

# Semesteroppgave ELE306

Gruppe 12

Oppgave 6

574938 Trygve Sognnæs

578146 Sivert Elias Åmelfot

578144 Robert Løland

578153 Andrés David Rodríguez Delgado

19.11.2020

## Innholdsfortegnelse

|  |    |
|--|----|
| 1. Introduksjon.....   | 3  |
| Problemet .....  | 3  |
| Løsningen .....  | 4  |
| 2.Designprosess .....  | 5  |
| Designiterasjon .....  | 5  |
| Utfordringer .....   | 5  |
| 3. Implementering .....  | 8  |
| Kinematisk modell.....   | 12 |
| Navigasjonsalgoritme.....  | 13 |
| 4. Eksperiment .....   | 15 |
| 5. Konklusjon.....   | 16 |
| Utfordringer ved implementering i den virkelige verden .....               | 16 |
| Fordeler ved implementering i den virkelige verden .....                   | 16 |
| Lenke til GitHub repository .....  | 17 |
| Bibliografi .....  | 17 |
| Appendiks .....  | 18 |
| Appendiks A: Gantt – diagram .....   | 18 |
| Appendiks B: MATLAB – skript.....  | 19 |
| Appendiks C: PDR.....  | 26 |
| Appendiks D: CDR .....   | 29 |
| <br>   |    |
| Figur 1: Det imaginære rektangelet rundt et fartøy på sjøen.....           | 6  |
| Figur 2: Forflyttet Trajektorpunkt .....                                   | 8  |
| Figur 3: Bøyens koordinatsystem.....                                       | 9  |
| Figur 4: Robotarmens DH-parameter .....                                    | 9  |
| Figur 5: De ulike referansesystemene på båten .....                        | 10 |
| Figur 6: Transformasjonsmatrisene del 1 .....                              | 10 |
| Figur 7: Transformasjonsmatrisene del 2 .....                              | 11 |
| Figur 8: Utregning av bøyelagring.....                                     | 11 |
| Figur 9: Bøyen .....   | 11 |
| Figur 10: Unicycle modell og dens formler 2 .....                          | 12 |
| Figur 11: Båten følger en trajektor .....                                  | 12 |
| Figur 12: Estimering av møtepunkt ved hjelp av probabilistic roadmap ..... | 14 |
| Figur 13: Bruk av D* til å nå møtepunktet.....                             | 15 |

# 1. Introduksjon

## Problemet

Vi valgte oppgave 6 som gikk ut på å designe en autonom båt som skulle plukke opp plastikk fra sjøen ved hjelp av en robotarm montert stasjonært på dekk. Det første vi gjorde var å søke opp lignende design på nett for å få et overblikk over hva som har blitt gjort fra før. De fleste løsningene vi fant baserte seg på et prinsipp som var svært ulikt det som var gitt i oppgaven, nemlig å på et eller annet vis lede store mengder plastikk inn på et samleband [1] [2] [5] [6]. Vi vurderte denne løsningen som en klar vinner, og det virket lite hensiktsmessig å skulle plukke plastikk fra sjøen med en liten robotarm montert på en 1 meter lang båt, som ikke ville kunne lagre en nevneverdig mengde før den måtte returnere til land. Derfor valgte vi å endre oppgaven til noe vi mente kunne være et bedre bruksområde til en slik autonom båt og dermed gjøre det mer motiverende å gjennomføre semesteroppgaven. Det gjøres mye forskning på sjøen med tanke på klimaet og det finnes sensorer plassert på forskjellige plasser for å holde øye med utviklingen av blant annet temperatur og pH [7] [8]. Derfor tenkte vi oppgaven kunne være å samle inn bøyer av en standard utforming som kan inneholde ulike sensorer. Tanken er at dersom slike sensorbøyer blir hentet av en liten elektrisk autonom båt, kan kostnadene ved denne operasjonen potensielt reduseres flere magnituder, i forhold til om et bemannet skip skulle gjort den samme jobben [9]. Innsamlingen av bøyene vil gjerne ta lengre tid, men man kan ha mange slike autonome båter som jobber samtidig og de kan jobbe dag og natt. Dette oppgaveforslaget fikk vi godkjent av faglærer.

Vi valgte å beholde de fysiske kravene til båten og robotarmen. Det vil si at båten skal være en meter lang, 40 cm bred, veie ca. 30 kg, ha en svingradius på mindre enn en meter og ha to propeller montert bak med 30 cm mellom som brukes til å differensialstyre båten. Båten vil nok ikke kunne laste 30 kg siden vi har designet den for å kunne lagre opptil tre sensorbøyer med en radius på 12,5 cm som hver veier opp til 4 kg. Kameraet som er nevnt i oppgaven, RealSense D435, er det vi endte opp med å bruke. Det gir båten god synsvinkel, oppløsning og bilderate. Vi har også antatt at det ikke er noen bølger, men det er en konstant strøm på 0,1 meter per sekund. Armen skal ha minst fire DOF, en ekstra DOF i griperen, nå 60 cm i det horisontale planet og kunne løfte en last på fire kg pluss griperens vekt på en kilo.

## Løsningen

Siden vi valgte å beholde spesifikasjonene til båten som var gitt i oppgaven, bruker vi en unicycle-model til styringen. Kartleggingen vil bli gjort ved  $D^*$  og når båten nærmer seg målet vil den følge en trajektor for å kunne matche farten til bøyen og plukke den opp. Posisjonen til bøyen vil bli funnet ved hjelp av en GPS-modul montert bak på båten. For at robotarmen skal kunne plukke opp bøyen må vi finjustere båtens positur, noe som krever en mer nøyaktig posisjon til bøyen enn det GPS leverer.

Derfor har vi valgt å bruke et 3D-kamera som kan gi rask og nøyaktig positur til bøyen, til enhver tid. Armen vil kunne plukke opp bøyene ved å gripe et håndtak som bøyen vil være utstyrt med, se figur 3. Deretter vil armen kunne løfte bøyen opp i båten og fortsette til neste mål. Valget av ledd som robotarmen bruker er gjort stort sett med tanke på at den skal kunne være i stand til å løfte en vekt på opptil 4 kg, som vil være maksvekten på bøyene. Prismatiske ledd vil være bedre egnet til løfting enn rotasjonsledd og designet til robotarmen reflekterer dette. Implementering av en vinsj har også vært en del av planen for å redusere stresset på robotarmens ledd. Vi tror at løsningen vår vil være en effektiv og billig måte å kunne samle inn slike bøyer på og at designet er optimalisert for dette formålet.

## 2.Designprosess

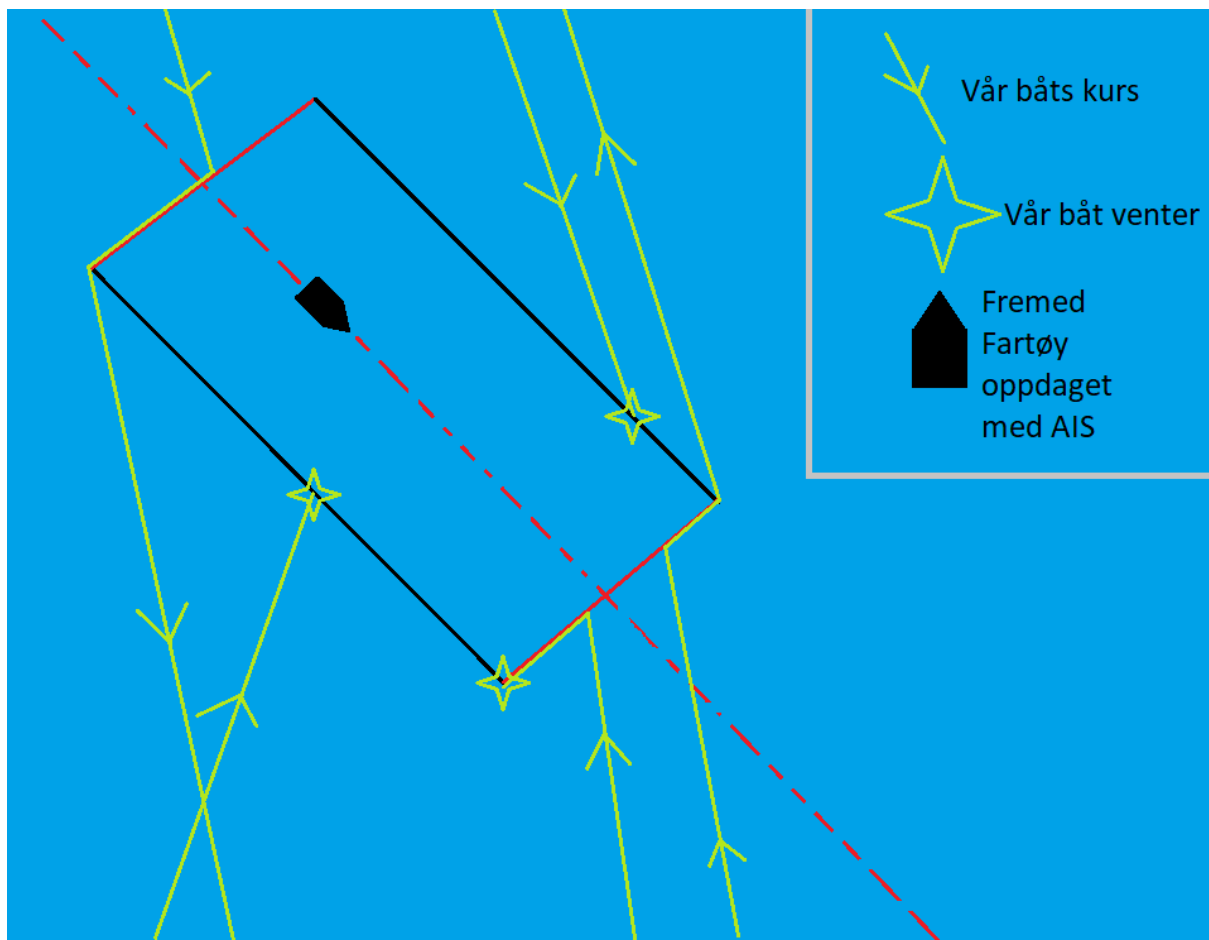
### Designiterasjon

Så snart vi hadde fått den omdefinerte oppgaven på plass begynte vi å tenke på designet av robotarmen, designet av båten er mer låst og ikke like mye i fokus i denne oppgaven. Først tenkte vi på en liten tradisjonell robotarm med kun roterende ledd. Vi studerte kravene og kom frem til at dette kanskje ikke ville være et veldig realistisk design for denne oppgaven, da det ene kravet var å kunne løfte opptil 4 kg. Ettersom robotarmen er såpass liten vurderte vi at rotasjonsledd kanskje ville slite med å løfte denne vekten. Derfor valgte vi heller å gå for to prismatiske ledd som gjør den tyngste jobben, rotasjon i basen, et rotasjonsledd for å kunne tippe opp armen og et rotasjonsledd for griperen. På skisser av armen tegnet vi på en vinsj som skal hjelpe robotarmen å tippe opp, men dette er ikke implementert i Matlab (selv om den ikke påvirker kinematikken), men vi ville måtte utvikle en kontroller for å styre vinsjen. Dette er noe vi ikke har kompetanse til.

Den eneste designendringen vi har gjort på båten i forhold til original design vist på PDR er endring av baugen fra Ulstein X-BOW til en konvensjonell baug for å få mer plass på dekk til robotarmen, og dermed mer plass til lagring av sensorbøyene.

### Utfordringer

Et problem som oppstår når man har en autonom båt på sjøen er at man må unngå å krasje med andre båter. Siden antallet båter er stort og de alltid er i bevegelse kan en ikke ta hensyn til de i navigasjonsalgoritmen, men må bruke en annen løsning. Vår løsning er en kombinasjon av bruken av kameraet vårt og AIS, automatisk identifikasjons-system, som gir GPS-posisjon, fart og kurs på mange store båter over hele verden. For slike båter vil vi kunne bruke AIS for å sjekke om de kommer til å kollidere med båten vår ved å lage et imaginært rektangel rundt det aktuelle fartøyet og se vår båt gjøre visse aksjoner dersom den treffer noen av disse områdene. Siden vår båt er mye tregere enn andre båter på sjøen, må systemet ta hensyn til dette. Måten vi har løst det på er at om vår båt treffer siden på en av disse rektangelene vil den stoppe og vente på at fartøyet har passert. Om den treffer framsiden av rektangelet vil den kjøre til den siden som er nærmest og enten vente eller fortsette. Treffer vi baksiden vil båten velge den veien som innebærer minst kursendring, som da vil unngå venting. Denne algoritmen vil være enkel å regne ut om bord på båten, og vil prøve å unngå kollisjoner og mest mulig unødvendige omveier for å spare strøm.



Figur 1: Det imaginære rektangelet rundt et fartøy på sjøen

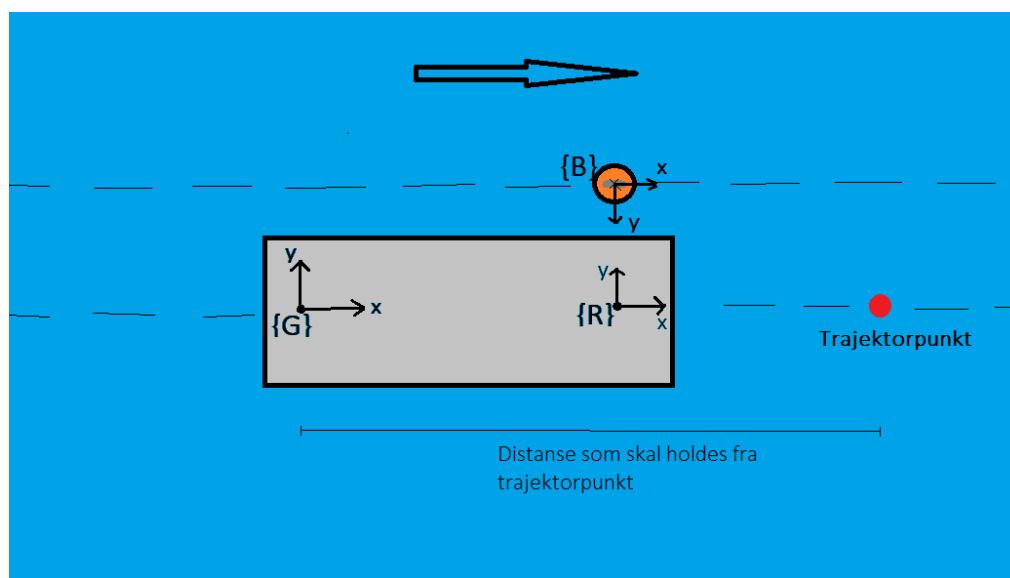
For mindre (ofte private) båter vil vår eneste mulighet til å detektere de være ved hjelp av kameraet ombord. Derfor vil vi legge inn en algoritme som gjør at dersom vår båt ser noe annet enn en bøye på vannet med kamera, vil den endre kurs til å kjøre i motsatt retning av objektet så fort som mulig. Denne algoritmen vil måtte overstyre all annen navigering for å prøve å holde båten trygg. I tillegg vil vi gjøre båten så synlig som mulig for andre båter for å unngå kollisjon. Dette kan gjøres ved bruk av høgkontrastfarger, lys og refleks.

En annen utfordring var at vi opplevde flere problem med motion planning i punkt 3.i.1. Vi ønsker å bevege armen i joint-space for å unngå unødig slitasje på ledd samtidig som det er raskere enn Cartesian motion. All rettlinjett bevegelse vi trenger blir tvunget av prismatiske ledd. Vi ønsket å bruke funksjonen `jtraj` i `SerialLink`-klassen, men vi møtte på en del begrensinger; det originale demonstrasjons-skriptet lot oss ikke rotere armen fra babord til akterut uten programkrasj, da det horisontale prismatiske leddet trekker seg helt inn og krasjer i basen. Et annet problem er at vi ikke kan lage en tidsvektor der sluttiden er 1.x sekund, da opplever vi at et av de prismatiske leddene får en lengde  $< 0$  som fører til krasj. Vi fikk til å bruke de statiske (frittstående) metodene `jtraj` og `mtraj` (både med `@lspb` og `@tpoly`), men da mister vi tilgang til egenskaper som er lagret i `SerialLink` objektet

som f.eks. maksimal og minimal vinkel / lengde på leddene og mulighet til å legge inn tyngdekraft. For våre demonstrasjoner går det greit, men dette er en alvorlig begrensning som kan føre til at armen krasjer i seg selv under kjøring på ekte og at vi ikke kan simulere armens dynamikk.

### 3. Implementering

Det ferdige produktet skal være en liten autonom båt med en robotarm montert helt fremme på dekk. Den skal høre til en basestasjon ved kysten hvor operatører kan spore bøyen som skal plukkes opp. Kursen til bøyen kan regnes ut (vi antar den driver i en rett linje som følge av strømmen på 0,1m/s gitt i oppgaven) og videre kan et estimert møtepunkt mellom sensorbøyen og båten regnes ut, gitt en viss navigasjonsmetode for båten (dette blir forklart mer i detalj senere). Dette møtepunktet kan bli brukt som et statisk målpunkt, noe som tillater oss å bruke D\* som navigasjonsalgoritme ut til møtepunktet. Dette sikrer optimal rute og sparer tid og strøm. Siden D\* krever et utregnet kart med kostnadsverdiene, kan dette da bli gjort på datamaskiner på basestasjonen, ettersom dette er en regnekrevende prosess. Kostnadskartet kan lastes over på et SD-kort som kan settes i båten rett før oppdraget starter. Når båten kommer frem til møtepunktet vil den være på linjen sensorbøyen driver langs, og kan da starte sin trajektor-følgning uten at det vil være særlig med hindringer i veien. Når båten kommer frem til sensorbøyen vil den prøve å holde seg i posisjon for å plukke den opp. Dette oppnås ved at trajektoren den skal følge er et punkt som er forskjøvet fram og til siden for selve bøyen.

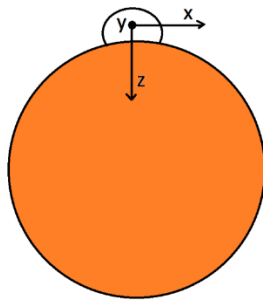


Figur 2: Forflyttet trajektorpunkt

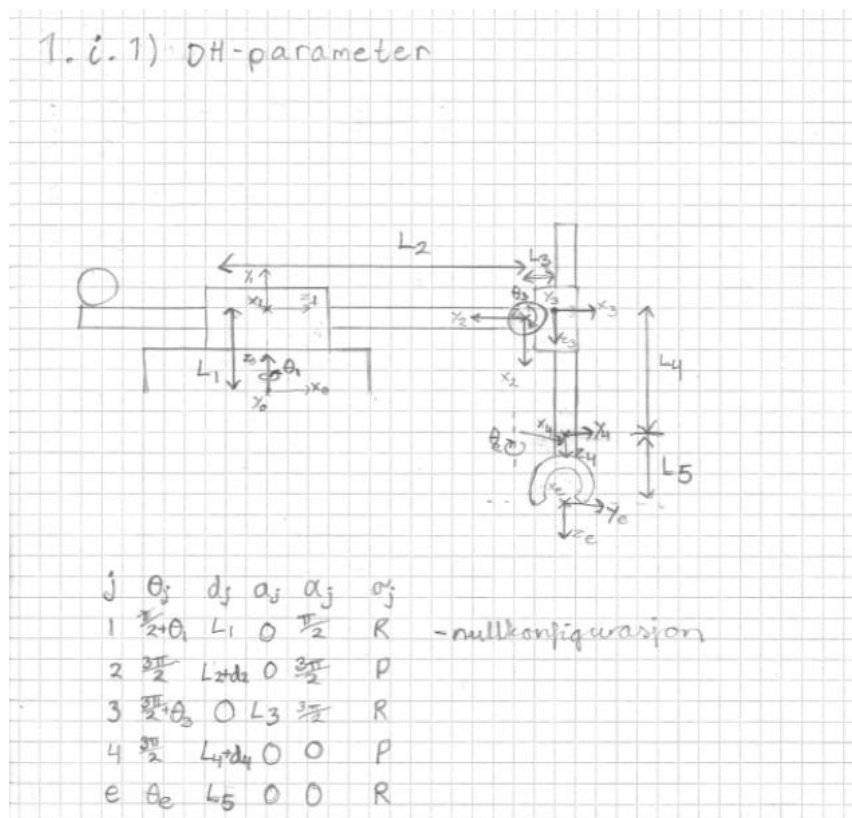
Ved å holde den innstilte avstanden fra dette punktet skal båten ha bøyen ganske så rett til siden for seg. Da er den klar til å plukke opp bøyen ved å lokalisere den med 3D-kameraet som er montert helt bak på båten på en liten mast for å få oversikt. Vi benytter inverse kinematics for å vite hvordan leddene til robotarmen må være innstilt for å komme til posituren som er inni bøyens håndtak. Referansesystemene til robotarmens griper og bøyen er designet slik at de to systemene kan overlappe i for eksempel referansesystemet til kameraet, og da vil robotarmen være i posisjon for å plukke opp bøyen, se Figur 2. Armen vil først navigere i et plan som garantert er over bøyen, og stille de to plankoordinatene til å samsvare med de tilsvarende koordinatene for bøyen.



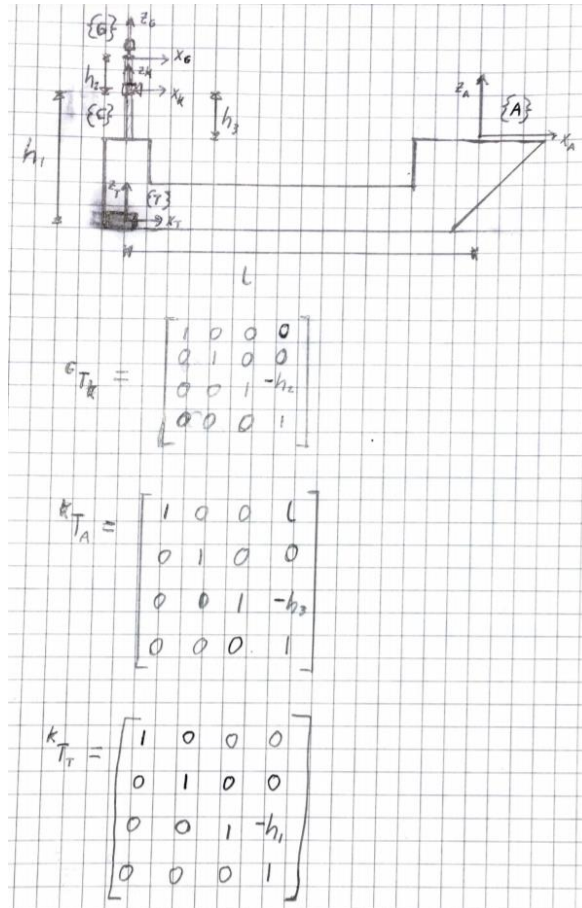
Når det punktet er nådd kan riktig orientering for griperen stilles inn og armen kan bevege seg ned for å gripe bøyen, slik at den siste koordinaten er lik bøyens.



Figur 3: Bøyens koordinatsystem



Figur 4: Robotarmens DH-parameter



Figur 5: De ulike referansesystemene på båten

På figur 5 kan man se en oversikt over referansesystemene på båten, og transformasjonsmatrisene fra GPS-modulen til kameraet, fra kameraet til robotarmens base og fra kameraet til midtpunktet mellom propellene.

2) Transformasjonsmatriser

$${}^0T_1 = \begin{bmatrix} -s\theta_1 & 0 & c\theta_1 & 0 \\ c\theta_1 & 0 & s\theta_1 & 0 \\ 0 & 1 & 0 & L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^1T_2 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & L_2 + d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$${}^0T_e = {}^0T_1 {}^1T_2 {}^2T_3 {}^3T_4 {}^4T_e$$

Figur 6: Transformasjonsmatrisene del 1

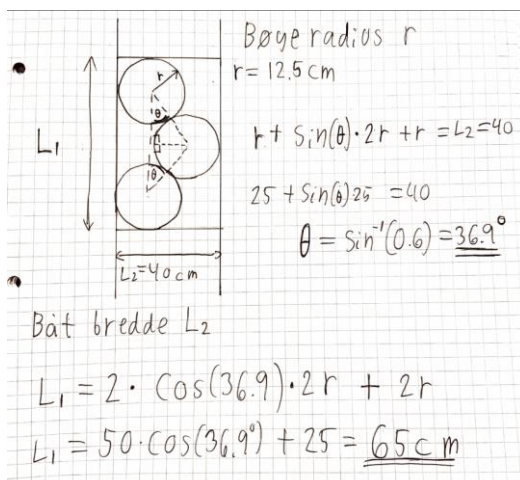
$${}^2T_3 = \begin{bmatrix} -s\theta_3 & 0 & c\theta_3 & -\theta_3 L_3 \\ c\theta_3 & 0 & s\theta_3 & -\theta_3 L_3 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^3T_4 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & L_4 + d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$${}^4T_e = \begin{bmatrix} c\theta_e & -s\theta_e & 0 & 0 \\ s\theta_e & c\theta_e & 0 & 0 \\ 0 & 0 & 1 & L_5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figur 7: Transformasjonsmatrisene del 2

På figur 6 og 7 over kan man se transformasjonsmatrisene fra robotarmens base til de ulike leddene og til end-effektoren. Sammen med transformasjonsmatrisen fra kameraet til robotarmens base ( ${}^K T_A$ ), kan  ${}^0 T_e$  (utregnet i Matlab 1.i.2) brukes til å beskrive end-effektoren i kameraets referansesystem, og som beskrevet er da målet å få end-effektorens system til å overlape med bøyens system, som også vil være uttrykt i kameraets system.  ${}^K T_e = {}^K T_A * {}^0 T_e$ , hvor A og 0 er samme system. Se [Lenke til GitHub repository](#) for Matlab-skript, video av arm og andre detaljer.

Dersom båten skal plassere ut en sensorbøye fra basestasjonen vil målpunktet være statisk og et kostnadskart kan regnes ut for hele veien ut, og båten vil kunne følge den samme ruten tilbake. Kostnadskartet vil bli regnet ut av datamaskiner på land og deretter lastet over på et SD-kort som settes i båten før oppdraget. Båten er designet for å kunne lagre opptil tre sensorbøyer på dekk i skålformede holdere. Da har vi antatt at hver bøye har en radius på 12,5 cm og har et håndtak på toppen slik at robotarmen kan løfte den. Tanken er at bøyen kan utstyres innvendig med ulike sensorer, batteri, GPS og annet utsyr, avhengig av formålet til det spesifikke oppdraget.



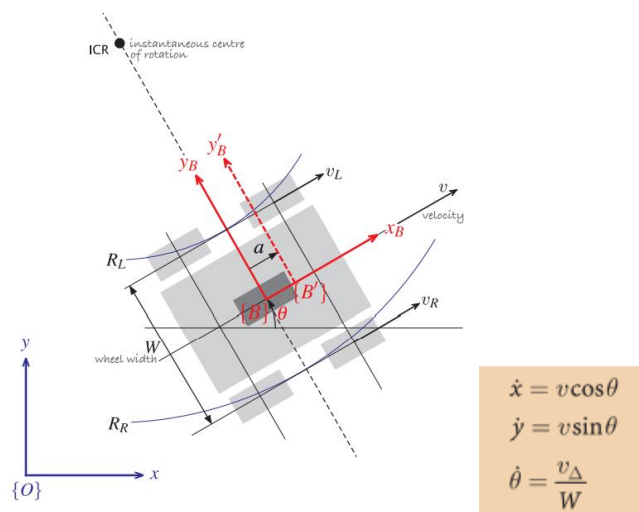
Figur 8: Utregning av bøyelagring



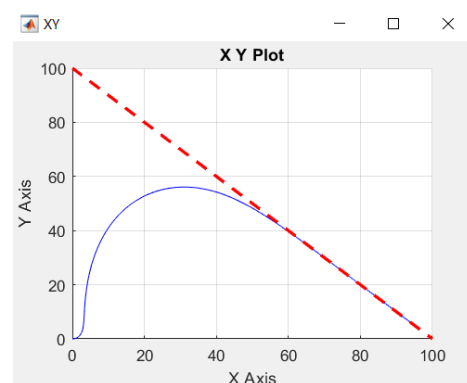
Figur 9: Bøyen

## Kinematisk modell

Siden vårt fartøy har to propeller bak vil det være differensialstyrt. Det vil si at båten vil svinge ved å ha forskjellig fremstøt på de to propellene. På denne måten vil båten kunne ha en svingradius på 0 og være nok så manøvrerbar. Båten vil ha ikke-holonomiske begrensninger fordi den vil ikke kunne akselerere i en vilkårlig retning uten å foreta en hjelpemanøver. En mobil robot som er differensialstyrt er beskrevet av en unicycle-modell. Unicycle-modellen bruker forskjellen i farten på de to hjulene, eller propeller i vårt tilfelle, for å modellere hvordan fartøyet vil svinge og bevege seg. De tre formlene som beskriver fartøyet bevegelse er vist på figur 10. Fartene i x og y-retning er beskrevet med gjennomsnittsfarten til propellene  $v$ , og vinkelen til kursen  $\theta$ . Vinkelakselerasjonen, altså endringsraten til kursen på fartøyet, er lik differansen mellom farten på de to propellene delt på  $W$ , distansen mellom propellene.



Figur 10: Unicycle modell og dens formler 2



Figur 11: Båten følger en trajektor

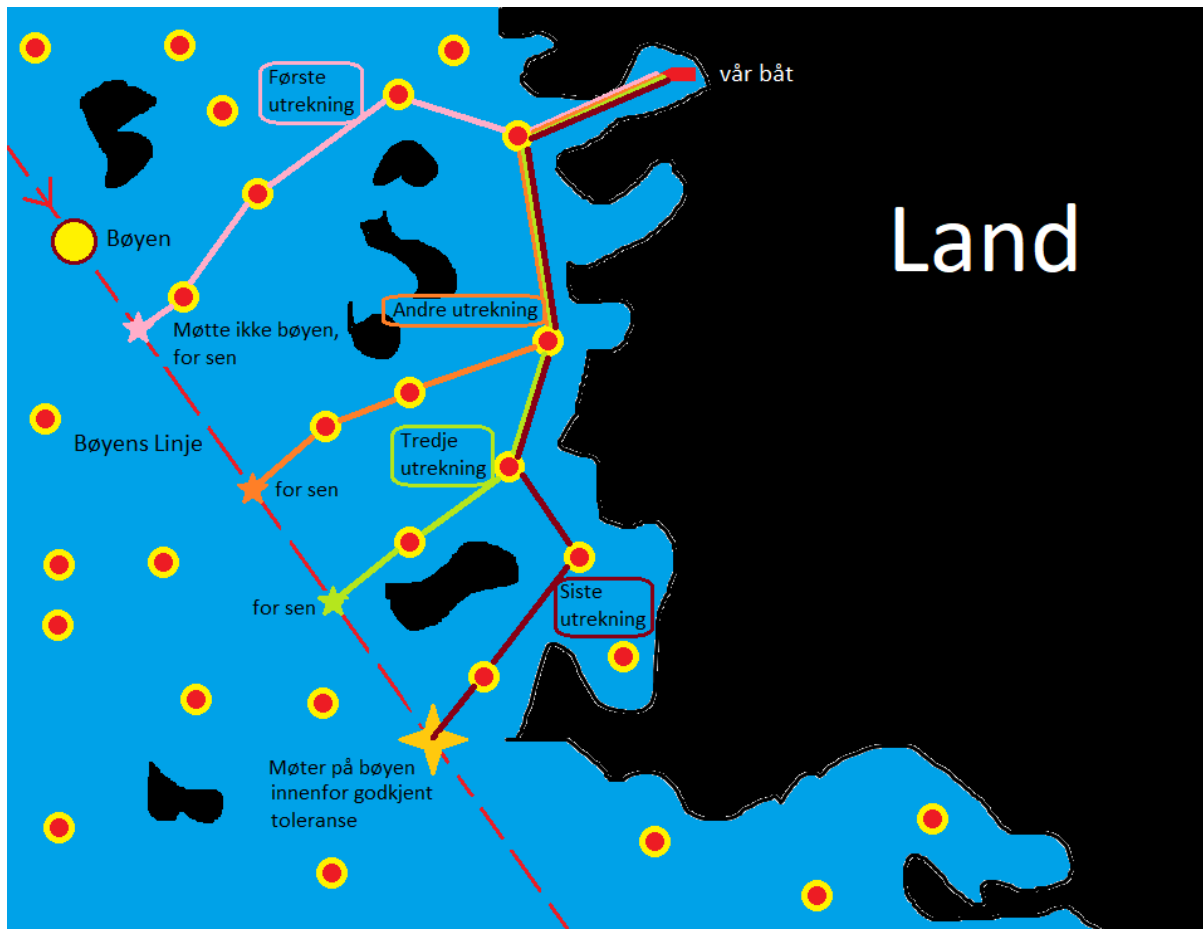
Ved hjelp av Simulink og ved å ta utgangspunkt i Peter Corke sin `sl_pursuit`-modell [3], har vi simulert vår båt som følger etter en trajektor. Siden bøyene vi skal plukke opp vil drive med strømmen (som i oppgaven er oppgitt som 0.1 m/s i en retning), vil bøynen bli simulert av en rettlinjert trajektor som båten skal prøve å følge. På figur 11 kan du se simuleringen. Bøynen begynner i (0,100) og beveger seg til (100,0) med en konstant hastighet på 0.1 m/s. Båten starter i (0,0) og begynner derfor å kjøre rett oppover mot bøynen. Etter hvert som bøynen beveger seg forbi båten vil båten begynne å svinge, for så å kjøre etter bøynen, ta den igjen og deretter kunne plukke den opp.

## Navigasjonsalgoritme

Båten skal (mesteparten av tiden) navigere ute på åpen sjø, hvor hindringer er få, store og statiske (f.eks. øyer). I tillegg vil båten vår være avhengig av å ha et lavt strømforbruk siden båten er liten, treg og vil bruke lang tid på oppdragene sine. Vi vil derfor ta i bruk en mest mulig effektiv navigasjonsalgoritme for å unngå unødvendige, ikke-optimale rutevalg. Med tanke på at båten skal bli sendt på oppdrag som har kjente mål og hindringene på veien vil være statiske, vil vi kunne benytte oss av en algoritme som er ressurskrevende og optimal. Dette kan vi tillate oss siden vi kan regne ut et passende kart i forveien for så å laste det opp på båten.

Vi har valgt å bruke D\* som vår hoved-navigasjonsalgoritme. D\* fungerer ved å regne ut den korteste veien til et bestemt punkt på kartet fra alle andre valgte punkt på kartet, og er en variasjon av distance transform. Nøyaktigheten til kartet kommer an på dimensjonen på området og hvor mange punkt en velger å bruke. Til våre formål vil kartet ikke trenge stor nøyaktighet ettersom sjøen som båten skal gå i er åpen og ruten vil bli svært god selv med lav oppløsning. Grunnen til at vi valgte D\* og ikke bare distance transform var for å gi oss muligheten til å kunne gi forskjellig farvann forskjellig kostnad, slik at båten kan unngå stormer og sjøområder med mye bølger, båter eller andre uønskede hindringer.

D\* vil gi svært gode ruter og er optimalt for utplassering av bøyer, men kan være en utfordring når båten skal hente bøyer på sjøen med tanke på at målet vårt driver med strømmen. Vår løsning på det er å regne ut et møtepunkt hvor båten forventer å treffe på bøyen og ta utgangspunkt i dette punktet for å regne ut et kostnadskart med D\*, se figur 13. Vi vil blåse opp størrelsen på holmer og annet land på D\*-kartet for å unngå kollisjon best mulig. Vi kan estimere møtepunktet ved å regne ut et simpelt probabilistic roadmap, og numerisk løse når båten og bøyen vil møtes (innenfor en viss toleranse), ved valg av den korteste ruten fra startpunktet (for eksempel med Dijkstras algoritme), se figur.



Figur 12: Estimering av møtepunkt ved hjelp av probabilistic roadmap

Denne løsningen vil spare tid og strøm over å prøve å kjøre rett møt bøyen ved hjelp av en trajektor eller lignende. Når båten har nådd møtepunktet vil den enten være foran eller bak bøyen på linjen bøyen følger. Da kan båten å følge denne linjen til den møter eller tar igjen bøyen. Når båten er nærme nok vil den ta i bruk trajektor for å komme i riktig posisjon for å kunne plukke opp bøyen.



Figur 13: Bruk av D\* til å nå møtepunktet

## 4. Eksperiment

Vår robot er ikke en fysisk konstruksjon og vi har derfor ikke kunne gjort noen fysiske eksperiment, men heller simuleringer i Matlab og Simulink. Vi har blant annet simulert armens bane for å plukke opp en bøye ved først å svinge ut og deretter bevege seg til en posisur gitt av kameraet. Denne simuleringen er gjort i Matlab og ligger under 2.iii.1 og 3.i.1. Annet enn problemene med mtraj og jtraj nevnt tidligere fungerte simuleringene som vi håpet og armen virker kapabel til oppgaven sin.

Vi brukte også en variasjon av Peter Corke sin `sl_pursuit`-modell (med unicycle) til å simulere båten som følger en bøye med konstant fart. Den fungerte perfekt til våre formål og simuleringen gav oss en god ide om hvordan båten vil oppføre seg når den skal prøve å kjøre til en bøye. Simulinkmodellen er lagt ved som en fil og resultatet er vist i figur 11.

## 5. Konklusjon

Vårt robotsystem oppfyller alle kravene fra oppgaven. Vi klarte å utvikle kinematikken til robotarmen vi valgte og å simulere at den gjør arbeidsoppgaven. Vi har valgt og utviklet en kinematisk modell og en navigasjonsalgoritme for den mobile basen som passer med oppgaven.

### Utfordringer ved implementering i den virkelige verden

I virkeligheten er det noen faktorer som kan påvirke det autonome systemet som for eksempel bølger, vær og vind. Det antas at sjøen er bølgefri i simuleringen, siden det gjør det enklere å designe kontroller for styring. En annen faktor er sensorbøyenes spesifikke utforming; de bør være sirkulære og ha radius på maks 12.5cm, slik at det blir nok plass til 3 bøyer i båten. Vekten til disse bøyene skal være under 4kg og det må være håndtak på toppen slik at robotarmen kan få tak i dem. Det autonome systemet vil ikke fungere med bøyer som ikke oppfyller disse spesifikasjonene.

En annen utfordring er batterikapasitet. Det er ikke bestemt en spesifikk type batteri eller størrelse. Båten må ha et batteri med stor nok kapasitet til at den får utført oppgaven. Siden båten er liten vil den kjøre i lav fart, som betyr at den vil være ute på sjøen lenge, opptil flere dager eller uker. Dette må undersøkes i en annen studie.

### Fordeler ved implementering i den virkelige verden

Det finnes en del positive sider ved implementasjon av dette autonome robotsystemet, for eksempel lave driftskostnader som lønn og drivstoff. Når det er et elektrisk autonomt system som kjører hele dagen og plukker opp sensorer sparer man på slike kostnader. Samtidig vil systemet være effektivt siden det kjører dag og natt, og kan lett transporteres og monteres av en person.

Det er verdt å merke seg at systemet er elektrisk og har et batteri som kan bli ladet opp av solceller, noe som bidrar til mindre bruk av fossilt drivstoff, som gjør det miljøvennlig.



## Lenke til GitHub repository

[https://github.com/SivertAamelfot/ELE306\\_GR12\\_O6](https://github.com/SivertAamelfot/ELE306_GR12_O6)

## Bibliografi

- [1] J. Butler, «<https://plugboats.com/>,» [Internett]. Available: <https://plugboats.com/meet-the-new-electric-boats-doing-plastics-clean-up/>. [Funnet 2020].
- [2] Cleanup, The Ocean, «[theoceancleanup.com](https://theoceancleanup.com/),» [Internett]. Available: <https://theoceancleanup.com/rivers/>. [Funnet 2020].
- [3] P. Corke, Robotics, Vision and Control, Springer, 2017.
- [4] P. Corke, «[PeterCorke.com](https://petercorke.com/),» 2020. [Internett]. Available: <https://petercorke.com/>. [Funnet November 2020].
- [5] J. McCarthy, «Globalcitizen,» [Internett]. Available: <https://www.globalcitizen.org/de/content/wasteshark-plastic-pollution-robot/>. [Funnet 2020].
- [6] B. Schiller, «[fastcompany.com](https://www.fastcompany.com/),» [Internett]. Available: <https://www.fastcompany.com/3028391/james-dyson-is-designing-a-giant-vacuum-on-a-boat-to-clean-ocean-trash>. [Funnet 2020].
- [7] NOAA, «[oceanservice.noaa.gov](https://oceanservice.noaa.gov/),» [Internett]. Available: <https://oceanservice.noaa.gov/facts/sea-surface-temperature.html>. [Funnet 20 11 2020].
- [8] e. a. K. S. Johnson, «ACS Publications,» [Internett]. Available: <https://pubs.acs.org/doi/pdf/10.1021/acs.analchem.5b04653>. [Funnet 20 11 2020].
- [9] S. A. W. Bureau, «[ocean-ops.org](https://www.ocean-ops.org/),» [Internett]. Available: <https://www.ocean-ops.org/dbcp/doc/buoyRecoveries/Buoy%20Recovery%20Techniques.pdf>. [Funnet 20 11 2020].

# Appendiks

## Appendiks A: Gantt – diagram

### GANTT DIAGRAM

|                | OPPGAVE  | PERSON                   | UKE 40 | UKE 41 | UKE 42 | UKE 43 | UKE 44 | UKE 45 | UKE 46 | UKE 47 |
|----------------|--|--------------------------|--------|--------|--------|--------|--------|--------|--------|--------|
| 1.i.1          | Utvikle DH-parametrene til robotarmen  | Alle                     |        |        |        |        |        |        |        |        |
| 1.i.2          | Utvikle transformasjonsmatriser for robotarmen   | Sivert                   |        |        |        |        |        |        |        |        |
| 1.ii.1         | Tegn modell av den mobile roboten med nødvendige variabler   | Alle                     |        |        |        |        |        |        |        |        |
| 1.ii.2, 1.ii.3 | Utvikle de kinematiske likningene for bevegelse av den mobile roboten                              | Alle                     |        |        |        |        |        |        |        |        |
| 1.iii.1        | Utvikle transformasjonsmatrisene fra valgte sensorsystem til relevante koordinatsystem til roboten | Trygve                   |        |        |        |        |        |        |        |        |
| 2.i.1          | Implementer robot i Robotics Toolbox og vis at denne modellen er lik modell uten RTB               | Sivert                   |        |        |        |        |        |        |        |        |
| 2.i.2          | Utvikle differential kinematics  | Sivert                   |        |        |        |        |        |        |        |        |
| 2.i.3          | Utvikle inverse kinematics   | Sivert                   |        |        |        |        |        |        |        |        |
| 2.i.4          | Demonstrer eksempel på motion planning   | Sivert og Trygve         |        |        |        |        |        |        |        |        |
| 2.ii.1         | Velg passende kontroller for den mobile roboten  | Alle                     |        |        |        |        |        |        |        |        |
| 2.ii.2         | Implementer den kinematiske modellen og kontroller til mobil robot i Matlab                        | Sivert                   |        |        |        |        |        |        |        |        |
| 2.iii.1        | Demonstrer bruk av robotens sensorsystem for å utføre relevante oppgaver                           | Sivert og Trygve         |        |        |        |        |        |        |        |        |
| 3.i.1          | Bruk motion planning til å bevege end-effector til krevd posisur                                   | Sivert                   |        |        |        |        |        |        |        |        |
| 3.ii.1         | Simuler valgte oppgave   | Sivert, Trygve og Robert |        |        |        |        |        |        |        |        |
| 3.ii.2         | Diskuter implementering av navigasjonstrategier  | Robert og Trygve         |        |        |        |        |        |        |        |        |
| 3.ii.3         | Diskuter implementering av lokaliseringstrategier  | Alle                     |        |        |        |        |        |        |        |        |
|                | PDR  | Alle                     |        |        |        |        |        |        |        |        |
|                | CDR  | Alle                     |        |        |        |        |        |        |        |        |
|                | Rapportskriving  | Trygve, Robert og Andres |        |        |        |        |        |        |        |        |
|                | Rettskriving av rapport  | Alle                     |        |        |        |        |        |        |        |        |

Signatur:  
 Robert Løland  
 Trygve Sognnes  
 Sivert Amelfot  
 Andres Rodriguez

## Appendiks B: MATLAB – skript

```
% ELE306 Semesteroppgåve Gruppe 12
%
% Dei to første seksjonane er det same som er gjort for hand, berre
% implementert i Matlab v.hj.a Corke Robotics Toolbox
% Programmet er meint å køyre ein og ein seksjon om gongen, bruk
% hurtigtasten (Ctrl + Enter) eller trykk "Run Section".

%%
% 1.i.1) Develop the table of DH-parameters
syms l1 l2 l3 l4 l5 th3
l1 = 0.2; l2 = 0.2; l3 = 0.05; l4 = 0.1; l5 = 0.1;

L1 = Link('d', l1, 'a', 0, 'alpha', pi/2, 'offset', pi/2)
L2 = Link('theta', 3*pi/2, 'a', 0, 'alpha', 3*pi/2, 'offset', l2)
L2.qlim = [0 0.6];
L3 = Link('d', 0, 'a', l3, 'alpha', 3*pi/2, 'offset', 3*pi/2)
L3.qlim = [-pi/2 pi/6]
L4 = Link('theta', 3*pi/2, 'a', 0, 'alpha', 0, 'offset', l4)
L4.qlim = [0 0.4];
L5 = Link('d', l5, 'a', 0, 'alpha', 0);

robotArm = SerialLink([L1, L2, L3, L4, L5], 'name', 'robotArm')

%%
% 1.i.2) Develop the transformation mapping the end-effector to base
%
% Skaler aksane så det er lettare å sjå
A = [-0.2 0.7 -0.2 0.7 -0.4 0.3];

% Definerer nødvendige varaibler symbolsk (for å sjå matriser utrekna)
% eller numerisk for å kunne plotte dei
syms l1 l2 l3 l4 l5 t1 t2 t3 t4 te
l1 = 0.2; t1 = 0;
l2 = 0.2; d2 = 0;
l3 = 0.05; t3 = 0;
l4 = 0.1; d4 = 0;
l5 = 0.1; te = 0;

% Transformasjonane mellom matrisene. T0 er basen.
T0 = [1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 1];

T01 = [-sin(t1) 0 cos(t1) 0; cos(t1) 0 sin(t1) 0; 0 1 0 l1; 0 0 0 1];

T12 = [ 0 0 1 0; -1 0 0 0; 0 -1 0 (l2 + d2); 0 0 0 1];

T23 = [-sin(t3) 0 cos(t3) -sin(t3)*l3; -cos(t3) 0 -sin(t3) -cos(t3)*l3; 0 -1 0 0; 0 0 0 1];

T34 = [0 1 0 0; -1 0 0 0; 0 0 1 (l4 + d4); 0 0 0 1];

T4e = [cos(te) -sin(te) 0 0; sin(te) cos(te) 0 0; 0 0 1 l5; 0 0 0 1];
```

```

% Reknar ut resultatet av matrisetransformasjon, slik at eg kan plotte dei
pos1 = T0 * T01;
pos2 = pos1 * T12;
pos3 = pos2 * T23;
pos4 = pos3 * T34;
pose = pos4 * T4e

% Teikner alle koordinatsystema og ser at dei stemmer overeins med det eg
% teikna for hand.
close all
trplot(T0, 'axis', A, 'length', 0.05);
hold on
trplot(pos1, 'length', 0.05)
trplot(pos2, 'length', 0.05)
trplot(pos3, 'length', 0.05)
trplot(pos4, 'length', 0.05)
trplot(pose, 'length', 0.05)

%%
% 1.ii.1,2 og 3 samt 1.iii.1 er gjort kun for hand.
%%
% 2.i.1) Demonstrate equivalence of forward kinematic solution obtained
% previously by hand
% Kjør 1.i.1 først, slik at robotArm objektet er definert.
q0 = [0 0 0 0 0];

%figure
T = robotArm.fkine(q0)
robotArm.teach(q0)

```

```

%%
% 2.i.2) Develop the differential kinematics and demonstrate how it could
% be used.
% Dersom armen står ut til eine sida på båten og båten har ulik fart
% i forhold til bøylene, kan vi kompensere for denne ulikheita som
% vist under. Armen vil då stå i ro i forhold til bøya, sjølv om båten
% køyrer fortare eller saktare enn bøya blir flytta av straumen.
% I eksempelet under køyrer båten raskare enn bøya, og arma må svinge
% bakover for å stå i ro i forhold til bøya.
% Vi ser at qd roterer basen i positiv retning (mot klokka sett ovanfrå) og
% det første prismatiske leddet køyrer bjelken utover.
% Vi ser også av xd at vi kun har fart i negativ x-retning.
% Eg har valt å utelukke siste rad i qd, då end-effector kun kan rotere
% rundt z-aksen, så det er ikkje vits å vise den.
qn = [pi/2 0.5 0 0.2 0]
J = robotArm.jacob0(qn)
qd = pinv(J) * [-0.2 0 0 0 0 0]';
qd(1:4,1)
xd = J*qd;
% J4dof = [J(1:4,:); J(6,:)]
% qd = inv(J4dof) * [0.2 0 0 0 0]' % sett inn verdi for ønska fart i end-effector
% xd = J4dof * qd;
xd'
robotArm.plot(qn)

%%
% 2.i.3) Develop the inverse kinematics and demonstrate how it could be
% used.
% Dersom vi ønsker at armen skal vere i ein bestemt positur,
% kan vi oppgi denne transformasjonsmatrisa og spørre kva leddvinklane /
% lengdene må vere for at vi skal oppnå dette ønsket.
% Sidan vi kun har 5 DOF må vi nytte ei maske, som seier at vi ikkje
% bryr oss om rotasjon rundt eller translasjon langs ein akse.
T = [ 1 0 0 0.5;
      0 0 1 0.5;
      0 -1 0 -0.15
      0 0 0 1]

fram = robotArm.ikine(T, 'mask', [1 1 1 0 1 1])
robotArm.teach(fram)

```

```

%%
% 2.i.4 ) Demonstrate example motion planning, on a task relevant to your
% robot design challenge
% Vi ønsker å rotere armen ut på eine sida av båten for å gjere seg klar
% til å plukke opp ei bøye. Vi har valt å gjere dette i joint-space, sidan
% det er raskare enn cartesian motion, og vi tar betre vare på ledda.
q0 = [0 0 0 0 0];
qn = [pi/2 0.5 0 0.2 0];
startPose = robotArm.fkine(q0)
sluttPose = robotArm.fkine(qn)
t = [0:0.01:2];

punkt = robotArm.jtraj(startPose, sluttPose, t, 'mask', [1 1 1 1 0 1]);
robotArm.plot(punkt)

%%
% 2.ii 1 er diskutert i rapporten i del 3 punkt om kinematisk modell
% 2.ii.2 Vi har bytta ut Bicycle med Unicycle i trajectory control
% (sl_pursuit). Sjå 3.ii.1 nedst i denne fila for simulering.

%%
% 2.iii.1) Demonstrate using the sensory system to command the robot,
% according to the task chosen. That is, show the calculations necessary to
% make the sensory data (e.g. an apple detected at an arbitrary location
% from a static 3D camera) useful to the robot (e.g. calculate the joint
% angles to put the tool point of the end-effector at the apple's location).

% Skaler aksane så det er lettare å sjå
A = [ -0.5 2 -0.5 1.5 -1.5 1];

% Transformasjon frå kamera til basen av armen
l = 0.8; h3 = 0.2;
TCA = [1 0 0 l;
        0 1 0 0;
        0 0 1 -h3;
        0 0 0 1]

% Transformasjon frå kamera til bøya
% Vi har valgt ein vilkårleg vinkel og avstand i alle retninger (unnateke Z),
% men som armen kan nå. Vi tenker at avstanden i Z (høgda) vil vere
% relativt fornuftig, slik at bøya ligg på vassflata og båten ligg passeleg
% djupt i vatnet.
theta = 2.417;
TCB = [cos(theta) sin(theta) 0 0.692;
        sin(theta) -cos(theta) 0 0.43;
        0 0 -1 -0.441;
        0 0 0 1]

```

```

% Transformasjon frå base av arm til bøye
TAB = inv(TCA) * TCB

close all
figure('Name', 'Bøye og arm i forhold til kamera')
trplot(TCB, 'axis', A, 'color', 'red')
hold on
trplot(TCA, 'color', 'green')

figure('Name', 'Bøye i forhold til arm')
trplot(TAB, 'color', 'red')

% For å nå bøya må leddvinklane / lengdene vere:
qn = [pi/2 0.5 0 0.2 0];
robotArm.ikine(TAB, 'mask', [1 1 1 1 0 1], 'q0', qn)

t = [0:0.01:2];
startPose = robotArm.fkine([pi/2 0.5 0 0 0]);
% Her ønsker vi at armen går til ein positur som er rett over bøya i
% Z-aksen. Setter derfor translasjonsdelen i z-retning i TAB til å vere lik
% translasjonsdelen i z-retning i startPose.
midtPose = TAB;
midtPose(3,4) = startPose.t(3);
sluttPose = TAB;
% Vi kan då sjå vekk frå translasjon i z-retning, bøya kan aldri vere over
% høgda i startposisjonen qn.
punkt1 = robotArm.jtraj(startPose, midtPose, t, 'mask', [1 1 0 1 0 1]);
punkt2 = robotArm.jtraj(midtPose, sluttPose, t, 'mask', [1 1 1 1 0 1]);
robotArm.plot(punkt1, 'fps', 60)
robotArm.plot(punkt2, 'fps', 60)

```

```

%%
% 3.i.1) Use motion planning to move the robot end-effector through the
% required positions/orientations for the task chosen
% Arma peiker no akterut, men utfører same bevegelser som over.
t = [0:0.01:0.9];
q_start = [0 0 0 0 0];
q_venstre = [pi/2 0.5 0 0 0];
q_ned = [pi/2 0.5 0 0.4 0];
q_inn = [pi/2 0.1 0 0.4 0];
q_opp = [pi/2 0.1 0 0 0];
q_hoyre = [0 0.5 0 0 0];
q_ned2 = [0 0.5 0 0.2 0];
q_opp2 = [0 0.5 0 0 0];
q_tilbake = [0 0 0 0 0];

T_start = robotArm.fkine(q_start);
T_venstre = robotArm.fkine(q_venstre);
T_ned = robotArm.fkine(q_ned);
T_inn = robotArm.fkine(q_inn);
T_opp = robotArm.fkine(q_opp);
T_hoyre = robotArm.fkine(q_hoyre);
T_ned2 = robotArm.fkine(q_ned2);
T_opp2 = robotArm.fkine(q_opp2);
T_tilbake = robotArm.fkine(q_tilbake);

punkt1 = robotArm.jtraj(T_start, T_venstre, t, 'mask', [1 1 1 1 0 1]);
punkt2 = robotArm.jtraj(T_venstre, T_ned, t, 'mask', [1 1 1 1 0 1]);
punkt3 = robotArm.jtraj(T_ned, T_inn, t, 'mask', [1 1 1 1 0 1]);
punkt4 = robotArm.jtraj(T_inn, T_opp, t, 'mask', [1 1 1 1 0 1]);
punkt5 = robotArm.jtraj(T_opp, T_hoyre, t, 'mask', [1 1 1 1 0 1]);
punkt6 = robotArm.jtraj(T_hoyre, T_ned2, t, 'mask', [1 1 1 1 0 1]);
punkt7 = robotArm.jtraj(T_ned2, T_opp2, t, 'mask', [1 1 1 1 0 1]);
punkt8 = robotArm.jtraj(T_opp2, T_tilbake, t, 'mask', [1 1 1 1 0 1]);

robotArm.plot(punkt1, 'fps', 60)
robotArm.plot(punkt2, 'fps', 60)
robotArm.plot(punkt3, 'fps', 60)
robotArm.plot(punkt4, 'fps', 60)
robotArm.plot(punkt5, 'fps', 60)
robotArm.plot(punkt6, 'fps', 60)
robotArm.plot(punkt7, 'fps', 60)
robotArm.plot(punkt8, 'fps', 60)

```



```
%%  
% 3.ii.1) Simulate your chosen challenge, and discuss the simulation  
% results in terms of chosen control strategy and performance  
% Vi utfører kun simulering v.hj.a. Simulink her. Sjå rapporten for  
% diskjon. Vi antar at bøya reiser i ei rett linje med straumen.  
sl_pursuit_unicycle;
```

# Gruppe 1: Oppgave 6

Robert Løland, Trygve Sognnæs, Sivert Åmelfot og Andrés Rodríguez



## Valg av Oppgave

- Så på eksisterende design
- Arm er uegnet for innsamling av plast
- Bestemte oss for å endre oppgaven
- Innsamling av sensorer
- GPS for å finne sensorer

## Design av Arm

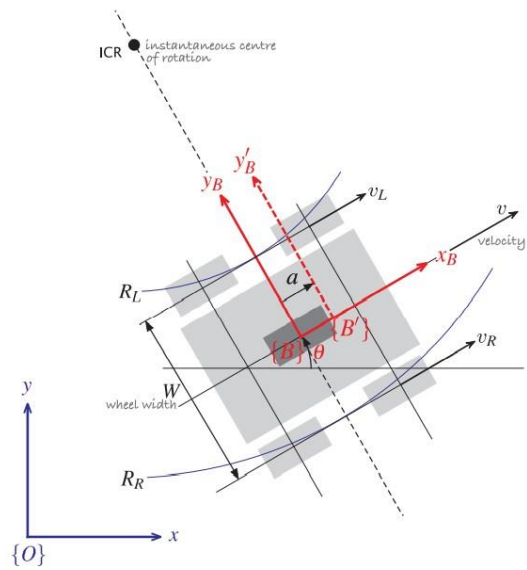
- Krav:
  - 60 cm horisontal rekkevidde,
  - 4 Dof
  - Kan løfte 4 kg
- Standard 6 Dof vs. winch-model
- 6 Dof: lettere
- Winch-model: realistisk



## End Effector og Kamera

- Krav:
  - Maks 4kg bøy
  - «handtak» til griper
  - 3d kamera med 10m rekkevidde
- Intel RealSense D435
  - FOV:  $86^{\circ} \times 57^{\circ} (\pm 3^{\circ})$
  - Høgbildehastighet (90 fps)
  - ca. 10m rekkevidde
  - Global shutter
- Bruke kameraet til å finne bøyens posisur





## Styring

- Krav:
  - $< 1$  m svingradius
  - $0.1$  m/s strøm
  - 2 bakre thrusterer, 30 cm mellom
- Differensialstyring
- Følgestrømmen
  - Bøye blir relativt sett i ro



## GRUPPE 12 CDR

Oppgave 6

Sivert Elias Åmelfot

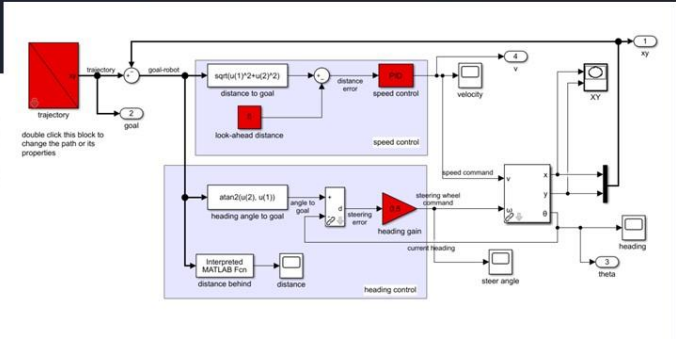


Robert Løland

Andres Rodriguez

Trygve Sognnæs

## NAVIGASJON

- Kinematisk modell - unicycle
- Navigasjonsalgoritme - D\*
- Navigering rundt båter



X Y Plot

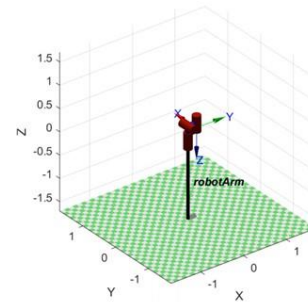
Y Axis

X Axis



# ROBOTARM-KINEMATIKK

- Invers kinematikk
- Differensiell kinematikk
- Jointspace



pose =

|   |   |    |    |
|---|---|----|----|
| 0 | 1 | 0  | 12 |
| 1 | 0 | 0  | 0  |
| 0 | 0 | -1 | -2 |
| 0 | 0 | 0  | 1  |

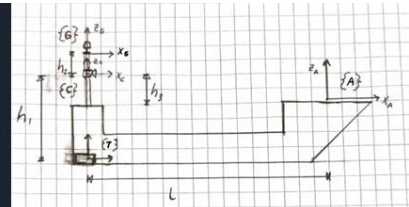
robotArm:: 5 axis, RPRPR, stdDH, slowRNE, Symbolic

| j | theta   | d  | a  | alpha   | offset  |
|---|---------|----|----|---------|---------|
| 1 | q1      | 11 | 0  | 1.5708  | 1.5708  |
| 2 | 4.71239 | q2 | 0  | 4.71239 | 0       |
| 3 | q3      | 0  | 13 | 4.71239 | 4.71239 |
| 4 | 4.71239 | q4 | 0  | 0       | 0       |
| 5 | q5      | 15 | 0  | 0       | 0       |

# DESIGN OG REFERANSESYSTEM

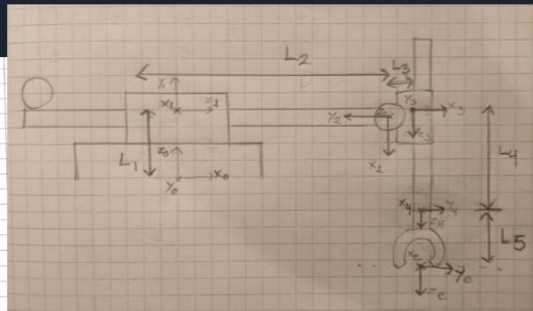
- Henting og utplassering
- Valg av robotarmdesign
- Båtdesign

Bøyene står i holdere



$${}^0T_c = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -h_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$${}^cT_A = \begin{bmatrix} 1 & 0 & 0 & L \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -h_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Bøyeradius  $r$   
 $r = 12.5 \text{ cm}$

$$r + \sin(\theta) \cdot 2r + r = L_2 = 40$$

$$25 + \sin(\theta) \cdot 25 = 40$$

$$\theta = \sin^{-1}(0.6) = 36.9^\circ$$

Båt bredde  $L_2$

$$L_1 = 2 \cdot \cos(36.9^\circ) \cdot 2r + 2r$$

$$L_1 = 50 \cdot \cos(36.9^\circ) + 25 = 65 \text{ cm}$$

## KONKLUSJON

- Utfordringer
  - Bølger*
  - Bøyer krever spesifikk konstruksjon*
  - Batterikapasitet -> solceller?*
  - Lav fart -> mye tid, vike unna båter*
- Fordeler
  - Lav driftskostnad*
  - Muliggjør mer forskning*
  - Jobber døgnet rundt*
  - Miljøvennlig*

