Klassediagram for *PersonDisplayer* er vist i Figur 29.

Denne klassen har mye samme arbeidsoppgaver som klassen *DecisionMaker* i Beslutningsmodulen. Informasjon om personers posisjon skal lagres i *people*, og person som følges kan finnes i *trackedPerson*.

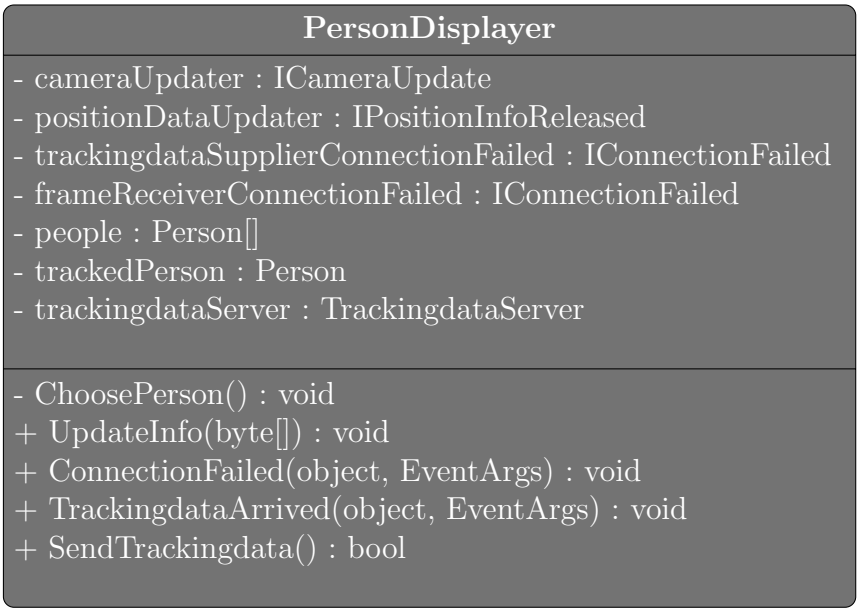
PersonDisplayer skal fange opp event av typen *OnNewPositionDataArrived* som opprettes av *FrameReceiver*-klassen når nye posisjonsdata blir tilgjengelige. Funksjonen *UpdateInfo()* skal da bli kjørt, akkurat som i *DecisionMaker* i Beslutningsmodulen, pluss at et event av typen *OnCanvasDataArrived* skal opprettes.

Forskjellen på Beslutningsmodulen og Brukergrensesnittmodulen, ligger i det at det er brukeren som bestemmer hvilken person som skal følges i Brukergrensesnittmodulen. ID- og posisjonsverdiene i *trackedPerson* skal kun være ulik 0 dersom person å følge er blitt valgt ved trykk i en grønn rute.



Figur 28: Aktivitetsdiagram for Mouse_ClickEvent() i AppWindow-klassen i Brukergrensesnittmodulen.

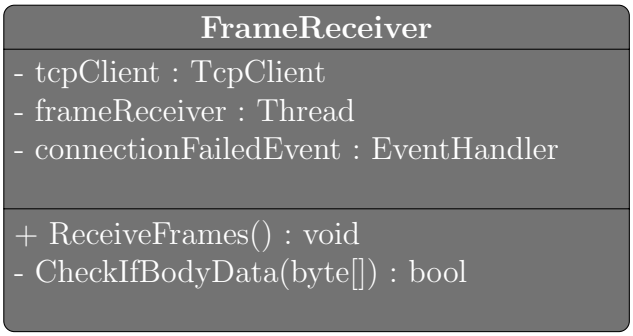
Dersom det ikke er valgt noen person å følge, vil *trackedPerson* være fylt med 0'ere. Det vil da også være en byte-tabell med 3 0'ere som blir sendt ut til en eventuell posisjonskontroller.



Figur 29: Klassediagram for PersonDisplayer i Beslutningsmodulen.

FrameReceiver

Klassediagram for *FrameReceiver* er vist i Figur 30.



Figur 30: Klassediagram for FrameReceiver i Brukergrensesnittmodulen.

Denne klassen skal motta to ulike informasjonstyper fra Distribusjonsmodulen; fargebilder og posisjonsdata. Dette gjøres ved hjelp av én TcpClient som sender to beskjeder til Distribusjonsmodulen, en med "body" og en

med "color". Mottagelse av informasjon fra Distribusjonsmodulen skal skje via funksjonen *ReceiveFrames()* som skal kjøres i tråden *frameReceiver*. Når denne mottar en byte-tabell, skal det først sjekkes hva slags informasjon som er blitt mottatt, ved hjelp av funksjonen *CheckIfBodyData()*. Denne skal returnere *true* dersom det er posisjonsdata i byte-tabellen, og *false* dersom det er bildedata i byte-tabellen.

FrameReceiver-klassen i Brukergrensesnittmodulen fungerer mye på samme måte som *FrameReceiver*-klassen i Beslutningsmodulen, ved at det settes opp eventer når ny informasjon er tilgjengelig. Eventet for nytt fargebilde heter *OnFrameArrived*, mens eventet for ny posisjonsdata heter *OnNewPositionInfoArrived*.

TrackingdataServer

Denne klassen skal fungere på tilsvarende måte som *TrackingdataServer*-klassen i Beslutningsmodulen. Informasjon om personen som følges skal sendes ut med et sekunds mellomrom.

Feilhåndtering

Det samme gjelder for Brukergrensesnittmodulen som for Beslutningsmodule når det kommer til feilhåndtering. Dersom det oppstår en kommunikasjonsfeil med en av klientene, enten i *FrameReceiver* eller *TrackingdataServer*, skal et event av typen *connectionFailedEvent* settes opp. Dette eventet skal så fanges opp av *AppWindow*, som avslutter hele Brukergrensesnittmodulen.

7 Implementasjon

7.1 Distribusjonsmodul

Implementasjonen av Distribusjonsmodulen har gått bra etter planen. Det oppstod ingen store problemer under implementeringen, og systemet er blitt som planlagt. Kildekode for modulen finnes i elektronisk vedlegg i mappen "FrameSupplier" under "Moduler". Brukermanual for Distribusjonsmodulen er gitt i Vedlegg A.

7.2 Beslutningsmodul

Implementasjonen av Beslutningsmodulen gikk som planlagt, og ganske problemfritt. Kildekode for modulen finnes i elektronisk vedlegg i mappen "DecisionMaker" under "Moduler".

Foreløpig er det implementert to måter å velge person for plassering i *trackedPerson*-variabelen:

ChooseRandomPerson() velger en tilfeldig person fra *people* inn i *trackddPerson*

ChooseNearestPerson() velger den personen som befinner seg nærmest Kinecten

Nye metoder kan legges til ved senere anledninger, dersom det er ønskelig.

7.3 Brukergrensesnitt

Implementasjonen av Brukergrensesnittmodulen har ikke blitt helt fullført. Designet har vist seg å muligens kunne være en smule optimistisk i forhold til beregnet implementasjonstid. Kildekode for modulen finnes i elektronisk vedlegg i mappen "BodyTracking" under "Moduler".

7.3.1 Problemer som har oppstått

Alle klassene er satt opp, og mye av funksjonaliteten i klassene er implementert, samt eventer og event handlere for intern kommunikasjon og signalisering. Implementeringen stoppet etterhvert litt opp, da det viste seg å dukke opp flere hindre på veien.

Det viste seg blant annet å være et problem knyttet til det å be om to ulike informasjonstyper fra Distribusjonsmodulen over samme kanal. Dersom det bes om posisjonsdata først, og bildedata etterpå, sender Distribusjonsmodulen kun posisjonsdata, og ignorerer forespørselen om bildedata. Dersom bildedata sendes først, blir det omvendt.

Det ser altså ut til at det er et problem for Distribusjonsmodulen å motta flere forespørsler fra samme klient, da det viser seg at det går fint an å motta forespørsler fra flere ulike klienter. Årsaken til dette er det ikke kommet helt til bunns i. Brukergrensesnittmodulen sender ut begge sine forespørsler, men det er kun den første som mottas hos Distribusjonsmodulen, som blir sittende og vente på nye forespørsler.

I tillegg til dette, viser det seg å være vanskeligere en antatt å få vist fargebilder i applikasjonsvinduet. Ved første implementasjonsforsøk dukket det opp et helt bilde ved første overføring, mens oppdateringen av bildet ikke gikk helt som planlagt. Når et nytt bilde skulle vises, ble bare toppen av bildet oppdatert, mens resten av det gamle bildet ble liggende igjen. Dette resultatet ble oppnådd ved bruk av et *Image* [Tawate, 2009].

I et forsøk på å fikse dette problemet, ble det undersøkt nye metoder og format for å legge bilder inn i en *Image.Source*. En mulig løsning viste seg å være å benytte et *BitmapImage* [Microsoft, 2015b], men denne løsningen resulterte i en minnehåndteringsfeil som det viste seg å være vanskelig å løse.

Videre søk etter mulige løsninger gav et tips om å benytte en *BitmapSource* [Pterneas, 2014], men uten endring av resultat. Minnehåndteringsfeilen bestod.

Ved hjelp av skriving til fil, er det funnet ut at bildedata ankommer Brukergrensesnittets *FrameReceiver* i riktig format. Det ser dermed ut til at problemet ligger mest i det å oversette en byte-tabell med bildeinformasjon til et passende format som kan plasseres i en *Image.Source*, og som ikke hoper seg opp i minnet når nye objekter av formatet skal lages.

7.3.2 Funksjonalitet som gjenstår

Da det har oppstått en del problemer, ble det valgt å ikke gå videre før disse eventuelt ble løst. Dette er grunnen til at det da også mangler en del annen funksjonalitet. Det ble ansett som svært uhensiktsmessig å utvide koden, når den eksisterende koden ikke fungerte.

Hovedfunksjonaliteten som da mangler er:

- funksjonalitet knyttet til valg av person / museklikk i firkanter

- videresending av posisjonsdata for person som følges

7.3.3 Endringer i designet

I tillegg til de ovenfornevnte hindringene, er det blitt en endring i forhold til det planlagte designet.

Varsel til *AppWindow* om nytt fargebilde fra *FrameReceiver* går via *PersonDisplayer*, istedenfor direkte, slik som planlagt i designet. Når et *OnFrameArrived*-event settes opp av *FrameReceiver*, fanger *PersonDisplayer* opp dette, og setter opp et nytt event av samme typen, som igjen fanges opp av *AppWindow*, som da oppdaterer bildet.

8 Formulering av tester

Under følger en rekke tester som skal utføres på systemet. Testene er laget med utgangspunkt i kravene som er listet i Delkapittel 5.8.

8.1 Distribusjonsmodul

Tester for distribusjonsmodulen:

Test 1: **Testens omfang:** teksten "body" sendes til modulen i en byte-tabell
Forventet resultat: distribusjonsmodulen sender en byte-tabell med nyeste informasjon fra BodyFrameReader tilbake med ett sekunds mellomrom
Krav som testes: 6, 8, 13

Test 2: **Testens omfang:** teksten "color" sendes til modulen i en byte-tabell
Forventet resultat: distribusjonsmodulen sender en byte-tabell med nyeste informasjon fra ColorFrameReader tilbake med ett sekunds mellomrom
Krav som testes: 6, 9, 12

Test 3: **Testens omfang:** teksten "depth" sendes til modulen i en byte-tabell
Forventet resultat: distribusjonsmodulen sender en byte-tabell med nyeste informasjon fra DepthFrameReader tilbake med ett sekunds mellomrom
Krav som testes: 6, 10, 12

Test 4: **Testens omfang:** teksten "infrared" sendes til modulen i en byte-tabell
Forventet resultat: distribusjonsmodulen sender en byte-tabell med nyeste informasjon fra InfraredFrameReader tilbake med ett sekunds mellomrom
Krav som testes: 6, 11, 12

Test 5: **Testens omfang:** teksten "body" sendes til modulen i en byte-tabell, fra to ulike avsendere
Forventet resultat: distribusjonsmodulen sender en byte-tabell med nyeste informasjon fra BodyFrameReader til begge
Krav som testes: 2, 4

Test 6: **Testens omfang:** en klient som mottar posisjonsdata fra Distribusjonsmodulen avsluttes
Forventet resultat: prosessen som kommuniserer med klienten avsluttes, og klienten fjernes fra klientlisten
Krav som testes: 5, 7

Test 7: **Testens omfang:** fem ulike klienter prøver å kontakte Distribusjonsmodulen. Klient 1 sender beskjeden "body", klient 2 sender beskjeden "color", klient 3 sender beskjeden "depth", klient 4 sender beskjeden "infrared", og klient 5 sender beskjeden "trala"
Forventet resultat: Klient 1, 2, 3 og 4 skal godkjennes av Distribusjonsmodulen og legges til i klientlista. Forespørselen fra klient 5 skal avslås
Krav som testes: 3

8.2 Beslutningsmodul

Tester for beslutningsmodulen:

Test 1: **Testens omfang:** et program kobler seg opp mot Beslutningsmodulen, og lytter etter resultater
Forventet resultat: programmet mottar nye beskjeder hver sekund
Krav som testes: 2, 3, 4

Test 2: **Testens omfang:** informasjonen som sendes til modulen endres
Forventet resultat: informasjonen som sendes fra modulen endres
Krav som testes: 8

Test 3: **Testens omfang:** person(er) står foran Kinecten, og posisjonsdata mottas av Beslutningsmodulen
Forventet resultat: informasjonen som sendes fra modulen inneholder verdier større enn null
Krav som testes: 9

Test 4: **Testens omfang:** ingen står foran Kinecten, og en byte-tabell fylt med 0ere mottas av Beslutningsmodulen
Forventet resultat: informasjonen som sendes fra modulen inneholder tre 0
Krav som testes: 10, 13

Test 5: **Testens omfang:** to personer står foran Kinecten
Forventet resultat: kun informasjon om en av dem videreformidles
Krav som testes: 11

Test 6: **Testens omfang:** person går inn og ut av Kinectens synsvidde
Forventet resultat: når personen er i bildet, sendes posisjonsinformasjon for denne personen. Når personen ikke er i bildet, sendes 0'ere
Krav som testes: 12

Test 7: **Testens omfang:** program som mottar posisjonsdata fra Beslutningsmodulen avsluttes, og startes opp igjen
Forventet resultat: Beslutningsmodulen prøver å gjenoppta kontakten, og fortsetter å sende data når den lykkes
Krav som testes: 14

Test 8: **Testens omfang:** program som mottar posisjonsdata fra Beslutningsmodulen avsluttes
Forventet resultat: Beslutningsmodulen prøver å gjenoppta kontakten i 6 sekunder før den gir opp
Krav som testes: 14, 15

8.3 Brukergrensesnitt

Tester for brukergrensesnittmodulen:

Test 1: **Testens omfang:** sende posisjonsdata for person i Kinectens synsvidde og fargebilde til modulen
Forventet resultat: fargebildet vises i applikasjonsvinduet, og personens hode markeres med en grønn firkant
Krav som testes: 2, 3, 4, 7, 8, 10

Test 2: **Testens omfang:** det befinner seg en person i bildet, og denne er markert med grønn firkant rundt hodet. Det skal gjøres to klikk i firkanten, med tre sekunders mellomrom
Forventet resultat: første klikk resulterer i at firkanten blir orange (følging aktivert). Andre klikk resulterer i at firkanten blir grønn (følging deaktivert)
Krav som testes: 9, 11, 13

Test 3: **Testens omfang:** to personer befinner seg i bildet, markert med grønne firkanter rundt hodet. Det skal igjen trykkes to ganger, en gang i hver firkant
Forventet resultat: trykk i første firkant skal resultere i at firkanten blir orange (følging aktivert for første person). Andre klikk skal resultere i at den første firkanten blir grønn igjen, mens den det ble trykket i blir orange (følging aktivert for andre person)
Krav som testes: 12, 13, 14, 15

Test 4: **Testens omfang:** person står foran Kinecten
Forventet resultat: informasjon om personens posisjon sendes fra modulen
Krav som testes: 16

Test 5: **Testens omfang:** ingen står foran Kinecten
Forventet resultat: informasjonen som sendes fra modulen inneholder tre 0
Krav som testes: 17

Test 6: **Testens omfang:** program som mottar posisjonsdata fra Beslutningsmodulen avsluttes, og startes opp igjen
Forventet resultat: Beslutningsmodulen prøver å gjenoppta kontakten, og fortsetter å sende data når den lykkes
Krav som testes: 14

Test 7: **Testens omfang:** program som mottar posisjonsdata fra Beslutningsmodulen avsluttes
Forventet resultat: Beslutningsmodulen prøver å gjenoppta kontakten i 6 sekunder før den gir opp
Krav som testes: 14, 15

9 Verifisering

Dette kapittelet går gjennom alle kravene som ble formulert i Delkapittel 5.8, og angir hvorvidt de er oppfylt eller ikke. Kravene som er testbare er oppfylt dersom testen(e) deres i Kapittel 8 gir forventet resultat. De ikke-testbare kravene er oppfylt dersom koden bekrefter dem. Oppfylte krav er markert med +, mens ikke-oppfylte krav er markert med −.

9.1 Distribusjonsmodul

For Distribusjonsmodulen er de ikke-testbare kravene vist i Tabell 6, mens de testbare er vist i Tabell 7.

Tabell 6: Ikke-testbare krav for Distribusjonsmodulen.

Krav nr.	Krav	Resultat
1	inn- og utdata for modulen er på byte[]-format	+

9.2 Beslutningsmodul

For Beslutningsmodulen er de ikke-testbare kravene vist i Tabell 8, mens de testbare er vist i Tabell 9.

9.3 Brukergrensesnittmodul

For Brukergrensesnittmodulen er de ikke-testbare kravene vist i Tabell 10, mens de testbare er vist i Tabell 11.

Tabell 7: Testbare krav for Distribusjonsmodulen.

Krav nr.	Krav	Test nr.	Resultat
2	forespørsler om tilkobling fra nye klienter skal mottas kontinuerlig	5	+
3	kun klienter med forespørsel lik "color", "body", "depth" eller "infrared" skal godkjennes	7	+
4	hver nye tilkobling skal håndteres av en egen prosess	5	+
5	prosessen avsluttes når forbindelsen går tapt	6	+
6	når en ny klient får koble seg til, skal den legges til i klientlisten	1, 2, 3, 4	+
7	når en klient kobles fra/mister forbindelsen med modulen, skal den fjernes fra klientlisten	6	+
8	når modulen mottar en forespørsel med melding "body", skal den med ett sekunds mellomrom sende ut oppdatert posisjonsinformasjon til klienten forespørselen kom fra	1	+
9	når modulen mottar en forespørsel med melding "color", skal den med ett sekunds mellomrom sende ut et nytt fargebilde til klienten forespørselen kom fra	2	+
10	når modulen mottar en forespørsel med melding "depth", skal den med ett sekunds mellomrom sende ut et nytt dybdebilde til klienten forespørselen kom fra	3	+
11	når modulen mottar en forespørsel med melding "infrared", skal den med ett sekunds mellomrom sende ut et nytt infrarødbilde til klienten forespørselen kom fra	4	+
12	lengden på bildemeldingene som går ut fra modulen er på $(bildebredde) \times (bildehøyde) \times (bitperpiksel) + 1$ byte	2, 3, 4	+
13	lengden på posisjonsmeldingene er på $(antallmuligepersoneribildet) \times (antalldatafelt) + 1$ byte	1	+

Tabell 8: Ikke-testbare krav for Beslutningsmodulen.

Krav nr.	Krav	Resultat
1	inn- og utdata for modulen er på byte[]-format	+
5	inndata skal mottas i egen prosess	+
6	utdata skal sendes i egen prosess	+
7	posisjonsdata skal lagres i modulen	+

Tabell 9: Testbare krav for Beslutningsmodulen.

Krav nr.	Krav	Test nr.	Resultat
2	lengden på inndata er på $(antallmuligepersoneribildet) \times (antalldatafelt) + 1$ byte, og inneholder posisjonsdata	1	+
3	lengden på utdata er $(antallkoordinaterforposisjon)$ byte, og inneholder informasjon om personen som følges	1	+
4	utdata sendes med ett sekunds mellomrom	1	+
8	når nye meldinger mottas, skal posisjonsdata i modulen oppdateres	2	+
9	tilgjengelige personer representeres med tall-verdier større enn null	3	+
10	manglende personer representeres med 0'ere	4	+
11	kun én person skal følges av gangen	5	+
12	så lenge det er personer i bildet, skal det være valgt en person å følge	6	+
13	dersom det ikke er registrert personer i bildet , blir utdata fylt med 0'ere	4	+
14	dersom tilkoblingen til en av de andre modulene brytes, skal det bli forsøkt å opprette ny tilkobling i 6 sekunder før kontakten anses for å være tapt	7, 8	+
15	når kontakten med en av de andre modulene anses å være tapt, skal kontakten med den andre modulen også brytes, og applikasjonen avsluttes	8	+

Tabell 10: Ikke-testbare krav for Brukergrensesnittmodulen.

Krav nr.	Krav	Resultat
1	inn- og utdata for modulen er på byte[]-format	+
18	dersom tilkoblingen til en av de andre modulene brytes, skal det bli forsøkt å opprette ny tilkobling i 6 sekunder før kontakten anses for å være tapt	–
19	når kontakten med en av de andre modulene anses å være tapt, skal kontakten med den andre modulen også brytes, og applikasjonen avsluttes	–

Tabell 11: Testbare krav for Brukergrensesnittmodulen.

Krav nr.	Krav	Test nr.	Resultat
2	modulen skal kunne ta imot to ulike inndata parallelt	1	–
3	den ene inndataen skal være posisjonsdata	1	+
4	den andre inndataen skal være fargebilde	1	+
7	når dataene i modulen oppdateres, skal de vises for brukeren	1	–
8	posisjonsdata skal resultere i firkantmarkeringer rundt personer hode	1	–
9	firkantene skal kunne ha to ulike farger	2	–
10	grønne firkant skal representere en person som ikke følges	1	–
11	orange firkant skal representere en person som følges	2	–
12	kun én person skal følges av gangen	3	–
13	trykk i grønn firkant skal aktivere følging av personen firkanten tilhører	2, 3	–
14	trykk i orange rute skal deaktivere følging av personen firkanten tilører	3, 7	–
15	ved aktivering av ny person å følge, skal tidligere aktivert følging av en person deaktiveres først	3, 7	–
16	dersom person å følge er satt, skal utdata settes til x-, y- og z-posisjon for denne personen	4	–
17	dersom person å følge ikke er satt, skal utdata fylles med 0'ere	5	–

10 Diskusjon

10.1 Vurdering av det resulterende systemet

Resultatet er ikke blitt helt som forventet. Det oppstod en del uforutsette problemer, som viste seg å være vanskeligere å løse enn først antatt.

Systemet fungerer som ønsket ved automatisk valg av person, altså ved valg av person å følge ved hjelp av Beslutningsmodulen. Denne modulen velger en person i bildet, og videreformidler denne personens posisjon. Nye måter å velge person å følge på kan legges til i klassen når ønsket.

En interessant måte å velge person å følge på, ville vært å få den organiske delen av kyborgen, nemlig cellekulturen, til å ta seg av valget. Dette kan for eksempel gjøres ved at den gjør et tilfeldig valg av tall større eller lik 1 og mindre eller lik antallet personer i bildet, altså tilfeldig valg.

Når det kom til implementasjon av Brukergrensesnittmodulen, dukket det derimot opp en del problemer. Det at denne ikke ble ferdig er trist med tanke på at det hadde vært veldig artig funksjonalitet å tilføre kyborgen. Samtidig er det ikke kritisk for systemet, da Beslutningsmodulen fortsatt fungerer, og valg av person å følge kan gjennomføres, bare ikke overstyres.

At en del av den planlagte funksjonaliteten mangler i Brukergrensesnittmodulen er forsåvidt dumt, men det ville vært enda dummere å legge til mer kode i et ikke-fungerende program, og muligens lage en enda større floke. Det anses dermed som en god avgjørelse at utviklingen ble stoppet da den ble stoppet.

10.2 Forbedringer i design

Det sies at det er enkelt å være etterpåklok, og i visse tilfeller kan det også være veldig lurt. I forbindelse med systemdesignet som ble presentert i Kapittel 6, er det oppdaget en del mulige forbedringer.

Enum i stedet for string for dataforespørsel: dersom det hadde vært benyttet en enum med kun de fire informajonstypene *body*, *color*, *depth* og *infrared* som mulige verdier, ville dette både fortkortet lengden på meldingene som blir sendt til Distribusjonsmodulen, samt at det forenkler kontroll av gyldige meldinger.

Ikke faste IP-adresser: foreløpig er systemet satt opp med faste IP-adresser for tilkobling, og disse må endres til å stemme overens med den ma-

skinen det er ønsket å koble til før systemet kan kobles opp korrekt. Å ikke ha dette hardkodet i modulene, gjør systemet mer motstandsdyktig mot geografiske forflytninger og omstart.

Distribusjonsmodulen bør sende ut informasjon etter forespørsel: slik systemet funker nå, sender klienter kun forespørsler til Distribusjonsmodulen ved oppstart. Dersom de godkjennes av Distribusjonsmodulen, blir de tilsendt ny informasjon hvert sekund. En bedre løsning, ville vært om klientene sendte ut nye forespørsler hver gang de ønsket ny informasjon, og at Distribusjonsmodulen kun svarte på disse. Det ville gjort systemet mer responsivt, og enklere å regulere for utviklere av nye moduler. Slik det fungerer nå, er systemet (i utgangsounktet) nødt til å motta ny informasjon hvert sekund.

Distribusjonsmodulen: tilbakemelding til moduler som avslås: slik det er nå, sendes det ingen respons tilbake til moduler som ikke godkjennes av Distribusjonsmodulen. Det burde sendes en respons som sier noe om hvorfor henvendelsen ikke ble godkjent.

Senke oppløsning på bildeinformasjon: dersom oppløsningen på bildeinformasjonen reduseres, blir byte-tabellene som sendes fra Distribusjonsmodulen til henvendende moduler mindre. Dette vil kunne gjøre kommunikasjonen mellom modulene mer effektiv, samt minke behandlingstiden av bildene på klientsiden.

Oppgradering av format på byte-tabellene: for å forbedre kommunikasjonen mellom modulene, burde formatet på byte-tabellene som sendes forbedres. Ved å innføre en protokoll der alle byte-tabeller starter med et tall som forteller hvor lang byte-tabellen er, samt at den ender med et termineringstegn, vil det bli lettere å skille ut meldinger som ikke skal aksepteres. Det vil også kunne bli mulig å be om to typer informasjon i en og samme melding. Dette kan muligens løse problemet med å be om flere typer informasjon fra til en og samme modul.

10.3 Vurdering av totalarbeidet

Tidsplanleggingen fra Delkapittel 4.2 fungerte svært bra de første ukene av prosjektet. Det var lagt av godt med tid til hver av fasene, og enkelte av dem

ble avsluttet noe før tiden, slik at neste fase kunne starte opp tidligere. Det var dermed fra starten av gjort forsøk på å spare opp litt tid.

Allikevel ble det knapt med tid mot slutten av prosjektet. Design- og implementasjonsfasene tok mer tid enn planlagt, og særlig implementasjonen ble mer krevende enn antatt. Problemene som oppstod ble større enn hva den gjenværende tiden gav mulighet for å rette opp i. Dermed ble ikke systemet fullført helt etter planen.

Selv om det ble gjort forsøk på å tidlig i prosjektet spare inn til et slags tidsbuffer, ble det altså uansett ikke nok arbeidstimer i uka til å fullføre hele prosjektet.

Med dagens kunnskap rundt tidsbruken i prosjektet, ville nok tidsforløpet for prosjektet sett annerledes ut enn det som var planlagt i Figur 15 i Delkapittel 4.2. Forarbeidstiden ville nok da ha blitt redusert til fordel for økt implementasjonstid.

En alternativ, og kanskje bedre løsning, ville vært å forenklet systemdesignet, da det muligens kan ha vært en overvurdering av egen GUI-kompetanse som er årsaken til problemet.

11 Videre arbeid

Det videre arbeidet med systemet vil bestå av vedlikehold, optimalisering/forbedringer i design, feiloppretting og videreutvikling.

Som påpekt i Kapittel 7 og Kapittel 10 har det gjennom implementasjon og testing blitt oppdaget en del hindringer for, samt forbedringer til, designet. Det videre arbeidet vil fokusere på å få i stand et velfungerende og feilfritt system, som gjør det mulig å velge person å følge både manuelt og uten ytre påvirkninger.

Det videre arbeidet vil dermed innebære:

- **Vedlikehold**
- **Forbedringer i design**
  - bytte ut stringformuleringer som for eksempel "body" og "color" med enum-typer, som for eksempel *FrameType.body* og *FrameType.color*
  - avvikle faste IP-adresser for modulene
  - endre Distribusjonsmodulen slik at den ikke sender ut informasjon med et sekunds mellomrom, men heller etter forespørsel
  - implementere funksjonalitet slik at moduler som avslås av Distribusjonsmodulen mottar tilbakemelding om hvorfor de ble avslått
  - redusere kvaliteten på bildeinformasjonen som sendes
  - forbedre formatet på byte-tabellene som sendes mellom modulene
- **Feiloppretting**
  - gjøre det mulig for Brukergrensesnittmodulen å motta både fargebilde og posisjonsdata fra Distribusjonsmodulen parallelt
  - finne en måte å oppdatere bildevisning i Brukergrensesnittmodulen på uten at den fører til minnelekkasje
- **Videreutvikling**
  - legge til flere funksjoner for valg av person å følge
  - utvikle posisjonskontroller for regulering av kyborgens posisjon

- legge til funksjonalitet knyttet til valg av person å følge i Brukergrensesnittmodulen
- legge til funksjonalitet for videresending av informasjon om valgt person å følge fra Brukergrensesnittmodulen

A Appendix A

Brukermanual for
Distribusjonsmodulen

En del av rapporten
NTNU-CYBORG: OPPSØKING
AV SOSIALE SITUASJONER

Skrevet av
Stine Lilleborge

Våren 2015

NTNU
Fakultet for informasjonsteknologi,
matematikk og elektroteknikk
Institutt for teknisk kybernetikk

1 Hva gjør Distribusjonsmodulen?

Distribusjonsmodulen leser bilde- og posisjonsdata fra en Microsoft Kinect v2-sensor. Det finnes tre ulike bildetyper

1. fargebilde
2. dybdebilde
3. infrarødbilde

Alle disse, inkludert posisjonsdata for personer som befinner seg i Kinectens synsfelt, videreformidles av Distribusjonsmodulen til moduler som har koblet seg opp mot den.

2 Hvordan ser informasjonen den sender ut?

Informasjonen som sendes ut fra Distribusjonsmodulen ser annerledes ut ettersom hvilken informasjon den sender ut.

Meldingsformat er vist i Tabell 1.

Tabell 1: Utdata for Distribusjonsmodulen

informasjonstype	format på output
posisjonsdata	byte$[6 \times 4]$
fargebilde	byte$[1920 \times 1080 \times (bitsperpixel)]$
dybdebilde	byte$[512 \times 424 \times (bitsperpixel)]$
infrarødbilde	byte$[512 \times 424 \times (bitsperpixel)]$

Alle data sendes i byte[]-format, og må konverteres til ønskelig format etter mottaging.

For bildedataene, ligger det informasjon om en piksel i bildet i hver byte. Høyde og bredde på bildet er som gitt i Tabell 1, og dette må tas hensyn til når det konverteres til bildeformat.

For posisjonsdata, vil det ligge informasjon om opptil 6 personer i meldingen. For hver person, er det 4 felter med informasjon;

1. identifikasjonsnr

2. x-posisjon: horisontal posisjon

3. y-posisjon: vertikal posisjon

4. z-posisjon: avstand fra Kinecten

Dataene er systematisert slik at all informasjon tilknyttet en person ligger samlet. Rekkefølgen på hver av informasjonsfeltene er som listet.

3 Hvordan koble seg til Distribusjonsmodulen

For å koble seg til Distribusjonsmodulen, trengs det et TCP-klient, samt en IP-adresse for Distribusjonsmodulen.

For å kunne motta data, må TCP-klienten koble seg opp mot Distribusjonsmodulens IP-adresse, og sende en av følgende meldinger:

- "body"
- "color"
- "depth"
- "infrared"

Dersom beskjeden "body" sendes, vil Distribusjonsmodulen sende posisjonsdata tilbake. Dersom beskjeden "color" sendes, vil Distribusjonsmodulen sende fargebildeinformasjon tilbake. Dersom beskjeden "depth" sendes, vil Distribusjonsmodulen sende dybdebildeinformasjon tilbake. Dersom beskjeden "infrared" sendes, vil Distribusjonsmodulen sende infrarødbildeinformasjon tilbake.

All informasjon vil sendes ut fra Distribusjonsmodulen med ett sekunds mellomrom.

Referanser

[Bræk, 1982] Bræk, R. (1982). *Håndbok i systemarbeid*. Tapir Akademisk forlag.

[Collins, 2006] Collins, R. (2006). Mean-shift tracking. Credit Yaron Ukrainitz and Bernard Sarel.

[Eisler, 2012] Eisler, C. (2012). Starting february 1, 2012: Use the power of kinect for windows to change the world.

[Fowler, 2004] Fowler, M. (2004). *UML distilled*. Pearson Education Inc., 3 edition.

[Hacks, 2011] Hacks, K. (2011). Top 10 best kinect hacks.

[Hansen and Hjertø, 2006] Hansen, T. B. and Hjertø, G. (2006). Utviklingsmodeller. Avdeling for informatikk og e-læring, Høgskolen i Sør-Trøndelag.

[Hollister, 2013] Hollister, S. (2013). We try intel's perceptual computing, play 'portal 2' with a wave of the hand and reach into a computer screen.

[Iglesia, 2013] Iglesia, D. (2013). The rangeinet.

[Lilleborge, 2014] Lilleborge, S. (2014). Funksjonsspesifikasjon for sosial robot: NTNU-Cyborg. Prosjektrapport ved institutt for teknisk kybernetikk, NTNU.

[MacCormick,] MacCormick, J. How does the kinect work? Weizmann Institute.

[Microsoft, 2015a] Microsoft (2015a). Colorframereader.framearrived event.

[Microsoft, 2015b] Microsoft (2015b). How to: Use a bitmapimage.

[Microsoft, 2015c] Microsoft (2015c). Interfaces (c# programming guide).

[Microsoft, 2015d] Microsoft (2015d). Kinect for windows.

[Microsoft, 2015e] Microsoft (2015e). Kinect for windows features.

[Microsoft, 2015f] Microsoft (2015f). Set up the hardware.

[Microsoft, 2015g] Microsoft (2015g). Techincal documentation and tools.

[Mikalsen, 1999] Mikalsen, A. B. (1999). Leksjon: 7, utforming av system/kravspesifikasjon.

[Miller, 2011] Miller, M. J. (2011). Primesense: Motion control beyond the kinect.

[Mobilerobots, 2015] Mobilerobots (2015). Pioneer lx research platform.

[Narcisse, 2011] Narcisse, E. (2011). Portal 2 review: Our first perfect 10.

[Nævra, 2014] Nævra, J. (2014). Robotteknisk forstudium av sosial robot - the NTNU-Cyborg. Prosjektoppgave ved Institutt for teknisk kybernetikk, NTNU.

[Pradeep, 2014] Pradeep (2014). Kinect for windows v2 sensor will be available from july 15th.

[Pterneas, 2014] Pterneas, V. (2014). Kinect for windows version 2: Color, depth and infrared streams.

[Ribu, 2004] Ribu, K. (2004). Systemutvikling.

[Smeenk, 2014] Smeenk, R. (2014). Kinect v1 and kinect v2 fields of view compared.

[StackOverflow, 2009] StackOverflow (2009). Understanding events and event handlers in c#.

[Tawate, 2009] Tawate, R. (2009). C# image to byte array and byte array to image converter class.

[The Chaos Computer Club, 2011] The Chaos Computer Club, C. (2011). Kinectfusion - real-time 3d reconstruction and interaction using a moving depth camera [28c3]. CCC is Europe's largest association of hackers. The CCC is based in Germany and other German-speaking countries.

[Thirumuruganathan, 2010] Thirumuruganathan, S. (2010). Introduction to mean shift algorithm. PhD candidate in Computer Science at University of Texas at Arlington. Majors in Data Mining, Artificial Intelligence and Machine Learning.

[WordPress & Atahualpa, 2015] WordPress & Atahualpa (2015). Waterfall model.