NTNU – Trondheim
Norwegian University of
Science and Technology

TTK4851: Robot & menneske

Autonom sykeseng

Prosjektrapport

Tommy Tollin
Erlend Hårstad
Lasse Henriksen
Michael McMillan
Sondre Steinsland Hegdal
Hauk-Morten Heimlund Lykke

3. mai 2017

Abstrakt

Dette prosjektet er utarbeidet som en del av emnet Eksperter i Team ved NTNU, og emnet representerer et tverrfaglig samarbeid. Emnet er obligatorisk for de fleste studenter ved NTNU, og er ment som god trening i samarbeid og prosjektarbeid som venter senere i arbeidslivet.

Landsbyen heter Robot og Menneske, og fokuserer på en teknisk tilnærming til ulike problemstillinger rundt menneske-robot interaksjon. Gruppen har valgt å kombinere ulike moduler som datasyn, bildestrøm, banegenerering og et kontrollsystem for å styre en NXT Lego-robot igjennom en konstruert bane med hindringer. Denne oppgaven er basert på et forstudie gjort av behovet for en autonom sykeseng på St. Olavs Hospital i Trondheim. Roboten representerer noen av de grunnleggende funksjonene til denne sengen i miniatyr.

Det ble tidlig definert spesifikke mål for hva den ferdige løsningen skulle takle av utfordringer. Hver modul fungerte bra hver for seg mot slutten av prosjektet. Da de ulike modulene skulles kobles sammen oppstod det komplikasjoner. På grunn av tidsrammen, ble det tatt en beslutning om å avslutte testfasen selv om roboten ikke var i stand til å oppfylle målene som ble satt. Løsningen legger derimot til rette for videre arbeid og kan være nyttig til framtidige prosjekt knyttet til lignende problemstillinger.

Innhold

1 Introduksjon

1.1 Valg av prosjektoppgave

Ved oppstart av prosjektet ble det gjort en kartlegging av kompetansen til alle gruppemedlemmene. Ettersom alle medlemmene var fra tekniske studieretninger var det naturlig at mye av kompetansen til medlemmene overlappet. Det viktigste var derimot å finne ut hvilke spesialiserte ferdigheter som fantes i grupppen. Temaer som dukket opp var bildebehandlig, algoritmedesign, kontrollsystemer, multiprosessering, produktutvikling og programmering.

Basert på dette ble det laget mange forslag til oppgaver hvor en ble valgt ut. Oppgaven som ble valgt var autonom sykeseng. Denne oppgaven ga rom for å bruke mange av de spesialiserte ferdighetene til gruppemedlemmene samtidig som det var et entydig ønske fra gruppen at oppgaven skulle være av samfunnsnyttig verdi.

1.1.1 Bakgrunn

Det blir stadig tatt i bruk flere roboter i samfunnet. Dette er et resultat av teknologiske fremskritt innenfor blant annet kunstig intelligens, datasyn og robotteknikk. Samtidig er det et økende behov for å automatisere samfunnet vi lever i.

Et område hvor roboter har begynt å gjøre sitt innrykk er på St. Olavs Hospital i Trondheim. Sykehuset har tatt i bruk autonome traller for frakt av gjenstander som sengeklær, sterilt gods, kjemikalier og annet utstyr. Robotene er nødt til å omgås mennesker i korridorene på sykehuset.

To viktige faktorerer når det kommer til roboter som beveger seg i umiddelbar nærhet til mennesker er sikkerhet og effektivitet. De er nødt til å bevege seg slik at de ikke kan skade mennesker, men samtidig klare å forflytte seg fra et sted til et annet innenfor en tidsramme som er tilfredsstillende. Dette bør også skje uten at det oppstår konflikter mellom menneske og robot.

1.1.2 Oppgave

Med bakgrunn i et menneske-robot interaksjonsmiljø slik som på St. Olavs har gruppen prøvd å utvikle en løsning for å styre en robot gjennom korridorer hvor mennesker ferdes og andre hindringer kan oppstå.

Oppgaven er begrenset til det tekniske aspektet samt hvordan man skal styre roboten ved bruk av et intuitivt brukergrensesnitt. Gjennomføring og kostnader av en reell fullskala implementasjon er ikke tatt med i betraktning. Selv om problemstillingen tar for seg autonome sykesenger, kan man tenke seg at et slikt system kan anvendes til andre bruksområder hvor roboter og mennesker omgås.

Tidligere forsøk på autonom styring av roboter har i stor grad basert seg på sensorer og kameraer montert på roboten slik at dataprosessering beregnes lokalt på roboten. I dette prosjektet har vi derimot valgt å gjøre beregningene eksternt og bruke sensorer på roboten i kombinasjon med kameraer i taket.

Denne oppgaven krever kunnskap fra flere felt innenfor datateknologi slik som kunstig intelligens og algoritmekonstruksjon samt inngående kunnskap om datamaskiners kapabilitet til å gjøre multiprosessering. Kybernetikk er relevant når det kommer til kontroll og styring, og produktutvikling når det kommer til utforming av brukergrensesnitt.

1.2 Motivasjon

Ettersom det blir flere roboter i samfunnet, er det interessant å undersøke hvordan de best kan ferdes blant mennesker. Det innebærer å undersøke hvilke sensorer og metoder som er best egnet til det spesielle tilfellet, hvor det i denne omgang er snakk om innendørs i korridorer.

Spesielt på sykehus kan det ha en stor samfunnsnytte å ha autonome sykesenger slik at den menneskelige arbeidskraften kan bli brukt til andre viktige gjøremål som er nærmere tilknyttet mennesker, samt å gjøre sykehuset mer effektivt. Sykehussenger kan i tillegg være svært tunge og vanskelige å manøvrere, og det fører derfor med at det å flytte dem er en jobb knyttet til ekstra fysisk påkjenning for de ansatte.

1.3 Bidrag

Det har blitt gjort forsøk med navigering av en selvkjørende robot basert på et oppsett med kamera i taket over roboten. Det har blitt konkludert med at dette er et lite hensiktsmessig oppsett til bruk for autonome sykesenger på et sykehus. Det har blitt utviklet et enkelt brukergrensesnitt som muliggjør manøvrering og bruk av autonome sykesenger på St. Olavs Hospital uten dedikert opplæringstid av portører og sykepleiere.

1.4 Sammendrag

I kapittel 2 diskuteres det hva som ble gjort under forstudiet. Her listes opp viktige funn som ble gjort i starten av prosjektet og som har vært med på å påvirke retningen til prosjektet. Kapittel 3 presenterer en brukeranalyse som ble gjort på St. Olavs Hospital. I kapittel 4 presenteres relevant teori som er blitt benyttet for å realisere prosjektet. I kapittel 5 diskuteres et forslag til løsning på problemstillingen. Her drøftes de tekniske utfordringene. Kapittel 6 omhandler implementasjonen av løsningen. I kapittel 7 presenteres resultatet av sluttproduktet samt testing og verifisering. I kapittel 8 diskuteres det hvordan prosjektet har gått, om målene har blitt nådd, samt om det endelige produktet tilfredstiller kravspesifikasjone. Avlslutningsvis diskuteres det hvordan proskjektet kan videreføres. Kapittel 9 avrunder og konkluderer prosjektet.

2 Litteraturstudie

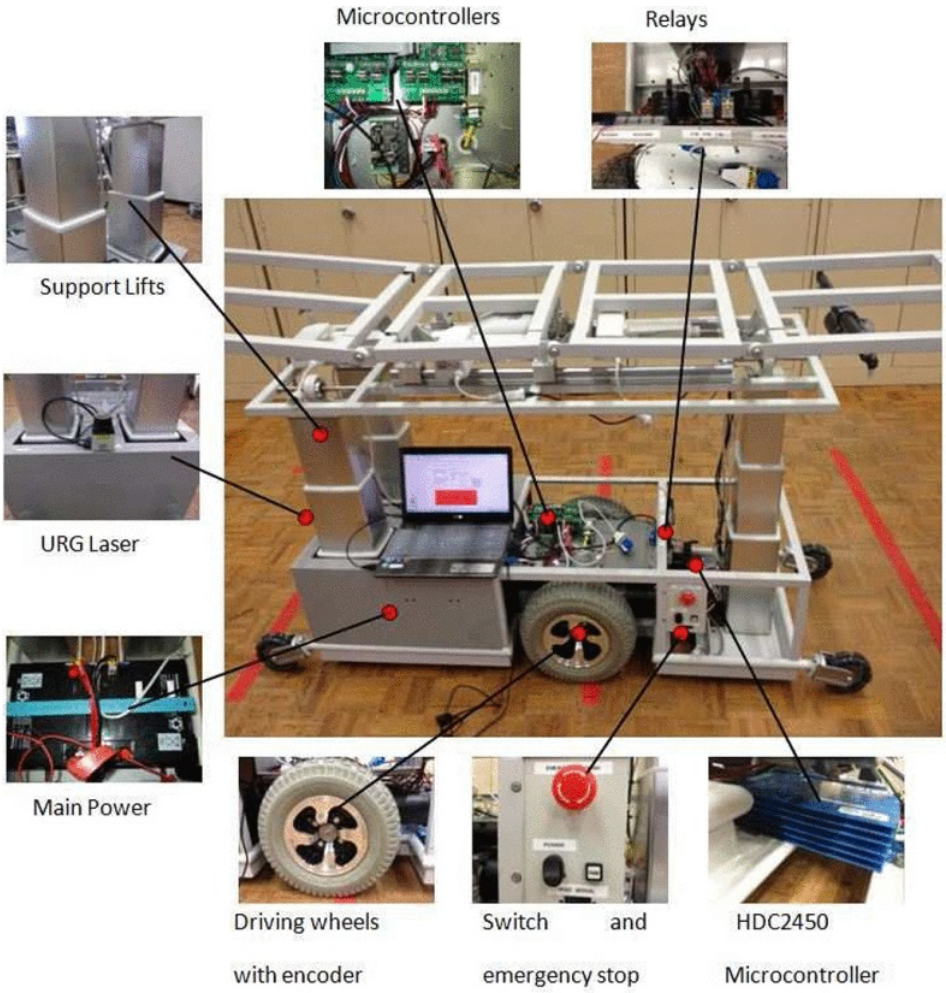
Et sykehus har mange ulike støttetjenester som sammen må fungere optimalt for å oppnå en god pasientomsorg. Disse tjenestene er ikke alltid synlige for pasienten, men er viktig for at pasienten skal få en bra, helhetlig opplevelse fra man ankommer til man sjekker ut.

Logistikk er en av disse viktige støttetjenestene, med innvirkning i alle ledd på et sykehus. Den viktigste logistikken er materialflyt, som gjerne resulterer i mer komplekse transportsystemer. I nyere tid er det flere og flere materialer som trenger å bli transportert raskere og mer effektivt, og menneskets kapasitet hindrer denne effektiviteten. Det er derfor stadig økende behov og interesse for automatisering i logistikken på et sykehus, både for å avlaste personalet, men også for å forbedre allerede eksisterende systemer. Effektiviteten av hverdagslige prosesser og kapasiteten kan bli forbedret. Personalet kan spare voldsomt mye tid, og heller spesialisere seg på mer pasientknyttede tjenester [7][8]. Dette kapittelet tar for seg noen relevante eksempler på autonome robotsystemer for sykehus.

2.1 Flexbed

Mest relevant til prosjektoppgaven er Flexbed, som i 2014 ble utviklet av et team fra UTS i Sydney. Flexbed inneholder rammeverket og madrassen til verdens første, fullt fungerende autonome sykeseng. Hovedmotivasjonen for denne prototypen er at transporteringen av pasienter er både tidskonsumerende og ineffektiv. Pasienter blir flyttet manuelt fra seng til seng, og for de kritiske nevrokirurgiske pasientene med hodeskader og lignende, så må transporteringen foregå trygt og raskt. Det trengs også opptil fire spesielt trente sykepleiere til denne typen transport. Disse sengene med kritiske pasienter må dyttes og styres til riktig destinasjon, noe som hviler fullt og helt på sykepleiernes reaksjoner og avgjørelser der og da. Konsentrasjon er kritisk for optimal betjening av sykehussengene under transport. Det er derfor vanskelig for sykepleierne å kontrollere bevegelsene til sengene samtidig som de ivaretar optimal pasientsikkerehet. Menneskelige feil er derfor en stor faktor til den potensielle og uforutsette risikoen i

transport av sykesenger. Det å flytte pasienten fra seng til seng kan videre føre til en potensiell fare for sekundærskader, som også er en bekymring rundt den nåværende transportprosessen. Å implementere intelligente sykehussenger med autonome navigasjonsegenskaper vil kunne bidra til å øke effektiviteten og sikkerheten rundt den nåværende transporteringsprosessen [11].



Figur 2.1: Flexbed

Denne sengen, som veier opptil 300kg, er ment til å kunne erstatte alle de forskjellige typene senger i ulik størrelse og funksjonalitet, som gjerne ble brukt til pasienttransport fra ambulanse til postoperativ. Den har samme funksjonalitet og bevegelighet som alle disse kombinert, og er autonom med motoriserte hjul. Personalet trenger heller ikke å løfte pasientene fra seng til seng, noe som bidrar til reduserte belastningsproblemer. Flexbed har i tillegg inkludert Trendelenburg-posisjon (flatt med beina høyere enn hodet), og reversert Trendelenburg, noe som er viktig i forhold til kirurgi. Sengen kartlegger miljøet rundt seg med lokal/live-navigasjon

basert på laser og et sporingskamera, og kan foreta kollisjonsunngåelse basert på sensordataene. Flexbed blir forhåndsprogrammert til å nå destinasjon, finner raskest mulige vei og vil kunne følge sitt tildelte personell igjenom folkemengder. Med fjernstyrt overstyring via Wi-Fi er den i tillegg lett manøvrerbar dersom det trengs. Kirurger kan løfte og senke sengen ved bruk av smarttelefon. Sengen har i tillegg innebygd HLR funksjon, som lar personalet utføre dette på pasienten dersom det er nødvendig [15].Flexbed er i dag ikke tatt i bruk, men er utviklet med tanke på videre forskning rundt dette viktige temaet.

2.2 Automatic Guided Vehicles

AGV eller Automatic/Automated Guided Vehicle er en annen form for mobil robot som bruker et navigasjonssystem til å bevege seg autonomt ved hjelp av datasyn, magneter eller laser. De opererer i en gitt infrastruktur, som er kartlagt på forhånd og installert rundt i hele anlegget der den opererer. Deretter implementeres denne infrastrukturen i roboten. Den følger ofte gitte ruter, og trenger ladestasjoner over hele anlegget slik at AGVene kan lade batteriene og avvente nye ordre etter endt oppdrag. AGVene er mye brukt til materialtransport og fungerer som assistanse til menneskelig arbeidskraft. De er også en avlastning til personalet, slik at fokus kan rettes mot andre viktige aspekter for å øke effektiviteten og kvaliteten på arbeidet. AGVer er tatt flittig i bruk på sykehus rundt omkring i verden i dag og under kommer to eksempler på mobile roboter som opererer på daglig basis i helsesektoren. [13][22]

Figur 2.2: Transcar

Transcar LTC 2 i figur 2.2 er en AGV utviklet av SWISSLOG, med en nyttelast på ca 400kg. Denne AGVen blir hovedsakelig brukt til transport av mat og forbruksvarer, og plasserer seg under varetraller (containere med utstyr), for så å løfte dem opp og transportere disse til destinasjon. De er i hovedsak kjøpt inn for å automatisere og effektivisere forsyningslinjene på sykehuset, noe som fører til en kostnads- og bemanningsreduksjon. Transcar LTC 2 opererer på egenhånd, tilkaller heiser selv, og via trådløs kommunikasjon over IP nettet kommer den seg inn og ut uten menneskelig interaksjon. Den kartlegger omgivelsene med et navigasjonssystem og vil kunne identifisere og unngå hindringer på sin kurs. Transcar LTC 2 beveger seg for det meste nært heisene og i egne underetasjer brukt til oppbevaring av utstyr for å unngå majoriteten av menneskelige konflikter. Den har blitt et kjent fjes på St. Olavs Hospital i Trondheim de siste årene. De første 7 robotene kom i 2006, og idag er det hele 21 av dem i drift i korridorene ved sykehuset [21] [20].

2.3 TUG

Figur 2.3: TUG

TUG fra Aethon, i figur 2.3, er en annen mobil robot som bruker innebygde kart og sensorer til navigasjon. Den trenger ingen spesifikk infrastruktur for å forflytte seg autonomt, noe som er en klar fordel og ulikt AGV. Den blir brukt til å "skyve" vogner og stativer, istedenfor å løfte den som Transcar LTC 2. Andre AGV-systemer må installere maskinvare til navigasjon rundt omkring på hele anlegget, samt ha mange og store ladestasjoner og dedikerte korridorer for kjøring. TUG trenger ikke det. Den bruker smart autonom navigasjon, noe som vil si at et kart over anlegget med etasjer blir målt opp med et laser-målingsredskap. Deretter blir dette kartet implementert og programmert med kjøreruter for heiser, dører, leveringssteder og ladestasjoner. TUG bruker dette kartet til veiledning, men bruker sensorer til sanntidsnavigasjon i et dynamisk miljø[6] [19].

2.4 Oppsummering

Ved å se på ulike prosjekt og prototyper som er gjort før, så vet gruppa mer om behovet rundt problemstillingen. Mobile autonome roboter er tatt godt imot i helsesektoren og er flittig brukt for å øke effektiviseringen og å avlaste sykehuspersonalet i det daglige arbeidet. Blant dem Transcar LTC 2, en AGV fra Swisslog, og TUG, en mobil robot fra Aethon.

Autonome sykesenger er per idag ikke i bruk i helsesektoren. Dette kan være primært fordi det er mange faktorer som spiller inn med tanke på menneske-robot interaksjonen i befolkede korridorer på et sykehus, men også kostnader kan være en viktig faktor. Flexbed, som ble utviklet av et team fra Sydney, er det nærmeste man kommer en fullt fungerende prototype av en autonom sykeseng idag. Denne sengen har alle funksjoner som trengs i en autonom sykeseng, og baserer seg på transport av kritiske nevrokirurgiske pasienter, mens gruppens prosjekt dette halvåret tar for seg autonom transport av tomme sykesenger. Et viktig punkt er at Flexbed er en prototype som åpner for videre forskning rundt dette temaet, som absolutt er aktuelt på et sykehus.

3 Inngående brukeranalyse

3.1 Brukerintervjuer

For å kunne definere problemstillingene med en autonom sykeseng, er to portører og to sykepleiere intervuet ved St. Olavs Hospital. Intervjuet med portør 1 ble gjennomført 25. januar for å tidlig kunne fastsette nytteverdi og potensial for prosjektet, samt uforutsette problemer. Intervjuet med sykepleierne ble gjennomført 15. mars for å få en mer detaljert oversikt over problemer etter at prosjektet var påbegynt. I utgangspunktet var gruppa skeptiske til en autonom sykeseng for bruk med pasienter i, denne skepsisen viste seg under intervjuene å være godt begrunnet. Derfor vil dette kapitlet først og fremst handle om autonome sykesenger som transporteres tomme. Intervjuene har handlet om de intervjuedes meninger og om hvordan sykesengene brukes i praksis på St. Olavs Hospital i dag. Også de eksisterende transportrobotene på St. Olavs som kalles AGV-er (Automated Guided Vehicle) ble undersøkt.

3.1.1 Brukerne

Sengene og bårene på sykehuset flyttes i utgangspunktet mest av portørene. Portørene har også flere oppgaver, de fungerer blant annet som "ambulansepersonale"inne på sykehuset som frakter akuttpasienter, og de frakter andre ting som ikke kan fraktes av sykehusrobotene, blant annet gassflasker. De frakter også familier fra fødeavdelingen til barselhotellet.

Det er i utgangspunktet portører som flytter på sengene og bårene mellom avdelinger, men det hender at sykepleiere gjør det hvis det ikke er nok portører tilgjengelig der og da. Sykepleierne flytter ofte senger inne på avdelinger, så da først og fremst korte avstander, mens portør 1 anslo at han flytter senger rundt 2 mil i løpet av en arbeidsdag.

3.1.2 Pasienter

Pasienter som flyttes i senger flyttes i dag først og fremst av portører, men også av og til av sykepleiere. Portør 1 mente at flere pasienter setter pris på å ha en de kan prate litt med når de flyttes rundt i seng, og portøren så på seg selv både som en transportarbeider og en samtalepartner for pasientene. I tillegg er det mange pasienter som forflyttes som er i en veldig sårbar situasjon, og som kan oppleve å få en akutt tilstand under forflytning. På bakgrunn av disse sammenhengene mente portørene at autonome sykesenger for forflytning av pasienter var en dårlig idé. På bakgrunn av dette ble ingen pasienter intervjuet, men også fordi ordentlig gjennomførte intervjuer av pasienter ville kreve uforholdsmessig mye arbeid bl. a. tilknyttet personvern og taushetsplikt.

3.1.3 Eksisterende senger

Alle sykesengene på sykehuset har hjul. Det er ca. like mange av en gammel type og en nyere type senger. Den gamle typen har fire hjul, to av dem kan låses i posisjon med en fothendel og blir slik styrehjul. Alle justeringsmuligheter på sengene er manuelle, f.eks. heving og senking. De nye sengene har fem hjul. Fire av dem er alltid bevegelige, det siste av dem er under midten av senga og kan låses. Justeringsmulighetene er elektriske på de nye sengene, og de nye sengene oppleves på alle måter som enklere i bruk av portører og sykepleiere.

Sengene flyttes fra ett sentrallager ut til de ulike avdelingene etterhvert som det er behov for dem, bestillingen av senger gjøres av sykepleierne på de respektive avdelingene gjennom et datasystem. Når en avdeling får behov for en seng, sendes en bestilling, bestillingen mottas av portøravdelingen, og de sender en portør for å flytte sengen fra lageret til avdelingen. Når avdelingen ikke har behov for sengen lenger, sender de en ny bestilling, og en portør kommer og henter sengen. Ifølge portør 1 som ble intervjuet står denne transporten av tomme senger for ca. 25% av sengetransporten portørene utfører. Frakt av tomme senger fra avdelingene til lageret haster stort sett ikke, frakt av tomme senger fra lager til avdeling haster av og til.

I tillegg til de vanlige sykesengene har også portørene ansvar for bårene på sykehuset. Disse ser ut som vanlige senger, men er noe mindre, og noe vanskeligere å håndtere. Disse fraktes fra akuttmottaket til avdelinger med pasienter i, og fraktes tomme tilbake. Transport fra akuttmottak til avdelinger haster ofte, transport tilbake haster av og til. Bårene fraktes også av og til av sykepleiere. Bårene representerer omtrent like mye arbeid for portørene som sykesengene.

3.1.4 Autonome transporttraller

Sykehuset har i dag 21 AGV-er. Disse frakter idag kun gjenstander som ikke haster, og ikke mennesker. Robotene kjører først og fremst rundt nede i en egen etasje i kjelleren og tar heisene til og fra de ulike avdelingene. De kjører i utgangspunktet ikke inne på avdelingene, men setter fra seg-, og henter varer rett ved heisene. Når det står gjenstander eller mennesker i veien for roboten, sier den ifra med en stemme slik at den skal få hjelp [21].

3.2 Relevante problemer fra intervjuene

3.2.1 Manuell kontroll

Et viktig problem som ble oppdaget med AGV-ene gjennom intervjuene er at de ikke kan overstyres manuelt på stedet. Dersom en robot stopper foran et hinder eller vil inn i en heis der det allerede står mennesker eller en annen robot, står trallen fast helt til det kommer en ansatt fra den sentrale logistikkavdelingen og manuelt overstyrer roboten. Dette oppleves som et stort problem for portørene, og et noe mindre problem for sykepleierne. Av denne grunn må en autonom sykeseng kunne overstyres manuelt av en hvilken som helst ansatt for å ikke være i veien.

3.2.2 Heiser

Et annet viktig problem er at robotene bruker veldig lang tid på å ta en heis. Robotene kjører såpass sakte når de skal ta en heis og opptar hele heisen, slik at portørene ofte ender med å måtte finne en annen heis dersom heisen de helst vil ta blir opptatt av en robot. Dette går ikke an å overstyre av portørene. Uansett om en portør kommer med en hastepasient og roboten kjører skittentøy, eller hvis heisen er åpen, men reservert for en robot, må portøren ta en annen heis. Dette er et problem som vil være akkurat like relevant for autonome sykesenger.

3.2.3 Brukervennlighet

Portører får i dag ordentlig opplæring av andre portører på hvordan de best kan håndtere sengene og bårene. Sykepleierne får ikke nødvendigvis det, og det har skjedd flere ganger at en portør har sett en eller flere sykepleiere slite med en seng og gitt et lynkurs. En seng med en tung pasient som en portør fint klarer å flytte alene, kan i dag kreve to sykepleiere, rett og

slett fordi ikke alle sykepleierne vet hvordan sengene brukes mest effektivt. Sengene er altså i dag laget for kompliserte for brukerne gitt at ikke alle brukerne får skikkelig opplæring. Dette er overførbart til en autonom sykeseng fordi den må være brukervennlig nok til at alle ansatte burde kunne bruke sengene uten opplæring.

3.2.4 Plass

Robotene tar mye plass idag fordi de kjører midt i korridorene, derfor kan man ikke komme seg forbi med større ting, f.eks. senger. Dette vil gjelde i enda større grad for en seng ettersom sengene er større. En autonom sykeseng burde altså helst kjøre på en side av korridoren. En tilleggsenhet som kan monteres på en vanlig sykeseng for å gjøre den selvkjørende vil nødvendigvis måtte utformes på en måte som ikke gjør at utstrekningen til sengen øker. Bredden på korridorene er nøye tilpasset at to senger akkurat kan passere hverandre, og lengden på heisene er også akkurat slik at en seng får plass med lengden de har i dag.

4 Teori

4.1 Datasyn

Datasyn er en viktig modul når du skal styre noe via bilder eller video. Under kommer en forklaring på metodene innenfor datasyn og bildeprosessering som ble brukt i dette prosjektet for å segmentere og hente ut posisjoner til objekter i bilder for å fungere som synet til roboten.

4.1.1 Fargekanaler

RGB (Rød, Grønn, Blå) er formatet bildene har når de kommer inn til datasyn-modulen av programmet. Formatet består av tre fargekanaler med gråtoneverdier, med en for hver av de tre fargene, som så blir oversatt til fargene vi ser på dataskjermen [18]. Derfor oversettes denne fargemodellen til formatet HSI/HSV (Hue, saturation, intensity/value) [18]. Hue gir kun den enkelte fargen (kun gul, kun hvit, kun blå osv.). Saturation angir i hvor stor grad pikslene har denne fargen, og intensity er en noe subjektiv kanal, og kan varieres etter bruk. Her er det blitt brukt en variant som bare tar gjennomsnittet av de originale RGB verdiene. Oversettingen er gjort på følgende måte [18]:

$$H = \begin{cases} \theta & \text{if } B \leq G \\ 360 - \theta & \text{if } B > G \end{cases} \tag{4.1}$$

$$\theta = cos^{-1} \left\{ \frac{\frac{1}{2}[(R-G)+(R-B)]}{[(R-G)^2+(R-B)(G-B)]^{\frac{1}{2}}} \right\} \tag{4.2}$$

$$S = 1 - \frac{3}{(R+G+B)}[min(R,G,B)] \tag{4.3}$$

$$I = \frac{1}{3}(R+G+B) \tag{4.4}$$

Som tilsammen blir til HSI/HSV.

4.1.2 "Canny edge detector"

For å segmentere et bilde er det ofte nødvendig å finne kanter til objektene i bildet. En kant kan være et område med piksler som har stor forskjell i intensitet sammenlignet med pikslene som er rett ved siden av. En algoritmene for å finne kanter i et bilde er "Canny edge detector" [10]. Algoritmen baserer seg på tre grunnleggende mål [18]:

1. Lav feilmargin. Den skal finne alle kantene og kantene skal være så nærme de korrekte kantene som mulig.
2. Punktet som regnes som en kant av kantgjenkjenneren skal være så nærme som mulig den faktiske kanten i bildet, så distansen mellom dem skal minimeres.
3. Kantgjenkjenneren burde bare returnere et enkelt punkt for det reelle punktet. Dette betyr at den ikke skal gi flere punkter tilbake der det bare skal være en enkelt kant.

I grove trekk er framgangsmåten i algoritmen som følgende:

- Glatt ut bildet med et Gauss-filter.
- Beregne "gradient magnitude" og "angle images".
- Utfør ikke-maksimum utelatelse på "gradient magnitude" bildet.
- Bruk en dobbel terskel og tilkoblingsanalyse for å gjenkjenne og lenke kanter.

For en mer detaljert forklaring av "Canny edge detector", se [18] sider 719-725 eller [10].

4.1.3 Binær morfologi

Binær morfologi innebærer å ta i bruk de matematiske morfologioperatorene dilation og erosion [18]. De bruker begge to en kernel (matrise) i et eller annet mønster, som har et midtpunkt, det vil si dimensjoner som er oddetall. Denne scanner da over bildet, med midtpunkt som hver av pikslene i det originale binærbildet, og gjør elementvis matrisemultiplikasjon med kernelen og området rundt pikselen. Ved dilation vil den velge maks fra resultatet, altså at dersom det etter multiplikasjon er en 1 verdi vil dette bli verdien til pixelen i samme posisjon som den som blir vurdert i det resulterende bildet [18]. For erosion er det motsatt, den ser etter den laveste verdien i resultatet og velger denne som den nye verdien [18]. Disse to metodene blir brukt til det som kalles for "opening" av et binærbilde, hvor du utfører dilation på original bildet, etterfulgt av erosion på bildet som blir gitt ut fra dilation [18]. Mer om morfologi kan lese om i [18] i kapittel 9.

4.1.4 "Flood filling"

"Flood filling" er en måte å fylle inn farge i objekter som er helt lukket av kantene sine. Den starter i et punkt og så vokser utover til den møter en kant i alle retninger. [17]

4.1.5 "Matching template"

"Matching template" er en måte å finne et objekt i bildet. Den scanner over bildet med en gitt stride og ser hvor mye den ligner på den malen som du gir den. Den returnerer tilslutt koordinatene på den plassen i bildet som passer best mot malen, og regner dette som objektet [18]. Formelen som blir brukt til å regne ut hvor nærme malen er til det faktiske bildet er [4]:

$$R(x,y) = \Sigma_{x',y'}(T'(x',y') - I'(x+x',y+y'))^2 \tag{4.5}$$

hvor

$$T'(x',y') = T(x',y') - 1/(w*h) * \Sigma_{x'',y''}T(x'',y'') \tag{4.6}$$

$$I'(x+x',y+y') = I(x+x',y+y') - 1/(w*h) * \Sigma_{x'',y''}I(x+x'',y+y'') \tag{4.7}$$

4.2 A*-søkealgoritme

En av de mest kjente beste-først søkealgoritmene er A*, som kan brukes på både grafer og på matriser hvor hver indeks representerer en unik node. Den kombinerer kostnaden $g(n)$ for å forflytte seg til en node n og en heuristikk $h(n)$ for å gjøre et informert anslag for avstanden til målet for å estimere billigste løsning igjennom node n.

$$f(n) = g(n) + h(n) = \text{estimert billigste løsning gjennom node n}$$

Kostnaden $g(n)$ er forbundet med kostnaden til noden som er bestemt av problemet. Heuristikken $h(n)$ velges ut i fra det spesifikke problemdomenet. Et typisk valg for $h(n)$ for noder som kan beskrives med posisjoner i et koordinatsystem er den euklidske distansen mellom n og målnoden s_g.

$$h_e(n) = \sqrt{(s_g.x - n.x)^2 + (s.g.y - n.y)^2}$$

eller den såkalte Manhatten-distansen

$$h_m(n) = |s_g.x - n.x| + |s.g.y - n.y|$$

Vist under er pseudokode for den generiske algoritmen. Gitt en startnode s_0 og målet s_g finner algoritmen billigste løsning gitt som en sekvens av tilstander fra start til slutt. Etterhvert som den genererer nye noder (naboer av forrige node) blir disse enten plassert i den åpne eller lukkede lista som viser at naboene ikke har blitt besøkt før eller allerede har blitt besøkt. Lista med noder som ikke er besøkt før (OPEN) blir også sortert etter stigende f-verdier slik at algoritmen blir "dratt" mot noder som den tror raskest vill lede til målet.

Funksjonen **attach-and-eval(s,p)** tilordner ganske enkelt noden s det som den så langt tror er dens beste (billigste) løsning og oppdaterer nodens f-verdi.

Funksjonen **propagate-path-improvements(s)** går rekursivt igjennom forfølgeren til s når det har blitt funnet en billigere vei til s og oppdaterer deres beste forfølger. Dette er nødvendig fordi at når en bedre sti blir funnet kan dette påvirke tidligere oppdagede noder sin f-verdi.

Generisk A*(startnode n_0, sluttnode n_g): returnerer billigste løsning

```
CLOSED ← ∅
OPEN ← ∅
g(n_0) ← 0
f(n_0) ← g(n_0) + h(n_0)
Legg n_0 i OPEN
while OPEN ≠ ∅ do
  X = pop(OPEN)
  push(X, CLOSED)
  if X er målet then
    return  Løsningssekvens += rekursivt kall på X.parent
  end if
  SUCC ← generate-all-successors(X)
  for all S ∈ SUCC do
    Hvis node S* har vært undersøkt tidligere, og hvis state(S*) = state(S), S ← S*
    push(S,kids(X))
    if S ∉ OPEN og S ∉ CLOSED then
      attach-and-eval(S,X)
      insert(S,OPEN), sorter OPEN etter stigende f-verdier
    else if g(X) + arc-cost(X,S) < g(S) (har funnet en raskere billigere vei til S) then
      attach-and-eval(S,X)
      Hvis S ∈ CLOSED, propagate-path-improvements(S)
    end if
  end for
end while
```

Algoritmen er vist å finne optimal/beste løsning så lenge heuristikken ikke underestimerer avstanden til målet. Mer om algoritmen er å finne i [12].

4.3 Kontroller

En kontroller er en algoritme som styrer et pådrag med den hensikt å få en variabel, y i et system til å oppnå og holde en ønsket verdi, en referanse y_0, basert på en tilbakekobling med faktisk verdi. Bruksområder kan innebære alle mulige elektriske og mekaniske systemer. Et

eksempel er en elektrisk vannpumpe hvor man ønsker å regulerer strømtilførselen til motoren for å oppnå en ønsket vannstrøm i pumpen. Kontrollere finnes i mange varianter og kan tilpasses systemet de skal benyttes i. Man kan blant annet benytte seg av en generell type og tilpasse variablene i kontrolleren eksperimentelt. En av de vanligste typen kontroller er en Proporsjonal-Integral-Derivat-kontroller. Algoritmen for en slik kontroller beskrives med følgende likning:

$$u(t) = K_p e + K_i \int_0^t e dt + K_d \frac{de}{dt} \tag{4.8}$$

hvor e er definert som avviket mellom ønsket verdi og reell verdi, $e = y_0 - y$ og u er pådraget. K_p, K_i og K_d er konstanter og er parameterne for kontrolleren.

4.3.1 Proporsjonalvirkning

Proporsjonalleddet $K_p e$ vil resultere i et pådrag som er proporsjonalt med feilen. Dersom K_p er lav, vil dette gi et sakte, men riktig pådrag dersom det ikke er noen konstant forstyrrelse i systemet. Et eksempel kan være at man ønsker å fylle en vanntank til et bestemt nivå. Dersom man bruker en proporsjonalkontroller til å styre en vannpumpe som fyller vann inn eller ut av tanken, vil denne klare å fylle riktig mengde vann dersom det ikke er hull i tanken eller andre kilder eller sluk. Dersom K_p er høy, vil kontrolleren skape et oversving, evt. oscillasjoner rundt referansen. Hvis det dannes et stasjonært avvik i systemet, fordi man har forstyrrelser til stede, vil en P-kontroller ikke fungere. Dette vil tilsvare at det er et hull i tanken i det ovennevnte eksempelet. Man vil klare å minske feilen, men det vil bli en stasjonær feil.

4.3.2 Integralvirkning

Integralleddet sørger for at kontrolleren tar hensyn til feil tilbake i tid. Integralleddet $K_i \int_0^t e dt$ vil integrere opp, eller i diskrete tilfeller summere alle tidligere feil, og ta høyde for dem. Hensikten med leddet er å motvirke stasjonære avvik. Kontrolleren vil altså ta hensyn til at den ikke klarer å motvirke forstyrrelsene i begynnelsen, og korrigere pådraget sitt for hver gang den registrerer at det forrige pådraget ikke var tilstrekkelig. I eksempelet med vannpumpen vil det gjøre at kontrolleren sørger for at vannpumpen øker inn-strømmen når den registrerer at det ikke holder med den mengden den pumper for å holde det ønskede vannnivået. Ulempen med integralvirkning er at den også vil kunne føre til oversving.

4.3.3 Derivatvirkning

Derivatleddet $K_d \frac{de}{dt}$ sørger for at kontrolleren tar hensyn til hvor raskt feilen endrer seg, og tar dermed høyde for framtidige feil. Leddet vil derfor kunne redusere oversving uten å påvirke et stasjonært avvik. En høy K_d vil skape et lavt oversving og motsatt. Ulempen med å ha med et derivatledd er at leddet vil kunne påvirkes veldig av signalstøy, sensorfeil og små, raske avvik. Dersom den variabelen man ønsker å holde konstant i virkeligheten er konstant mens målesignalet for variabelen har feil som er større enn disse endringene, kan leddet føre til raske oscillasjoner eller output-signal som er større en pådragsorganet har kapasitet til. Dette vil føre til ustabilitet. Problemet kan unngås med å filtrere ut høyfrekvente signaler fra input-signalet med et lavpassfilter.

4.4 Produktutvikling

4.4.1 Påvirkingsmulighet

I en typisk utviklingsprosess har man store endringsmuligheter for små kostnader tidlig i prosessen, og ofte mindre og mindre endringsmuligheter for større og større kostnader lengre ut i prosjektet. På det stadiet man er når man har en helt funksjonell prototype er det typisk svært vanskelig å gjøre store endringer på produktet. Ved å gjennomføre intervjuer og brukerobservasjoner kan man derfor spare seg for mye problemer sent i utviklingsprosessen. Mange av problemene kan identifiseres før man i det hele tatt har startet med å designe produktet, og kan slik unngå mange fallgruver [16].

4.4.2 Intervjuer

Når man skal utvikle er produkt for en bruker er det viktig å identifisere brukerens ønsker og behover, samt produktets ulike brukssituasjoner. Kundens og brukerens behover og ønsker kan ofte være svært ulike. Det er helt avgjørende for utviklerne å intervjue de ulike potensielle brukerne av produktet. Ettersom man som utvikler i utgangspunktet skal lage noe som ikke finnes fra før, kan det være utfordrende å identifisere potensielle problemer ved produktet og bruken av det før man har en funksjonell prototype. Derfor er det viktig å vinkle spørsmålene under intervjuene på en slik måte at det man finner ut vil være overførbart til det produktet man skal faktisk skal lage, og ikke bare lignende produkter. Når man skal intervjue en bruker om brukssitasjoner eller produkter er en vanlig metode å først innlede med et relativt åpent

spørsmål, og deretter følge opp hvert svar med "Hvorfor det?", 5 ganger rett etter hverandre. På denne måten kommer man helt til kjernen av brukerens opplevelse, og lærer om brukerens følelser og holdninger. Grunnen til at dette er viktig er at man da vil kunne identifisere mange problemer som brukeren selv ikke er klar over eller som de ikke kommer på under selve intervjuet, selv om problemene kan være betydelige. Dette kan være en noe ubehagelig prosedyre, spesielt for nordmenn som kanskje er litt mer private enn de er i mange kulturer, ettersom man til slutt gjerne ender opp med opplysninger som er mer personlighetstrekk enn meninger om ett spesifikt produkt. Den er ikke desto mindre et utrolig nyttig verktøy for raskt å identifisere utfordringer[16].

Gjennom brukerintervjuer kan man videre identifisere hva slags situasjoner det produktet man utvikler vil kunne havne i. Dette gjelder både de situasjonene produktet er ment å brukes i, og alle situasjoner det kan havne i som man som produktutvikler har vansker for å forutse uten å kjenne omgivelsene til produktet fullstendig. De situasjonene man som ingeniør ikke ser for seg med det første under utviklingen, er veldig viktige å identifisere ettersom de kan være akkurat like avgjørende for produktets suksess som de sitasjonene man selv kommer på[16].

4.4.3 Prototyping

Når man har en formening om hvordan produktet skal se ut og fungere på bakgrunn av intervjuer og brukerobservasjoner, kan man gå videre til å begynne med å utvikle prototyper. Tradisjonelt har prototyper vært laget med håndfaste og solide modeller når det gjelder fysiske produkter, og utkast til kode i programvaretilfeller. Disse tar gjerne mye tid og arbeid å lage, og er krevende å endre. Derfor er det ofte hensiktsmessig å lage de første prototypene med helt enkle, fysiske materialer, eksempelvis papp og teip. Det går ekstremt fort å lage slike prototyper, og de ser sjelden spesielt representative ut. Det holder imidlertid at prototypene ligner nok på et sluttprodukt til at brukeren forstår idéene. Eksempelvis kan man lære mye om hvordan en bruker ønsker at et brukergrensesnitt til en programvare skal se ut og fungere, ved å lage en papirversjon av grensesnittet. Man kan så flytte manuelt på figurene når brukeren later som om han/hun gjør forskjellige handlinger. Ettersom slike prototyper både er raske og billige å lage, kan man lage svært mange av dem for å videreutvikle idéer. Man kan finne ut ekstremt mye på veldig kort tid på denne måten, og det lar utviklerne identifisere alle de viktigste egenskapene som må med i det ferdige produktet lenge før de har begynt å realisere en fullt funksjonell prototype. Slik kan man bruke de senere og dyrere prototypene til å kun luke ut småfeil [16].

5 Forslag til løsning

En autonom sykeseng kan implementeres på forskjellige måter. Det er mange hensyn som må taes og det er flere måter å løse problemet på. Det er blitt undersøkt hvilket utstyr som er tilgjengelig og hva som er gjennomførbart i tidsrommet til denne oppgaven. For å få mest ut av tverrfagligheten i gruppen har også gruppemedlemmenes faglige bakgrunn vært utslagsgivende i valg av løsning.

5.1 Idé og oppdeling av oppgaven

Gruppen har valgt å gå for en kamerabasert løsning hvor et kamera festes i taket i korridoren hvor sengen skal kjøre. Fra bildestrømmen hentes det ut data om sengens posisjon, samt vegger og andre hindringer i korridoren. Basert på denne informasjonen kan en optimal kjørerute for sengen regnes ut. Dette gjøres ved bruk av en korteste-vei-algoritme. Når en bane er funnet, sendes veipunktene til et kontrollsystem bestående av to PID-kontrollere som styrer orienteringen og posisjonen til roboten.

For å lage en mest mulig robust løsning, er det blitt lagt vekt på å lage en modulbasert løsning. Modulene er delt opp slik at hver modul omhandler et tema og har klare grensesnitt for input og output. Robotsystemet består av fire moduler som ble utarbeidet hver for seg, for så å bli koblet sammen til det ferdige systemet. I starten av prosjektet ble det utformet klare rammer for hva hver modul skal løse, samt hvordan de skal interagere med hverandre. Modulene som er definert er kamera, bildebehandling, banegenerering og kontrollsystem. Disse blir alle styrt fra ett hovedprogram som koordinerer dataflyten mellom modulene. Alle behandles på en datamaskin før instruksjoner sendes til selve roboten, slik at roboten ikke gjør noe av dataprosesseringen selv.

I de neste delkapitlene beskrives modulene og sensorene som samler inn data fra omverden. Til slutt følger en beskrivelse av forenklinger som er gjort for å gjennomføre prosjektet.

5.2 Modell av sykeseng

For å simulere en autonom sykeseng er det utarbeidet en prototype i miniatyrstørrelse. Med hensyn til de tilgjengelige ressursene falt valget på en Lego Mindstorm-robot. Den valgte modellen har en 32-bit ARM7TDMI-core Atmel AT91SAM7S256 mikrokontroller med 256 KB FLASH minne og 64 KB RAM. I tillegg har den en 8-bit Atmel AVR ATmega48 mikrokontroller. Roboten har støtte for USB og Bluetooth [1]. Datamaskinen kalles en "Intelligent Brick" og styrer to servomotorer. Roboten tar inn data fra et kompass, men har utover dette kontakter for en ekstra servomotor og tre ekstra sensorer. Designet for "Brick"-en er offentliggjort og det er på bakgrunn av dette laget et fritt Python-bibliotek kalt NXT-Python av en tredjepart [14]. Det er dette biblioteket roboten benytter seg av.

5.3 Sensorer

Det ble gjort en vurdering av hvilke sensorer som skal benyttes. Lego Mindstorms har støtte for en rekke sensorer som ultrasonisk avstandsmåler og kompass. For å bestemme den faktiske orienteringen til roboten falt valget på kompasset. Den ultrasoniske sensoren ble brukt som avstandsensor for testing av kontrollsystemet, men ble ikke en del av den ferdige løsningen. Avstand beregnes utifra posisjonen til roboten i bildebehandlingen.

GPS ble også sett på som en mulighet for å finne posisjonen til roboten. Dette alternativet utgikk ettersom sengen er ment til innendørs bruk, hvor GPS-signalene kan være svake og for unøyaktige til fin-navigasjon. Siden Lego-roboten er en miniatyr av en virkelig full-skala implementasjon ville ikke GPS latt seg bruke i prototypen heller.

5.4 Brukergrensesnitt

For at en elektrisk sykeseng skal kunne kjøres manuelt av portører og sykepleiere, kreves det et brukergrensesnitt som er lett å forstå og lett å bruke. Det er ønskelig at opplæring skal være unødvendig da portører og sykepleiere allerede har en hektisk arbeidsdag. I tillegg burde grensesnittet være behagelig å bruke slik at portørene som flytter sengene over lengre distanser, ikke risikerer å pådra seg belastningsskader over tid. Alt burde være tydelig, oversiktlig og lett å se, slik at også folk i alle aldre som aldri har brukt sengene skal kunne se og forstå hvordan de bruker dem.

En autonom sykeseng vil ha behov for en form for adgangsbegrensning. Dette er fordi man vil unngå at pasienter prøver å kjøre rundt med sin egen seng, eller at noen kommer borti styringskontrollene ved et uhell.

For at sengen skal kunne gå på batteridrift, vil den ha behov for å lades med jevne mellomrom. Det er ønskelig at sengene selv skal ha oversikt over ladestasjonene, slik at de vet hvor nærmeste lader er og kan kjøre dit på egenhånd. For at en bruker skal kunne la sengen kjøre til en ladestasjon på egenhånd, er det nødvendig med en indikator som viser ladestatus til brukeren.

Sikkerhet er helt fundamentalt for alle former for kjøretøy med egen drift, som skal interagere med mennesker. For at en skal være sikre på at roboten kan stoppes hvis noe skulle gå galt, er det nødvendig med en eller flere synlige nødstopp-knapper, og lett tilgjengelige steder på sengen.

5.5 Begrensninger

Ettersom prosjektet har hatt en relativt kort tidsramme med begrensede ressurser, ble det definert en rekke tekniske forenklinger slik at prosjektet skulle være gjennomførbart. De neste delkapitlene beskriver de forenklingene som er gjort med tanke på roboten og dens omgivelser.

5.5.1 Miniatyrmodell

Prosjektet har hverken hatt tid eller ressurser til å lage en fullskala autonom sykeseng. Det ble derfor bestemt at det skulle lages en miniatyrmodell ved bruk av Lego Mindstorms. Roboten har samme bevegelighet som en sykeseng, det vil si at den skal kunne snu rundt sin egen akse.

5.5.2 Datasyn

Vårt system er begrenset til ett kamera, festet direkte over roboten og korridoren den skal kjøre gjennom. Dette gjør avstandsberegninger lettere. I et reelt scenario på et sykehus vil det være behov for flere kamera for å dekke alle områder. Det kan også være behov for kameraer eller avstandssensorer festet på sengen. Ved å forholde seg til ett kamera trenger man ikke å ta hensyn til bilder fra flere kameraer. Det er en forenkling at kameraet roboten bruker ikke sitter på selve roboten. Det vil være problematisk både personvernmessig og rent teknisk å montere- og bruke data fra kameraer i taket over hele sykehuset.

5.5.3 Omgivelsene

Omgivelsene er forenklet til ensfargede vegger og hindringer formet som rektangler av forskjellige størrelser. Veggene er markert med høykontrast-teip for å kunne oppdages lettere med kamera. Et kryss med høykontrast-teip er festet på toppen av roboten for å kunne gjenkjenne den på bilder. Hindringer er forenklet til ensfargede rektangulære pappesker med høykontrast-teip.

5.5.4 Kontrollsystem

Kontrollsystemet er begrenset til to PID-kontrollere, en for avstandskontroll og en for vinkelkontroll. For enkelhetsskyld er kontrollsystemet lagt opp slik at det ikke vil være mulig å svinge og kjøre framover eller bakover samtidig. Roboten må altså stoppe å kjøre framover før den kan svinge, og motsatt.

5.5.5 Brukergrensesnitt og brukere

Det er først og fremst fokusert på utforming og utseende av grensesnittet og hva slags funksjonalitet det skal ha. Fysiske knapper og tilkobling av disse til selve roboten er utelatt. Det er valgt å kun intervjue brukerne, og ikke prøve ut brukssituasjonene selv. Brukerne er heller ikke observert mens de jobber.

6 Implementasjon av valgt løsning

Dette kapittelet er dedikert til en inngående beskrivelse av hvordan hver modul er implemetert, samt utfordringer knyttet til arbeidet og dynamikken mellom modulene. Programmet er implementert i Python 2.7.

6.1 Kamera

Kameraet som ble brukt var et trådløst internettkamera fra D-Link med modellnummer `DCS-2121` kjørende på Firmware `1.06.7712`. Kameraet ble montert i taket for å få størst mulig oversikt over banens tilstand og for å gjøre det enklere å identifisere roboten på bildet.

6.2 Datasyn

For å kunne behandle bildene raskt nok til å kunne styre roboten, brukes OpenCV [5] til objektgjenkjennelse. OpenCV er en åpen kildekode for bildegjenkjennelse i sanntid som er skrevet i C. For at algoritmen skal klare å gjenkjenne vegger, ble det valgt å bruke grønn høykontrast-teip som festes på gulvet for å illustrere vegger. I tillegg til OpenCV brukes også Scipys objektgjennkjennings-modul [3], kalt ndimage for metoden labeling. Objektgjennkjenningen er delt opp i syv steg, presentert i kronologisk rekkefølge under.

6.2.1 Fargekonvertering

Det første steget er å konvertere bildet fra den vanlige RGB-modellen, til HSV. Dette blir gjort for å kunne bruke Value biten av HSV til objektgjenkjenningen, da denne ofte gir veldig gode kontraster å jobbe med når du helst bare vil bruke en enkelt kanal. Fordelen med dette er da at man får et nøyaktig bilde med store kontraster som er godt å jobbe på, og ikke trenger å bruke tre ganger så mye ressurser på å kjøre hele objektgjenkjenningen på alle tre fargekanalene. Fordelen over et vanlig svart og hvitt bilde oversatt fra et farge-bilde er at det

ofte tar bedre vare på kontrastene. Dette er veldig viktig for demobanen som roboten er blitt testet i. For å gjøre dette ble OpenCVs innebygde metode "cvtColor" brukt for å konvertere fra RGB modellen til HSV.

6.2.2 Kantgjennkjenning

Etter at HSV-versjonen av bildet er funnet brukes Value biten videre til å finne kantene i bildet. Dette gjøres ved bruk av "Canny edge detection". Denne ble valgt for at den er rask og en av de beste algoritmene til å finne kanter i et bilde. Den innebygde funksjonen i OpenCV ble brukt for å utføre denne algoritmen. Algoritmen finner kantene for lettere å kunne fylle inn de korrekte plassene til objektene for segmenteringen.

6.2.3 Morfologi

Blir brukt til å binde sammen kanter, så objekter der kantgjennkjenningen ikke finner hele kanten rundt et objekt kan bli bundet helt sammen for fyllingen som kommer senere. I algoritmen som er brukt ble det benyttet to iterasjoner av dilation med en 3x3 kernel med kun enere for å la kantene vokse sammen, etterfulgt av en iterasjon med erode for å fjerne unødvendige piksler/forbindelser som ble laget av dilation, denne bruker samme kernel.

6.2.4 Objektfylling

Nå skal alle kantene til objekter være koblet sammen, så da kan objektene fylles. Dette gjøres ved hjelp av "flood fill" algoritmen, med OpenCVs innebygde versjon av denne algoritmen. Dette ble gjort for å fylle inn alle kantene i bildet, slik at det blir lettere å finne senterposisjonen til alle objekter, og for å få satt opp banen til den skal inn i banegenererings algoritmen.

6.2.5 Objektmerking

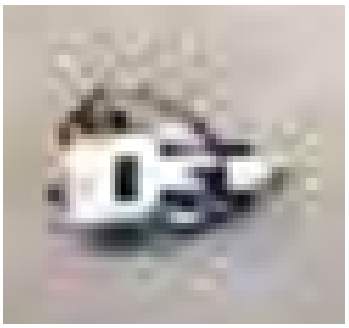
Etter alt er fylt inn får objektene hver sin merking, som er et tall fra 1 til antall objekter i bildet, gjennom en prosess som heter labeling. Til dette ble det brukt ndimages innebygde funksjon "label". Det blir da lett å se hvor store elementene i bildet er, og de små elementene som mest sansynelig er støy i bildet kan fjernes for at det skal bli "renere".

6.2.6 Finn objektsenter

Objektsenter finnes ved å gå gjennom alle objektene som nå er i bildet, og finner alle koordinatene som hører til dette elementet. Deretter tas median for hvert av elementene for å finne senteret.

6.2.7 Finn robotens posisjon

Til dette brukes algoritmen "Match template", som scanner bildet for en gitt mal den skal se etter. Posisjonen den finner ut at det er størst sjanse for at denne malen ligger blir returnert som robotens posisjon. Malen den bruker for å se etter roboten i bildet kan bli sett i Figur 6.1.



Figur 6.1: Mal av roboten for "Match Template"

Her brukes også OpenCVs implementasjon av algoritmen, som ligger i funksjonen "matchTemplate". Etter alle disse stegene returneres robotens posisjon, det segmenterte bildet, listen med objektsentere og det markerte bildet, som så sendes til videre bruk i banegenerering og til å styre roboten.

6.3 Banegenerering

Hvordan den generiske A* algoritmen fungerer er forklart i kapittel 4.2, her utdypes hvordan algoritmen er spesialisert til å løse det spesifikke problemet.

6.3.1 Tilstandsrommet

For å finne en vei igjennom løypa med hindringer er det benyttet en A*-algoritme som bruker bilder fra kamera som numpy arrays (matrise-form). Piksler med verdi 0/False blir vurdert

som et punkt uten hindringer, og piksler med verdi 1/True blir vurdert som hindringer. Kostnaden for å passere en piksel/node med verdi 0 er 1, mens noder med verdi 1 har uendelig stor kostnad og vil ikke engang bli vurdert av algoritmen. Det vil si at de går rett i den lukkede lista når de blir oppdaget.

I Python-koden er det benyttet to klasser for å definere tilstandsrommet: Pixel og Map. Pixelklassen representerer noder i klassen Map. Map klassen sitt navn kommer av fugleperspektivet til kameraet som får tilstandsrommet til å se ut som et kart i sanntid, og vil i praksis fungere på lik linje som et navigasjonssystemet i en båt som mottar satellittsingaler.

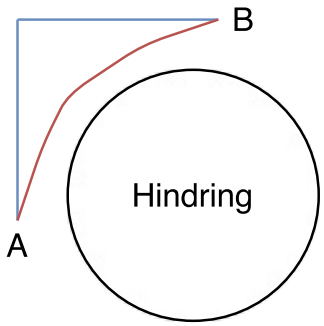
6.3.2 Søk

For å generere naboer i søket ser algoritmen på naboer som ligger i umiddelbar nærhet langs aksene i tilstandsrommet i høyden og i bredden. Dette er vist i Python-kode i figur 6.2. Man kan også se på naboer på tvers og dermed generere åtte naboer istedenfor, men fordi det er valgt å forenkle måten roboten manøvrer seg frem på - kjøring og svinging i to separate operasjoner - er det ønskelig at roboten skal endre retning så lite som mulig. Derfor er Manhatten-distansen brukt som heuristikk for at roboten skal ha preferanse for å fortsette å gå i rette linjer.

På grunn av måten roboten styrer på vil det i teorien kreve et uendelig antall retningsendringer å kjøre lands kanten på den sirkulære hindringen fordi en sirkel er en "uendelig-kant". Dette problemet er illustrert i figur 6.3. I praksis vil det kreve et ukjent antall retningsendringer som er proposjonalt med oppløsningen på bildet, størrelsen på objektet og hvilken offset som aksepteres av kontrolleren på kompasset før den søker å endre retning.

```
def generate_close_successors(self, track):
    successors = []
    if (self.y + 1 <= track.gridHeight - 1):
        pixel_up = track.map[(self.x), (self.y + 1)]
        successors.append(pixel_up)
    if (self.y - 1 >= 0):
        pixel_down = track.map[(self.x), (self.y - 1)]
        successors.append(pixel_down)
    if (self.x + 1 <= track.gridLength - 1):
        pixel_right = track.map[(self.x + 1), (self.y)]
        successors.append(pixel_right)
    if (self.x - 1 >= 0):
        pixel_left = track.map[(self.x - 1), (self.y)]
        successors.append(pixel_left)
    return successors
```

Figur 6.2: Søket genererer fire nabonoder i retning aksene i tilstandsrommet, altså høyden og bredden i bildet.



Figur 6.3: To forskjellige baner for å flytte roboten om et rundt objekt. Blå: Ved generering av fire naboer i høyden og i bredden. Rødt: Ved generering av 8 naboer i høyden, bredden og på tvers.

Algoritmen returnerer til slutt en redusert liste med veipunkter (koordinater) som kontrolleren ønsker å styre roboten mot. Reduseringen går ut på at man langs en rett linje kun trenger et start- og sluttpunkt fordi de andre punktene mellom dem er implisitt gitt siden roboten uansett kjører langs rette linjer mellom punkter. Hvis algoritmen finner en bane som kun er en rett linje vil den med andre ord returnere en liste med to punkter.

6.4 Kontrollsystem

Når en ønsket posisjon for roboten er beregnet blir den sendt til kontrollsystemet sammen med posisjonen til roboten. Et avvik mellom faktisk posisjon og ønsket posisjon blir regnet ut. Et tilsvarende avvik for kjøreretningen blir regnet ut. For å drive disse avvikene mot null, det vil si til en tilstand hvor roboten er i ønsket posisjon, brukes to PID-kontrollere.

Som nevnt i 5.5 er oppgaven begrenset til at kun en av kontrollerne kan kjøre om gangen, noe som forenkler kontrollsystemet betraktelig. Dette er løst ved å gi vinkelkontrolleren prioritet over avstandskontrolleren. I praksis vil det si at kontrolleren som styrer vinkelen er aktiv så lenge vinkelavviket er større enn en gitt verdi. Les mer om denne verdien under delkapittelet om utfordringer knyttet til kontroll i7.3.

Modulen får robotens posisjon samt koordinater til ønsket posisjon som inndata fra bildebehandlingen og banegenereringen. Hver kontroller er opprettet som objekter av klassen PID. Denne klassen tar utgangspunkt i denne diskrete PID-kontrolleren [2].

Kontrollerene tar inn en ønsket verdi ved oppstart, dette er verdien kontrolleren prøver å dra den faktiske verdien mot. Den målte verdien blir sendt til kontrolleren kontinuerlig som returnerer et pådrag. Pådraget sendes så som argument til NXT-funksjoner for svinging og kjøring.

6.4.1 Vinkelkontroll

Vinkelkontrolleren har høyest prioritet av de to kontrollerne. Dersom avviket er større enn tillat grense, aktiveres kontrolleren. Kontrolleren tar inn ønsket vinkel og roboten sin faktiske verdi. Ønsket vinkel blir regnet ut basert på posisjonen til roboten i forhold til ønsket posisjon i et todimensjonalt plan. Den trigonometriske likningen er gitt ved

$$\theta_{ref} = atan2((x_{ref} - x_{robot}), (y_{ref} - y_{robot})) \tag{6.1}$$

Denne verdien blir gitt til kontrolleren som referanseverdi når kontrolleren aktiveres.

Den faktiske vinkelen til roboten er hentet fra roboten sitt kompass. Denne verdien blir lest av og sendt til kontrolleren kontinuerlig. For mer om hvordan kontrolleren minimerer avviket mellom ønsket og faktisk verdi, se kapittel 4.3.

6.4.2 Avstandskontroll

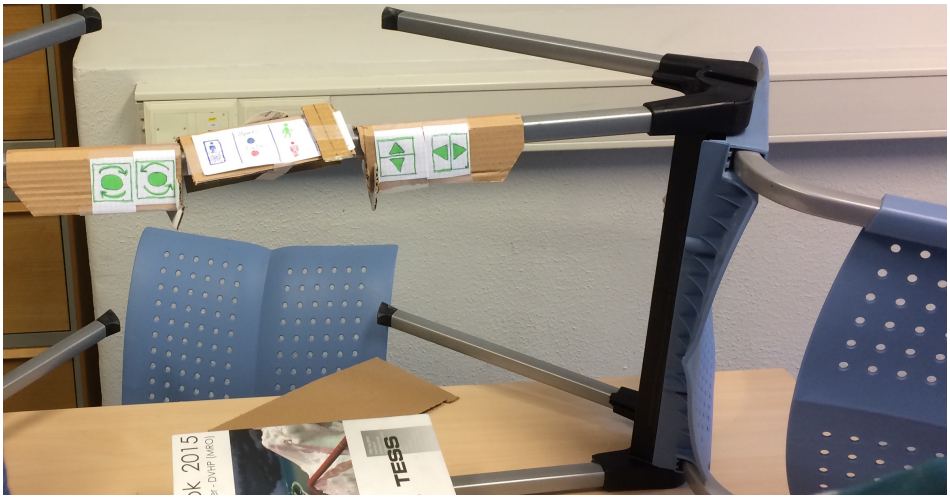
Avstandskontrolleren bruker antagelsen om at roboten kjører i riktig retning, det medfører at kontrolleren kun trenger å forholde seg til avstand langs en akse. De to-dimensjonale XY-koordinatene får kontrollsystemet som inndata fra bildebehandlingen. Disse konverteres til en en-dimensjonal lengde med Pytagaros læresetning,

$$distanse = \sqrt{(x_{ref} - x_{robot})^2 + (y_{ref} - y_{robot})^2} \tag{6.2}$$

Til forskjell fra retningskontrollen er ønsket verdi nå satt til å alltid være lik 0. Da er avviket lik distansen vi regner ut i likning (6.2). Denne lengden blir regnet ut kontinuerlig, basert på inndata og sendt direkte til avstandskontrolleren som jobber for å drive distanse mot null.

6.5 Brukergrensesnitt

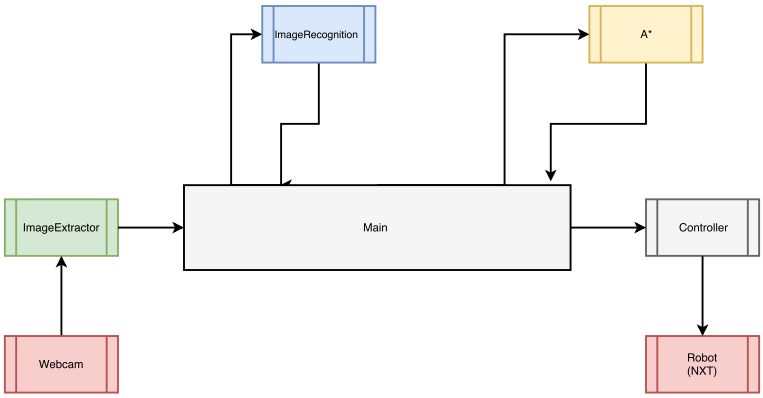
Det har blitt laget en prototype for et brukergrensesnitt for styring av sykesengen. Prototypen er utformet slik at brukeren kan holde hendene der de vanligvis holder dem for å trille på en normal sykeseng, altså på hodegavlen med omtrent en skulderbredde mellom hendene. For å få tilgang til manuell styring må man holde adgangskortet sitt mot en leser. Da vil det lyse grønt i "Manuell"-lyset. Deretter kan man styre sengen med styreknapper. Det er fire knapper for å flytte senga fram, tilbake, til høyre og til venstre. Disse knappene er plassert får å kunne brukes med høyre hånd, ettersom størsteparten av befolkningen er høyrehendte. For å rotere senga om sin egen senterakse er det to knapper, en for hver vei, på venstre side. Rotasjonsknappene er plassert på samme side for å gjøre det mulig å svinge med venstre hånd samtidig som man kjører med høyre hånd. I midten er det plassert et kontrollpanel med indikatorer og kortleser, samt en knapp for robotkontroll. Når man er ferdig med å flytte på sengen manuelt, kan man trykke på knappen "Robotmodus", og senga tar over styringen. Da vil også "Manuell"-lyset slukke, og "Auto" lyset vil tennes. Nedenfor lysene vil det være plassert en batteriindikator som viser strømnivå både grafisk og med en skjerm som viser gjenstående strømnivå i prosent. Prototypen som er vist i figur 6.4 ble resultatet etter tre iterasjoner med testing.



Figur 6.4: Tredje prototype til brukergrensesnitt.

6.6 Koordinering av komponenter

Hver komponent hadde hvert sitt isolerte ansvarsområde. Figur 6.5 viser hvordan hovedprogrammet fasiliterte kommunikasjonen mellom komponentene. Hver komponent fungerte som en egen prosess som kommuniserte med neste prosess ved å bruke IPC (inter-process communication).

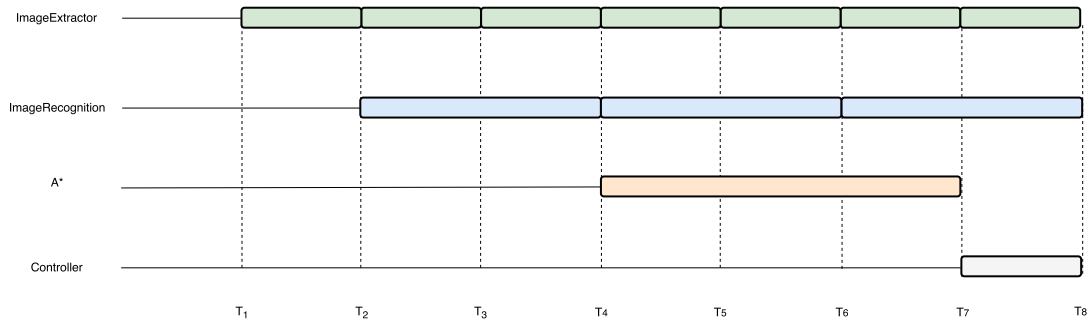


Figur 6.5: Komponentene i hovedprogrammet

Tabell 6.6 beskriver hva hver enkelt komponent tok inn som input og hva den produserte av output.

Komponenter I/O		
Komponent	Input	Output
`Webcam`	—	Video
`ImageExtractor`	Video	JPEG-bilde
`ImageRecognition`	JPEG-bilde	Robot (x, y) og 2d matrise
`A*`	2d matrise	Waypoints (x, y)
`Controller`	Robot (x, y) og waypoints (x, y)	Pådrag til motor
`Robot (NXT)`	Pådrag til motor	—

Tidsdiagrammet i figur 6.6 viser hvor lang tid hver komponent brukte på å behandle input og produsere output. For å gjøre programmet så effektivt som mulig jobbet hver komponent i parallell. Med andre ord kunne for eksempel `ImageRecognition` utføres uavhengig av om `A*` var opptatt med å finne korteste vei. Ettersom hver komponent var en egen prosess ble det operativsystemet sin oppgave å disponere CPU-ressursene best mulig.



Figur 6.6: Tidsdiagram over komponentes arbeidsmengde

Et annet eksempel som belyser viktigheten av å utføre oppgavene i parallell var samspillet mellom `ImageRecognition` og `Controller`. Dersom sistnevnte måtte vente på at `A*` fant korteste vei mellom robotens posisjon og destinasjonen ville effektiviteten vært mye lavere.

7 Resultater og testing

I starten av prosjektet ble det definert klare mål for hva produktet skal takle av utfordringer. Disse målene representerer forskjellige vanskelighetsgrader for navigering igjennom en gang hvor alle målene må være bestått for at en eventuell full-skala implementasjon av roboten skal kunne - på tilsfredsstillende måte - navigere igjennom et sykehus. Før modulene i systemet kobles sammen og kan testes i sin helhet, ble hver modul testet isolert fra de andre modulene. Når alle modulene er ferdigstilte og fungerer tilfredstillende, kobles de sammen og testing av hele robotsystemet startes. Det er også gjort grundig testing av brukergrensesnittet til styringenheten på sengen.

7.1 Testing av Datasyn

Datasynbiten av implementasjonen ble testet ved hjelp av ad hoc testing [9] i løpet av utviklingsfasen. Dette ble gjort ved å mate inn bilder av samme størrelse, format og utforming som det bildene fra kamera var. Algoritmene for å segmentere bildet og finne roboten gav med dette gode resultater, som stemte overens med det som var forventet. Den fant objekter der den skulle og klarte å finne robotens posisjon med god presisjon.

7.2 Testing av banegenerering

Ved testing av banegenereringsmodulen ble det oppdaget problemer knyttet til kjøretiden til A*. Kjøretiden og minnet på algoritmen øker eksponentielt med størrelsen på tilstandsrommet. Hvis algoritmen må finne en relativt vanskelig bane vil også kjøretiden øke fort. Det er brukt en oppløsning på 200×200, og til og med på så små bilder er kjøretiden på algoritmen relativt stor.

Løsningen på dette ble å generere naboer ved hjelp av steglengder. Dette betyr at ikke nærmeste nabo blir generert, men naboer i en steglengde. Dette gjør at tilstandsrommet effektivt

Execution time with stride 1 = 0.44004297256469727

Execution time with stride 5 = 0.0009992122650146484

Figur 7.1: Kjøretiden på algoritmen med henholdsvis steglengder en og fem på et 200×200 bilde.

blir mindre med en faktor lik steglengden. Ytelsesforbedringen som følge av dette trikset er vist i figur 7.1, og søket går nesten 500 ganger så fort. Python koden for dette er vist i figur 7.2.

```
def generate_stride_successors(self, track, stride):
    successors = []
    if (self.y + stride <= track.gridHeight - 1):
        pixel_up = track.map[(self.x), (self.y + stride)]
        successors.append(pixel_up)
    if (self.y - stride >= 0):
        pixel_down = track.map[(self.x), (self.y - stride)]
        successors.append(pixel_down)
    if (self.x + stride <= track.gridLength - 1):
        pixel_right = track.map[(self.x + stride), (self.y)]
        successors.append(pixel_right)
    if (self.x - stride >= 0):
        pixel_left = track.map[(self.x - stride), (self.y)]
        successors.append(pixel_left)
    return successors
```

Figur 7.2: Søket genererer fire nabonoder i steglende på 5 langs aksene i tilstandsrommet, altså høyden og bredden i bildet.

En fallem å være obs på er at steglengden ikke må være lengre enn den tynneste hindringen på bildet slik at algoritmen ikke risikerer å overse den.

7.3 Testing av kontrollsystem

For å teste kontrollsystemet som en individuell modul ble det satt opp en testprosedyre, samt at modellen ble tilpasset testing uten andre moduler. Den enkle testen som ble utført var å gi kontrollsystemet et settpunkt og la kontrollsystemet manøvrere til dette punktet. Siden feedback fra datasynet om posisjonen til roboten ikke er inkludert i denne testfasen, brukes ultralydsensoren til denne testen. Kontrollsystemet blir gitt et tilfeldig settpunkt for vinkel, mens settpunkt for posisjon er alltid definert som null.

Testprosedyren er som følgende:

- En vinkel blir gitt som settpunkt til vinkelkontrolleren
- Roboten starter i en tilfeldig orientering
- Kontrollsystemet skal stille inn orienteringen til roboten
- Roboten kjører til den møter første hindring og stopper
- Gjenta for å skjekke reproduserbarhet

Ved gjennomføring av denne prosedyren ble det oppdaget at Lego-NXT har flere svakheter. Kompasset som er påmontert roboten viser seg å være veldig sensitivt for støy. Støyen ga utslag på målingene og det var vanskelig å få nøyaktige målinger.

Hver av motorene til roboten styres med et pådrag fra -100 til 100, hvor -100 er full fart bakover, 0 er stillestående og 100 er full fart framover. Ved testing har det vist seg at motorene trenger minimum $\pm$ 60 i pådrag for bevege roboten. Ved alle verdier lavere enn dette vil ikke roboten bevege seg.

En annen svakhet som ble oppdaget under testing er at motorene oppfører seg forskjellig basert på hvor mye batteri det er igjen på modellen. Batteriet har kort levetid og dermed skal det lite til før kjøreegenskapene endrer seg. Dette fører til utfordringer når det kommer til tuning av kontrollerene.

Isolert fra de andre modulene er kontrollsystemet nå i stand til å kjøre til- og stoppe på et gitt referansepunkt basert på feedback fra kompass og ultralydsensor. Når kontrollsystemet kobles til de andre modulene vil feedback-data fra ultralydsensoren byttes ut med koordinater gitt fra datasnsmodulen. Grunnet forskjellig oppløsing på koordinatene i forhold til ultralydsensoren, er det naturlig at posisjonskontrolleren må tunes på nytt.

7.4 Testmiljø for seng

For å teste robotens egenskaper, er det blitt utarbeidet flere delmål for hvilken utfordringer den skal klare å forholde seg til. Det er totalt utarbeidet fem manøvreringstester for roboten, hvor testene blir gradvis mer avanserte. De forhåndsdefinerte testene var:

- Sengen skal manøvrere gjennom en tom, rett gang.
- Sengen skal manøvrere gjennom en tom gang med en 90° sving.
- Sengen skal manøvrere gjennom en gang med 90° sving og statiske hindringer.
- Sengen skal manøvrere gjennom en gang med 90° sving og bevegelige hindringer.
- Sengen skal kjøre tilbake til start dersom den den ikke finner en vei til målet.

Før testing er det nødvendig å kalibrere kontrollsystemet slik at nullreferansen for vinkel er den samme for kompasset som i kameraet. Dette ble gjort ved å manuelt måle vinkelen basert på bildestrømmen fra kameraet og deretter legge på en konstant verdi på kompassmålingen.

Ved testing ble det oppdaget mange feil i kommunikasjonen mellom modulene, som førte til at roboten ikke oppførte seg som ønsket. Det ble gjort mye feilsøking for å lokalisere årsakene til problemet, men for å overholde tidsrammene til prosjektet ble ikke denne feilsøkingsfasen fullført. Vi kunne derfor ikke gjennomføre noen av testene som var tenkt.

7.5 Brukergrensesnitt

Det er viktig at brukergrensesnittet til styringsenheten på sengen er intuitivt og enkelt å bruke for portører og andre som trenger å flytte på sengen. Samtidig er sikkerhet et viktig aspekt. Ikke hvem som helst skal ha tilgang til å styre sengen, men samtidig bør nødstoppbrytere være strategisk plassert slik at de er lett tilgjengelig dersom det oppstår en situasjon. Det er derfor utarbeidet en rekke tester for å finne den beste løsningen.

7.5.1 Brukertesting innad gruppen

For å teste ut hvordan brukergrensesnittet ville oppleves i en reell brukersituasjon, ble det festet en pappmodell av grensesnittet på to stolbein. Den første prototypen hadde knapper for styring og indikatorlys for manuell og automatisk modus. Stolene ble lagt på et bord for å få høyden nogenlunde tilsvarende høyden på sengegavlene på sykesenger. Først ble det prøvd en modell som var ca. 30 cm bred. Etter litt utprøving og tilbakemelding fra gruppemedlemmer ble det funnet ut at man vanligvis holder bredere når man flytter noe så stort og tungt som

en seng, så knappene ble flyttet lengre fra hverandre slik at brukeren kan holde hendene der de vanligvis holder dem når de flytter på senger. Videre ble det prøvd ut å ha styring framover og svinging den ene veien på én hånd, og motsatt på den andre hånden. Videre ble det funnet ut at det ville hindre mulighet for å svinge og kjøre framover samtidig, og endte deretter til slutt med en løsning hvor svingeknappene er på en hånd, mens framover, bakover og sidelengs-knappene er på den andre hånden.

Etter tilbakemelding fra medlemmer av gruppen, ble det lagt til kortleser og batteriindikator, samt nødstopp-knapp i grensesnittet.

7.5.2 Brukertest 1

Under intervjuene med sykepleierne ble de konseptene som var undersøkt testet for tilbakemelding. Og verdifulle idéer kom frem.

Konseptet ble presentert med at sengene skal kjøre autonomt med mindre de har pasienter i eller må hjelpes løs av mennesker rundt seg. Sykepleier 2 reagerte svært positivt på muligheten til å kjøre sengen manuelt med motorer, ettersom belastningen for den som flytter på sengen da blir betraktelig mindre. Denne funksjonalitet kunne vært vurdert som en mulighet for når roboten har kjørt seg fast, men dersom roboten først har kapasitet til å kjøre lange strekk autonomt og til å kunne overstyres manuelt på stedet, vil det implisere at den kan brukes manuelt med motorer til forflytning av pasienter. Dersom denne funksjonen skal brukes mye, stiller det ekstra krav til brukervennlighet og ergonomisk utforming. Denne oppfatningen delte sykepleierne også, med bakgrunn i at det ikke er alle sykepleierne som er like komfortable med ny teknologi generelt.

Sykepleierne påpekte videre at det er nødvendig for de ansatte å kunne ta helt fullstendig kontroll over sengene. For at intervjuobjektene skulle føle seg trygge rundt robotene, ønsket de en veldig tydelig løsning på å skille på når roboten kan gjøre noe som helst selv, og når sengen styres helt og fullt av brukeren.

7.5.3 Brukertest 2

Den første prototypen på brukergrensesnitt ble presentert for en ny portør (portør 2) under intervju 27.mars. Det ble montert en prototype på en eksisterende seng av gammel type, forklarte scenarier for portøren og spurte ham deretter hvordan han ville gått fram for å bruke sengen. Hva som var hva i grensesnittet ble forklart, men de tiltenkte funksjonene ble

ikke forklart. Portøren fikk altså vite hva som var kortleser, hva som var lys og indikatorer og hva som var knapper.

Det første scenariet var at sykesengen hadde kjørt seg fast og ba om hjelp. Portøren skulle så flytte på sengen manuelt. Portøren ble spurt om hva han ville gjøre. Han forsto hvilke knapper som gjorde hva uten forklaring og ville kunne brukt sengen uten bruksanvisning. Han var fornøyd med knappenes plassering og utforming, og syntes at grensesnittet var intuitivt utformet. Det ble videre spurt om hva man mente en eventuell bruksanvisning burde inneholde. Etter litt vurdering fram og tilbake bestemte han seg for at det ville holde med en tekst under hver knapp med forklaringer, f.eks. Roter høyre"under knappen som gjør det. Han mente at det var tilstrekkelig til å gjøre en bruksanvisning unødvendig.

Portør 2 likte konseptet svært godt og mente at autonome sykesenger slik som det ble beskrevet for dem ville være veldig bra for sykehuset og avlaste kjedelig arbeid fra portørene.

8 Diskusjon

For å overholde tidsrammene til prosjektet ble det tatt en beslutning om å avslutte testfasen selv om roboten ikke var i stand til å fullføre målene som ble satt i starten av prosjektet. Feilsøking på komplekse program kan ta mye tid og det kan være vanskelig å finne ut hvor i programmet eventuelle feil ligger. På bakgrunn av dette er det naturlig å tro at en løsning ville vært nærliggende dersom det hadde vært mer tid til feilsøking. Men at prosjektet ble avsluttet før en prototype som fungerte i den grad det var tenkt i løpet av prosjektet gjør at det er vanskeligere å diskutere resultater siden ikke alle resultatene kan tolkes uten å ha fullført prosjektet. Forslag til hva som kan gjøres for å buktes med problemene som ble oppdaget og forbedringer er å finne i kapittel 9.

8.1 Brukergrensesnittet

Hva angår brukergrensesnittet som ble utarbeidet forsto brukeren hvordan brukergrensesnittet var ment å brukes uten instruksjoner. Alle medlemmene av gruppa forstod hvordan grensesnittet skulle brukes intuitivt, men medlemmene av gruppa er kanskje ikke representative for andre større gruppe mennesker med tanke på kunnskaper om roboter og teknologi generelt. Det ble gjort en antagelse på at grensesnittet var ganske selvforklarende, og fikk dette bekreftet. Det er mulig at et større utvalg av prøvebrukere ville gitt et annerledes resultat, men vår testperson mente at dette ikke ville vært tilfelle dersom det hadde blitt lagt til litt tekst ved hver knapp.

8.2 Datasyn

Datasynet fungerte godt og resultatene var som forventet. Men til en fullskala implementasjon av en sykeseng så vil det sannsynligvis bli for simpelt. For eksempel vil det være nødvendig å skille mellom hindringer for datasynet, og dersom det er flere roboter i et bilde må datasynet

også kunne skille mellom dem for å unngå kaos. Det er i tillegg usikkert hvordan denne kompleksitet kan påvirke kjøretiden til programmet.

8.3 A*-algoritme som navigasjonssystem

Algoritmen for banegenerering viste seg å være lite robust med tanke på kjøretid da den vokste eksponentielt både i minne og kjøretid. Siden kjøretiden er såpass varierende og lite forutsigbar er det vanskelig å tro at dette er en gjennomførbar metode dersom man trenger bedre oppløsning på bilder, noe man sannsynligvis trenger hvis man skal navigere i en gang med mennesker.

8.4 Rådata fra kompasset til Lego Mindstorm roboten

En stor svakhet med Lego-robotten har visst seg være rådataen som kommer fra dens output enheter. Målingen fra kompasset viste seg å være svært påvirket av støy og fungerte derfor svært dårlig for navigering av roboten, og filtrering av dataen er nødvedig for å få et brukbart resultat.

8.5 Kontrollsystem

Det ble oppdaget flere utfordringer knyttet til den fysiske modellen ved testing. Ettersom det ikke er aktuelt for denne oppgaven og gå over til en mer egnet modell, ble svakhetene i modellen tatt hensyn til i kontrollsystemet. Problemet med at motorene til roboten er for svake til å kjøre roboten ved lavt pådrag er løst ved å skalere pådraget fra kontrollerene slik at det alltid vil være kraftig nok for roboten til å kjøre. Dette løser problemet, men det er verdt å merke seg at en slik løsning fører til at kontrolleren har et mye mindre område å regulere på, noe som gjør det vanskeligere for kontrolleren å gjøre finjustering i pådraget. Nøyaktigheten til roboten er dermed svekket som følge av dette. Med manglende evne til finjustering er det blitt lagt inn terksler for hvor nærme ønsket verdi som er godtatt av kontrolleren.

Problemet med at modellen oppfører seg forskjellig basert på batteriprosenten er unngått ved å gjøre en antagelse om at batteriet alltid er fullladet og tune kontrolleren basert på dette. Alternativet er igjen å gå for en annen modell, noe som ikke lar seg gjennomføre i tidsrommet som er tilgjengelig.

8.6 Produktet

Metoden for å navigere roboten basert på datasyn og banesøk effektiviserer roboten med tanke på banen den velger for å kjøre til målet sitt. Dette kan være svært nyttig i operasjoner hvor for eksempel energi og drivstoff er kritisk, og vil fungere bedre enn i metoder hvor en kun navigerer med f.eks et kamera eller sensorer montert på roboten. Dette kommer av at man ved hjelp av et kamera i taket får et fullt observerbart tilstandsrom for navigering og man kan derfor vite hvilken bane det lønner seg å ta.

Med en fungerende løsning ville dette kunne hjelpe på med å redusere belastning på portører ved sykehus. Dette betyr at de kan bruke tiden sin til andre ting enn å kjøre rundt på tomme, tunge senger. Dette kan også redusere belastningsskader hos portører.

9 Konklusjon

Selv om kravene som ble satt for denne oppgaven ikke ble møtt på grunn av tidsmangel er det naturlig å tro at en løsning på problemmene er nærtliggende. Når det er sagt er løsningen vår kun en prototype av en sykeseng. Den viser de grunnleggende funksjonene til en autonom sykeseng og hvordan den er tenkt i bruk. Derfor har brukerundersøkelsene som er gjort vært til stor hjelp for å få en bedre forståelse for behovet og hva som er vitkig for personene som skal omgåes slike senger.

Løsning er ikke nødvendigvis optimal på noen måte og det mye arbeid igjen før en virkelig sykeseng kan bygges og virke i den grad det er godt nok. Her presenteres hva som kan arbeides med videre på roboten og hvilke utfordringer som kan vise seg å være gjeldende, samt forslag til alternative løsninger for enkelte moduler.

9.1 Videre arbeid

For at denne oppgaven skulle være gjennomførbar i tidsrommet som var gitt ble det gjort en rekke forenklinger i starten av prosjektet. Disse er oppført i kapittel 5.5. Videreutvikling av denne oppgaven bør fokusere på å lage en modell som fungerer utover disse begrensingene. Videre nevnes flere forslag til forbedringer i mer detalj.

9.1.1 Forbedringer av modell

For at en autonom sykeseng skal ferdes trykt blandt mennesker i en sykekuskoridor trengs det langt flere sensorer på sengen enn det som er implementert i dette prosjektet. Et videre studie av hvilke sensorer som er best egnet til formålet er anbefalt. Spesielt en vurdering av Lego-roboten sitt eget gyroskop bør gjøres.

Banegenerering

Kjøretid er av en avgjørende faktor for systemet både med tanke på presisjon, båndbredde og sikkerhet. Som vist i kapitel 6.3 skalerer A* algoritmen svært dårlig med større bilder og selv om man bruker steglengder vil man ikke oppnå en tilfredsttilende kjøretid. En løsning som kan få søket til å gå fortere (hvor steglengder kan brukes i tillegg) er å bruke en grådig søkealgoritme, eller modifisere eksisterende algoritme til å være grådig. Dette vil si at den går den første og beste veien den finner i den retning hvor målet er (som den har en viss ituisjon av på grunn av heuristikken) istedenfor å planlegge å søke igjennom alle potensielle noder for å finne en vei.
Ulempen med dette er at banen som algoritmen finner ikke lenger kan garanteres å være den korteste (sannsynligvis vil den ikke være det), og at roboten derfor kan finne på å gå ruter som for en menneskelig observatør vil virke ineffektivt og lite intuitivt. Dette kan i verste fall føre til at roboten oppfører seg uforutsigbart som gjør menneske-robot samspillet vanskeligere.

Bevegelige objekter

Det ville vært til stor hjelp for roboten dersom den kan gjenkjenne bevegelige objekter, og finne ut hvor fort disse kan bevege seg. Den vil da kunne navigere bedre i et miljø der ting kan bevege seg, siden den da kan forutse hvor ting kommer til å være innenfor et visst tidsperspektiv og kan med større treffsikkerhet unngå kollisjoner med disse objektene.

Kontinuerlig- og fin styring av robot

Implementasjonen presentert i denne rapporten forutsetter at roboten gjør dreining og kjøring i to separate operasjoner. Dette ble gjort som en forenkling for å det til å passe med tidsrammen til prosjektet. Dette er naturligvis veldig ineffektivt, og det vil være mer effektivt å konstruere en kontroller som gjør begge deler samtidig.

I tillegg er ikke kontrolleren nøyaktig nok for fin navigering og derfor kan det være nødvendig å få til mykere bevegelser med PWM (Pulse Wave Modulation) på inputen til motorene til roboten.

Anti-kolisjons system

Med tanke på bruksområde til roboten (sykehus) er sikkerhet meget viktig. Derfor må det eksistere et system som passer på at roboten ikke kjører på og skader mennesker eller andre ting. Dette kan f.eks tenkes å implementeres ved hjelp av flere ultralyd sensorer plassert på roboten som har full 360 graders oversikt over objekter rundt den. Disse dataene kan brukes til å forutse kolisjoner og roboten kan dermed reagere når den mener at noen kommer for nærme eller står i fare for å kollidere med den.

Filtrering av rådata

Rådataen som kommer ut av kompasset er svært påvirket av støy. For å kunne navigere presist må derfor denne dataen kalibreres og korrigeres. Dette kan tenkes å gjøres ved hjelp av et Kalman filter for tilstandsestimering.

9.1.2 Alternative løsninger

Kamera plassert på roboten

Istedenfor å ha flere kamera i taket kan et kamera settes på selve roboten. Det vil kreve en annen måte å finne vegen på, men vil være en mer realistisk løsning i forhold til den som har blitt laget til nå, med tanke på tilgjengelig infrastruktur og personvern som på et sykehus. Hvis en i tillegg bruker f.eks ultralyd sensorer kan dataene brukes til å danne et 3D kart av objekter i omegn roboten, og roboten kan dermed navigere ved hjelp av kartet og et kompass/gyroskop gitt at den vet hvilken retning den skal.

Datasyn

Et punkt som kan forbedre datasyn biten av roboten vil være å bruke former for maskinlæring, som deep learning med Artificial Neural Networks (ANN), for å lære seg hvor den kan og ikke kan kjøre (lærer hva som er hindringer). Disse nettverkene or ofte gode på å lære seg slikt, spessielt Convolutional Neural Networks (CNN) er gode på å lære seg å se ting i bilder som gjør at den kan svare på det den skal lære seg (kan/kan jeg ikke kjøre her?). Etter de er ferdig trent er de også veldig raske på å beregne svaret når de får inn et bilde. Med tanke på tiden som er tilgjengelig på et semester så hadde det nok ikke vært god nok tid til å få nok treningsdata av god kvalitet, og tid til å trene nettverket, til at det kan bli av god kvalitet.

Det ble derfor ikke sett på som en realistisk løsning innenfor tidsrammene som ble gitt, men for videre arbeid er det noe som kan gjøre en betydlig forbedring av datasynet og gjøre det langt mer robust.

Bibliografi

[1] Nxt utviklersett for maskinvare. https://lc-www-live-s.legocdn.com/r/www/r/mindstorms/-/media/franchises/mindstorms/2014/downloads/firmwareandsoftware/nxtsoftware/hdk_download1.zip?l.r2=-1260971408, 2006.

[2] ivpid: Python pid controller. https://github.com/ivmech/ivPID/blob/master/PID.py, 2016.

[3] Multi-dimensional image processing (scipy.ndimage). https://docs.scipy.org/doc/scipy/reference/ndimage.html, 03 2017.

[4] Object detection matchtemplate. http://docs.opencv.org/2.4/modules/imgproc/doc/object_detection.html?highlight=matchtemplate#matchtemplate, 2017.

[5] Opencv. http://opencv.org/, 2017.

[6] Aethon. Tug: How it works. http://www.aethon.com/tug/how-it-works/.

[7] Steen Dawids Henrik Aanes Jens Klestrup Kristensen Kim Hardam Christensen Ali Gurcan Ozkil, Zhu Fan. Service robots for hospitals: A case study of transportation tasks in a hospital. http://ieeexplore.ieee.org/document/5262912/.

[8] Kalle Turkerud Anette Holth Hansen. «effektive roboter frigjør mer tid til pasienter». https://www.nrk.no/ostlandssendingen/effektive-roboter-frigjor-mer-tid-til-pasienter-1.12644057.

[9] James Bach. Exploratory testing explained, 2003.

[10] John Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698, 1986.

[11] Ray Clout Hung T. Nguyen Chao Wang, Andrey V. Savkin. An intelligent robotic hospital bed for safe transportation of critical neurosurgery patients along crowded hospital corridors. http://ieeexplore.ieee.org/document/6879265/authors.

[12] Keith L. Downing. *Essentials of the A* algorithm.* Institutt for datateknologi og informatikk, NTNU.

[13] Matt Edwards. The difference between agvs and mobile robots. https://cross-automation.com/blog/difference-between-agvs-and-mobile-robots.

[14] Elvin Luff et. al. https://github.com/Eelviny/nxt-python, 2012.

[15] Kate Sharkey Julia Stepowska. Robotic hospital beds are the future of patient transportation. https://medicalxpress.com/news/2014-11-robotic-hospital-beds-future-patient.html.

[16] Federico Lozano. Lecture notes in tmm4121 product development, 2015.

[17] Satya Mallick. Filling holes in an image using opencv (python / c++). http://www.learnopencv.com/filling-holes-in-an-image-using-opencv-python-c/, 11 2015.

[18] Richard E. Woods Rafael C. Gonzales. *Digital Image Processing.* Pearson Education, Inc., Upper Saddle River, New Jersey 07458, 3 edition, 2008.

[19] Matt Simon. This incredible hospital robot is saving lives. also, i hate it. https://www.wired.com/2015/02/incredible-hospital-robot-saving-lives-also-hate/.

[20] Swisslog. ≪transcar® automated guided vehicle≫. http://www.swisslog.com/en/Products/HCS/Automated-Material-Transport/TransCar-Automated-Guided-Vehicles.

[21] Eric B. Utheim. ≪hei, kanj du pass dæ litt?!?≫. http://e24.no/digital/hei-kanj-du-pass-dae-litt/20339749.

[22] Wikipedia. Automated guided vehicle. https://en.wikipedia.org/wiki/Automated_guided_vehicle.