Høgskulen på Vestlandet

Bacheloroppgave

ELE350

Predefinert informasjon

Startdato:	08-05-2023 09:00 CEST	**Termin:**	2023 VÅR
Sluttdato:	22-05-2023 14:00 CEST	**Vurderingsform:**	Norsk 6-trinns skala (A-F)
Eksamensform:	Bacheloroppgave		
Flowkode:	203 ELE350 1 O 2023 VÅR		
Intern sensor:	Gizem Ates Venås		

Deltaker

Navn:	Mikael Walde
Kandidatnr.:	308
HVL-id:	590327@hvl.no

Informasjon fra deltaker

Egenerklæring *:	Ja
Inneholder besvarelsen konfidensielt materiale?:	Nei
Jeg bekrefter at jeg har registrert oppgavetittelen på norsk og engelsk i StudentWeb og vet at denne vil stå på vitnemålet mitt *:	Ja

Gruppe

Gruppenavn:	Enmannsgruppe
Gruppenummer:	7
Andre medlemmer i gruppen:	Deltakeren har innlevert i en enkeltmannsgruppe

Jeg godkjenner avtalen om publisering av bacheloroppgaven min *

Ja

Er bacheloroppgaven skrevet som del av et større forskningsprosjekt ved HVL? *

Ja, Cobot AGV

Høgskulen på Vestlandet

BACHELOROPPGAVE:

Autonom presisjonsparkering med AR

Autonomous precision parking with AR

Mikael Walde

Ingeniørfag - elektro, automatiseringsteknikk fakultet for ingeniør- og naturvitenskap/Institutt for datateknologi, elektroteknologi og realfag

22.05.2023

Veileder: Gizem Ates Venås

Kontakt person: Marcin Andrzej Fojcik

Dokumentkontroll

Rapportens tittel: BO23EF-03 Bacheloroppgave	*Dato* 22.05.2023
	Rapportnummer: BO23EF-03
Forfatter(e): Mikael Walde	*Studieretning:* Automatiseringsteknikk med robotikk
	Antall sider 47 + 13 Vedlegg
Høgskolens veileder: Gizem Ates Venås	*Gradering:* Åpen
Prosjekttittel: Autonom presisjonsparkering med AR	

Oppdragsgiver: HVL	*Oppdragsgivers referanse:*
Oppdragsgivers kontaktperson(er) (inklusiv kontaktinformasjon): *Marcin Andrzej Fojcik*	

Sammendrag

Denne bacheloroppgaven presenterer en autonom parkeringsløsning for Robotino, en differensialrobot produsert av Festo Didactic. Løsningen bruker AR-koder for å gi visuell informasjon til Robotino, slik at den kan parkere nøyaktig og presist på forhåndsdefinerte posisjoner. Målet er å montere en robotarm på toppen av en AGV for ulike produksjonsoppgaver. Kodeutviklingen fokuserer på struktur og tydelighet, slik at det er rom for videre utvikling.

Summary

This bachelor thesis presents an autonomous parking solution for Robotino, a differential robot produced by Festo Didactic. The solution utilizes AR tags to provide visual information to Robotino, enabling accurate and precise parking at predefined positions. The aim is to mount a robot arm on top of an AGV for various production tasks. The focus of the developed code is on structure and clarity, allowing room for future development.

Emneord

B023EF-03, Bacheloroppgave, Robotino, AR tag, automasjon, UR5-robot, navigasjon, 2023_Bachelor_Project_AGV_Docking.

Forord

Detter er hovedprosjektrapport fra bacheloroppgave BO23EF-03 " Autonom presisjonsparkering med AR" som ble skrevet av en student som gikk 6. semesteret på bachelor automasjonsingeniør ved Høgskolen på Vestlandet i Førde, vår 2023. Denne oppgaven er 20 av 180 studiepoeng som er hovedkravet for godkjent bachelorgrad og det forventes arbeid på ca. 500 timer

Oppgaver handler om å parkere AGV-en autonomt og teste presisjonen og nøyaktigheten på parkeringen.

Jeg vil takke Høgskulen på Vestlandet for en utfordrende og lærerik oppgave.

Takker Gizem Ates Venås for den fantastiske veiledningen og støtten jeg fikk under heile prosjekttiden. Hun har vist stor interesse for oppgaven, noe som har vært en god motivasjonsfaktor for meg.

Takker også Marcin Fojcik for en fantastisk mulighet for å reise til AIUT i Polen for å få muligheten til en faglig diskusjon med forskere rundt bacheloroppgaven. Denne turen ga meg nyttig erfaring og gode ideer til prosjektet.

Forkortelser og ordforklaringer

AGV	Automated Guided Vehicle
AR	Augmented Reality
RViz	Applikasjon for visualisering i ROS
CV	Computer Vision
Dwa_local_planner	Dynamic Window Approach local planner
Euler angle	En samling av tre vinkler som representerer orienteringen til en kropp i tredimensjonal rom.
FK	Forward Kinematics
Holonomitet	Egenskapen til en robot som kan svinge i alle retninger uten å måtte endre på orientasjon
IK	Inverse Kinematics
ISO	International Organisation for Standardization
LAN	Local Area Network
Move to point	Navigasjon av en robot til et predefinert punkt uten å tenke på orienteringen til en robot
Move to pose	Navigasjon av en robot til et predefinert punkt med et bestemt orientasjon til en robot
Node	En fil eller en programvare som er designet for å utføre en spesifikk oppgave.
Open CV	Open Source Computer Vision
Point Cloud	Samling av diskrete 3D-Koordinater som representerer punkter i et tredimensjonalt rom
Quaternion	Matematisk objekt som representerer rotasjonen som består av fire dimensjoner
REST_API	Representational State Transfer Application Programming Interface
ROS	Robot Operating System
TF	Transormation matrix
VS Code	Visual Studio Code

Tabeller

Figurer

Innhold

Sammendrag

Denne bacheloroppgaven presenterer en autonom parkeringsløsning for Robotino, en differensiell robot som ble produsert av Festo Didactic. Oppgaven er et initiativ fra det polske firmaet AIUT, som planlegger å montere en robotarm på toppen av en AGV og bruke denne i produksjon til diverse arbeidsoppgavene.

For å beregne riktig offset til robotarmen er det avgjørende at AGV-en som vi skal simulere med Robotino kan parkeres nøyaktig på forhåndsdefinerte posisjoner.

Løsningen bruker AR tags som gir Robotino visuell informasjon om parkeringsplassen og dens omgivelser, og gir Robotino mulighet til å parkere trygt, nøyaktig og presist ved hjelp av en navigasjonsstrategi.

Resultatet står jeg igjen med en mobil base som kan parkere autonomt og målingene som blir tatt etter at roboten har parkert.

Dette er ikke et ferdig produkt og av denne grunnen all koden som jeg kommer til å utvikle blir med fokus på struktur og oversiktlighet, slik at det er mulighet for videreutvikling.

II Innledning

1 Oppdragsgiver

Høgskulen på Vestlandet (HVL) campus Førde står bak dette Bachelor prosjektet. Den sammenslåtte Høyskolen på Vestlandet ble opprettet 01. januar 2017. Campus Førde lokalisert i Sogn og Fjordane og har ca. 300 ansatte for heile HVL. Høyskolen driver også med forsking innen områder som Datateknologi, Folkehelse, Bærekraftig energi og miljø og mye annet.

2 Problemstilling

En autonom AGV fra Figur 1 med påmonter robotarm skal parkeres med et visst nøyaktighet til en arbeidsbenk der robotarmen skal utføre arbeid. Her er nøyaktighet og presisjonen på parkeringen er veldig viktig siden denne kan påvirke nøyaktigheten og ytelsen til robotarmen.

Ideen er å finne og verifisere måte for en nøyaktig posisjonering av en AGV fra Figur 1 i forhold til arbeidsområdet. Det vil si når roboten har dokker må også nøyaktigheten testes.

Figur 1 AGV Formica 1. Eksempel på AGV. [1]

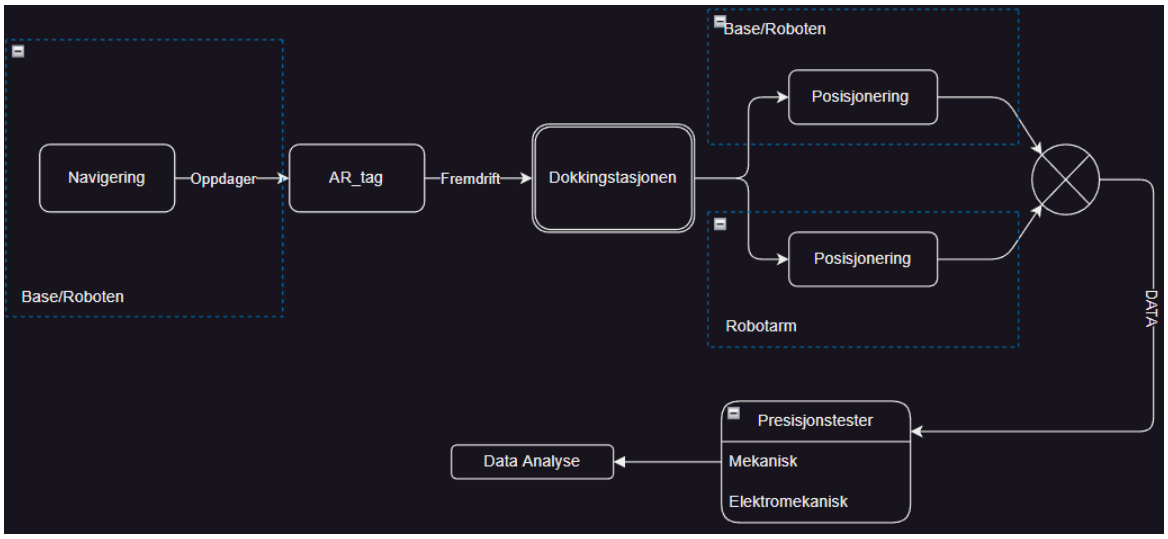
3 Kravspesifikasjon

3.1 Hovedmål

Målet med prosjektet er å demonstrere at roboten kan dokke til dokkingstasjon fra Figur 6 autonomt ved hjelp av et av teknologiene som blir valgt. Målet er også å vise nøyaktigheten for denne teknologien, altså hvor presis og nøyaktig klarer roboten å parkere.

3.2 Delmål

- Roboten skal navigere autonomt fram til dokkingstasjon.
- Basen må posisjonere seg mekanisk i forhold til dokkingstasjon.
- Robotarmen må posisjonere seg i forhold til arbeidsområdet.
- Presisjon og nøyaktighets testene skal utføres ved hjelp av robotarmen.



Figur 2 Sekvens til Prosjektet

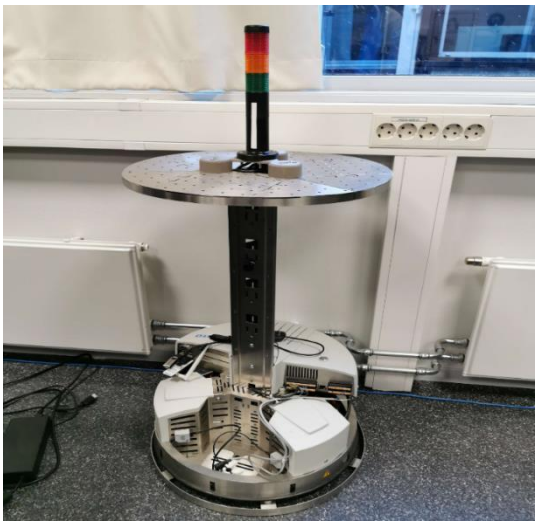
4 Analyse av problemet

4.1 Roboten skal navigere autonomt fram til dokkingstasjon

Basen skal søke etter AR-markøren fra Figur 5 og om den blir detektert vil kameraet lese av posisjonen og orientasjon til Ar-markøren, deretter vil ROS (Figur 7) beregne posisjonen til den mobile basen i forhold til destinasjonsmålet og flytte den til dokkestasjonen.

4.2 Mekanisk Posisjonering av basen

Om roboten har nådd målet vil vi utføre dokking, altså treffe dokkestasjonen med koblingen fra Figur 6 som er montert på den mobile basen. For å simulere en AGV blir det brukt Robotino i Figur 3 fra HVL labben i Førde.



Figur 3 Robotino

4.3 Posisjonering av Robotarm:

Når dokkingen er fullført, ønsker vi at robotarmen skal posisjonere seg i klarstilling for å gjøre seg klar til å plassere et tusjmerke på arbeidsområdet. Dette merket blir da brukt for å måle presisjonen og nøyaktigheten på parkeringen. Armen som blir brukt er UR5 robotarm fra Figur 4 som etter planen skal monteres på arbeidsområdet nær parkeringsplassen til Robotino.

Figur 4 UR 5 Robot arm til Universal Robotics. [2]

4.4 Presisjonstest:

Når roboten har dokket og robotarmen har posisjonert seg i klarstilling, plasserer den et merke, og deretter utføres en manuell test for å vurdere presisjonen og nøyaktigheten til teknologien.

5 Utforming av mulige løsninger

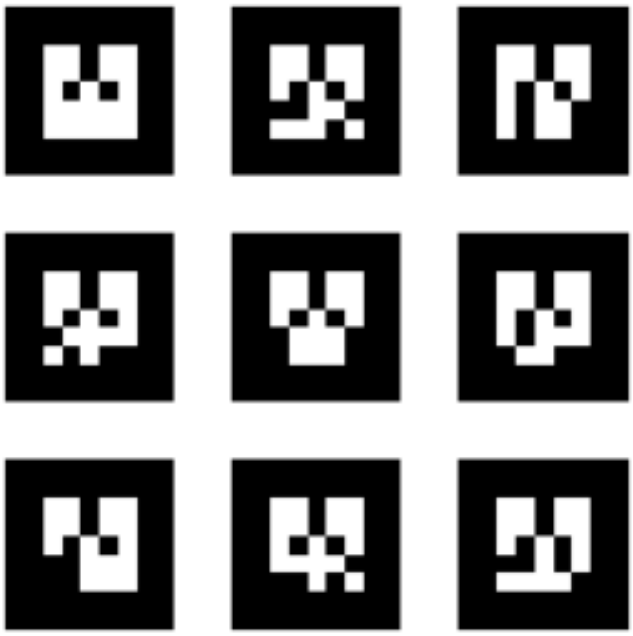
Det finnes mange løsninger på dette problemet. For eksempel:

- Dokking med AR-markører
- Mekanisk kobling
- Dokking med ultralydssensor
- Laser skanner (Lidar)

5.1 AR markører:

AR-markører fra Figur 5 er en markør som brukes ofte i oppgaver som er relaterte til CV. Ar-markører inneholder informasjonen som kan leses av med et kamera.

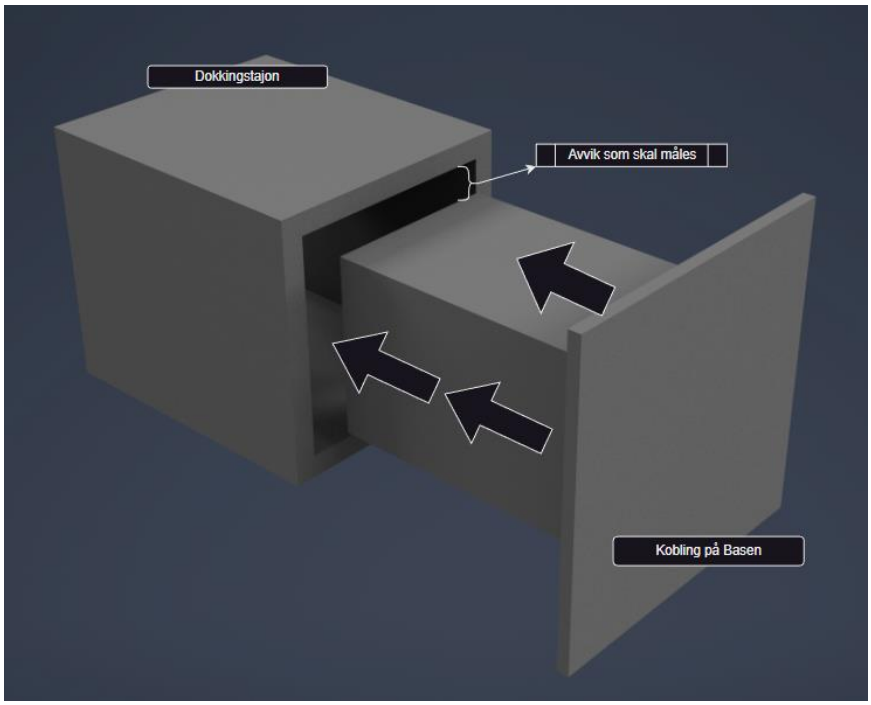
Typer informasjonen kan variere, men hovedsakelig er det markørens id/nummer, posisjonen og orientasjon som kan ytterligere brukes for å beregne nøyaktig hvor roboten er i forhold til markøren og mange andre ting.



Figur 5 Samling med AR-tags. Referansen [3]

5.2 Mekanisk Kobling:

Etter å ha navigert til dokkingstasjonen kan vi også bruke mekanisk kobling, eksempel på denne type dokking kan vare simpel låsemekanisme uten noen roterende deler som vil ytterligere øke presisjonen og nøyaktigheten. Dette er ikke hoveddel i prosjektet, men vi kommer med en simplifisert representasjon av denne løsningen fra Figur 6 som blir brukt for å gjøre det visuelt enklere å se om roboten er parkert.



Figur 6 Oppsett for mekanisk dokking 3D modell

5.3 Ultralydsensor

Ultralydsensor eller en akustisk sensor er en sensor som bruker lyd for å beregne hvor objektet er i forhold til sensoren. Slike sensorer sender ut en lydbølge som er også kalt for ekko, ved å måle tiden det tar for lyden er registrert med sensoren, kan avstanden til objektet beregnes med viss nøyaktighet. Dette er ei løsning som kan vare attraktiv for bachelorprosjektet.

5.4 Lidar

Lidar som kan også kalles for en laser skanner er en teknologi som brukes til å måle avstander til et objekt ved å sende ut lys strålene og måle tiden det tar får det reflekterte lyset kommer tilbake til sensoren. Slike sensorer blir brukt i ulike arbeidsområder, fra avstandsmåling og kartlegging til robotikk og autonome kjøretøy. Dette løsningsalternativet kan også bli brukt til å navigere den mobile plattformen til destinasjonsmålet.

5.5 Verktøyet som er nyttet

II.5.5.1 ROS

Fra tidligere semester ble vi godt kjente med ROS via et fag som ELE306-1 22H Robotikk. ROS er et utviklingsverktøy som tillater enkel kommunikasjon mellom Python eller C++ modulene via Publisher/Subscriber struktur. Det finnes flere versjoner av ROS som kan nyttes. For dette prosjektet blir det brukt en nyere Ros versjon melodic. Denne versjonen støttøtter mesteparten av pakkene som finnes.

Figur 7 Logo: ROS

II.5.5.2 Python

Jeg er personlig godt kjent med Python programmeringsspråket fra tidligere privat prosjekter og fag som ING301-1 Datateknologi og videregående programmering for ingeniører. ROS hovedsakelig bruker C++ og Python. Men min kjennskap til Python er på en mye høyere nivå i forhold til C++. Dette er hovedgrunnen til at jeg bruker dette programmeringsspråket.

Figur 8 Logo: Python

II.5.5.3 VS Code

Dette er den viktigste programvaren for prosjektet som vil bli brukt for utvikling, testing og visualisering av koden og modulene knyttet til prosjektet. Programvaren støtter ulike programmeringsspråk som Python og C++. Dette er de primære programmeringsspråkene som brukes av ROS. VS Code blir også brukt til å lage Flytdiagrammer med eksisterende visuelle utvidelser.

Figur 9 Logo: VS Code

II.5.5.4 Autodesk Inventor

Dette er programvare for utvikling av 3D modeller knyttet til Mekanisk dokking. Her blir de fleste mekaniske delene designet. Denne kan også brukes for stress, veske og strømningshastighet simulasjoner. Denne programvaren er jeg spesielt godt kjent med fra tidligere prosjekter på privaten, derfor synes jeg at denne programvaren passer perfekt til dette prosjektet.

Figur 10 Logo: Autodesk

II.5.5.5 Clickup

Clickup er brukt til loggføring, dirigering av arbeidsoppgaver. Programvaren kan og er brukt for loggføring av timene for prosjektet og arbeidsoppgavene.

Figur 11 Logo: Clickup

5.6 Valg av løsning

Vi valgte å bruke AR-markører i stedet for laserskannere eller andre teknologityper på grunn av flere fordeler som teknologien tilbyr. For det første gir AR-markører en høy grad av presisjon i posisjonering og måling. Denne teknologien er også vel kjent og har utviklet seg betydelig, og det finnes allerede gode og velkjente pakker som støtter denne typen teknologi. Dette er avgjørende for å oppnå nøyaktig navigering og plassering av ulike objekter. Videre er AR-markører enkle å bruke og iverksette, og de krever minimal teknisk kunnskap eller spesialutstyr sammenlignet med andre tilgjengelige teknologityper. Sammenlignet med alternative metoder som kan kreve spesialisert programvare og utstyr kan AR-markører opprettes med eksisterende programvare som kommer med markørens installasjonspakke. Når det gjelder deteksjon av slike merker, kan vanlige ekstern videokameraer brukes, og det kan kjøpes i de fleste elektronikkbutikker.

En annen fordel ved å bruke slikte tags er kostnadseffektivitet. Denne teknologien er relativt rimeligere i prisen sammenlignet med andre avanserte teknologier som for eksempel laserskanner. Med disse fordelene kan vi sei at denne løsningen passer best for vårt prosjekt.

Årsaken til at vi bruker en mekanisk kobling er at denne mekaniske dokkingen kan fungere som en låsemekanisme som ytterligere har muligheten til å øke presisjonen og nøyaktigheten ved parkering. Samtidig gjør det også enklere å visuelt bekrefte om roboten har parkert eller ikke. Slike mekaniske løsninger er kostnadseffektive både i design og produksjon, samtidig som de kan ha en betydelig innvirkning på presisjonen til robotens parkering.

III Teoretisk rammeverk

1 Autonom robotteknologi

Det finnes mange eksempler på mobiler roboter som kan parkere autonomt. I dag brukes slike systemer for å utføre en rekke oppgaver, blant annet autonom transport og logistikk, de kan også brukes til å parkere autonomt, som er ganske relevant til dette bachelorprosjektet.

Roboter av denne typen som for eksempel i Figur 12 er ofte utstyrt med avansert programvare og maskinvare som ofte avhenger av arbeidsoppgavene og miljøet roboten skal utføre arbeid i. Kombinasjon av disse tillater roboten bevege seg i et område autonomt, for eksempel kjøre fra punkt A til punkt B på en mest effektiv måte eller navigere langs en forhåndsdefinert bane.

Slike systemer har utviklet seg eksponentielt i den siste tiden der sensorer, styrings systemer og brukergrensesnittet blir stadig mer og mer avanserte. I dag Autonome systemer blir brukt i alt fra offentlig til privat sektor.

Med en teknologi som er i rask utvikling vil også etterspørselen øke. Det vil si at det kreves pålitelighet og effektivitet fra teknologien for å utføre automatiserte oppgaver på best mulig måte. Personlig tror jeg at Autonom robotteknologi kommer sannsynligvis til å spille stadig større rolle i fremtiden.

Figur 12 Autonom Robot-plattform til Posten. [4]

2 AR-markører og Open CV

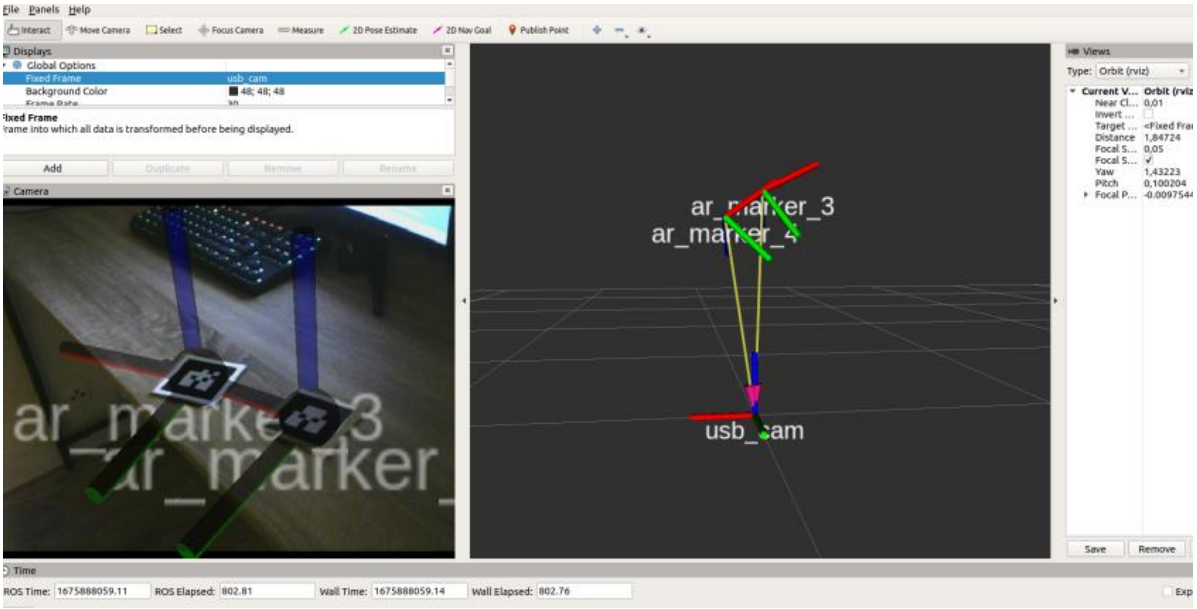
I dette kapittelet blir det introdusert og forklart hva AR-markørene og Open CV er og hvordan disse kan utnyttes.

2.1 Deteksjon av AR-tags

I dette prosjektet blir det brukt AR-markører for å navigere seg frem til en målstilling. For å få dette til må det først og fremst monteres en video kamera på roboten og iverksettes en programvare som detekterer markører med en deteksjonsalgoritme. I prosjektet blir det brukt en *Ar_track_alvar-* pakke.

Dette er en ROS basert pakke som tillater brukeren å detektere nøyaktig posisjonen og orientasjon til ar markørene i forhold til kameraet, med tilleggsinformasjonen som for eksempel navnet til taggen som blir detektert.

Pakken tillater også deteksjon av flere AR-markører samtidig og gir samme type informasjon om alle markørene. Når kameraet oppdager en markør, genererer programvaren en matematisk sammenheng mellom kameraet og markøren, som deretter kan brukes av den mobile basen. Denne sammenhengen kalles også en transformasjonsmatrise mellom markøren og kameraet. I Figur 13 kan vi se et eksempel på en vellykket test av AR_track_alvar-pakken, der transformasjonsmatrisene for kameraet og begge markørene visualiseres i Rviz.



Figur 13 TF relasjoner i Rviz med 2 AR tagger

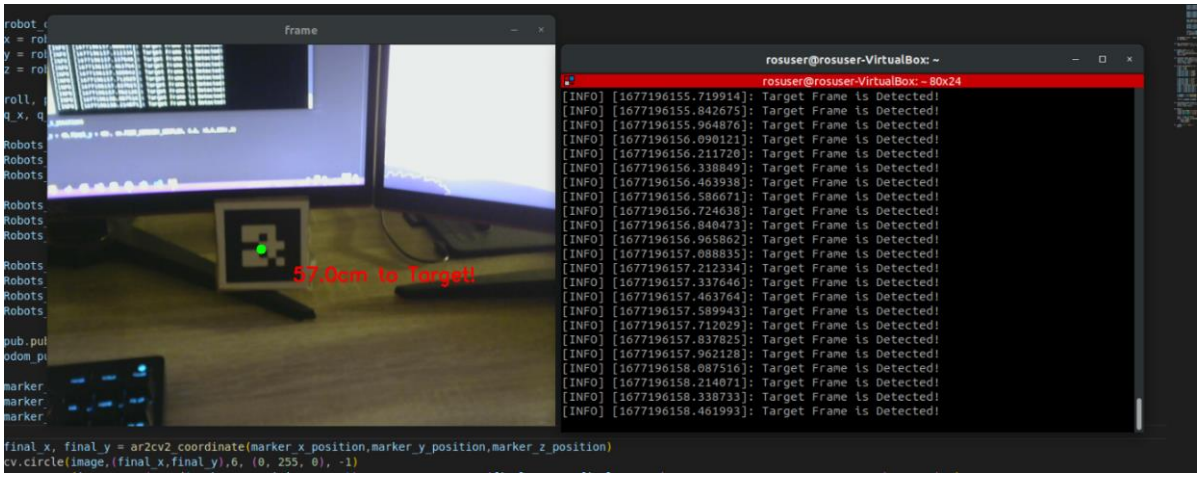
Med denne type informasjonen tilgjengelig kan vi lett beregne hvor roboten med påmonter kamera og markøren er i forhold til hverandre. Et grafisk eksempel på slik relasjon er vist i Figur 15.

Flere detaljer om denne pakken finnes under denne koblingen [5] og generelt mere om posisjonering med AR tags finnes under disse to referansene [6] og [7]

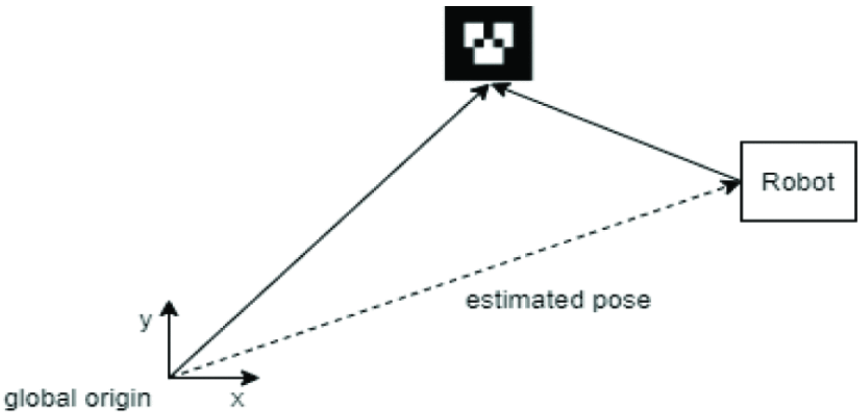
2.2 Introduksjon til Open CV

Open CV er en åpen kildekode bibliotek (pakke) som brukes i sammenheng med bildebehandling. Denne pakken inneholder viktig for oss funksjonalitet som tillater å kombinere bildebehandling med Python programmeringsspråket.

Ideen er å bruke Open CV til å visualisere nødvendig for oss informasjon om målstillingen. For eksempel visualisere hvor langt er roboten i fra destinasjonen, denne informasjonen vises rett over direktesendt videostrøm fra kameraet og et eksempel på dette kan observeres på Figur 14. Implementasjon av Open CV blir forklar i Kapittel (3.3 Utvikling og iverksetting av systemet). Også mere informasjon om Open CV kan finnes under denne referansen [8].



Figur 14 Eksempel på bruk av «Open CV» med «Ar tags» i ar_for_navigation.py modulen



Figur 15 Pose diagram av roboten og taggen i forhold til «global origin». [7]

3 ROS navigasjon

3.1 Hvorfor ROS?

For dette prosjektet er ROS en svært attraktiv programvare, dette er et åpent kildekodeprosjekt som har et stort samfunn av ROS utviklere, med mange ferdigutviklete applikasjoner som inkluderer simulering, 3D–visualisering, maskinlæring, robotikk og my mer.

ROS tillater meg å gjenbruke og kombinere eksisterende programvare innen navigering og AR visualisering for å bygge en kompleks og robust robotapplikasjon til bachelorprosjektet.

3.2 Valg av Navigasjonsmetode

Grunnet kompleksiteten på installasjon av enkelte pakker i ROS kommer vi til å se på to forskjellige måter å navigere roboten til parkeringsplassen. Der den første måten er ved å bruke Navigasjonsstakken til ROS og den andre måten er ved å designe en egen navigasjonsmodell uten å installere noen tilleggspakker. Hvilken metode som blir brukt skal vurderes underveis og blir diskutert under Kapittel (3.3 Navigasjonsalgoritmen).

III.3.2.1 Navigasjonsstakken til ROS

Navigasjonsstakken er en ferdiglaget ROS pakke med mange klare til bruk navigasjonsmodeller. Alle modellene fra stakken er ganske simple på et konseptuelt nivå. Det vil si at uansett hvilken modell jeg velger fra navigasjonsstakken vil den alltid bestå av tre grunnleggende elementer:

- Odometrien:
  - Dette gir robotens posisjon i forhold til verdenskoordinatsystemet. Denne informasjonen er avgjørende for at roboten skal kunne navigere i et miljø den befinner seg i.

- Destinasjonsmålet:
  - Denne angir hvor roboten skal. Det vil si at uten et klart mål for hvor roboten skal bevege seg til, vil navigasjonsmodellen ikke være i stand til å generere riktig utgang for roboten.

- cmd_velocity:
  - Denne kommandoen gir lineære hastigheter som navigasjonsmodellen sender til roboten. «Cmd_velocity» er i hovedsak det som styrer robotens hastigheter.

Selv om prosjektet mitt ikke fokuserer seg primært på navigasjon, kan jeg dra nytte av navigasjonsstakken for å enkelt iverksette modeller og oppnå pålitelige bevegelser.

Modellen som kommer til å bli brukt for å navigere Robotino heter «dwa_local_planner».
I faget ELE306-1 22H Robotikk ble vi kjent med TurtleBot3 og dens navigasjons pakke som bruker «dwa_local_planner» fro å navigere TurtleBot fra punk A til B.

Siden denne navigasjonsmodellen til TurtleBot er ikke robotavhengig, kan pakken gjenbrukes til å styre Robotino. Mere om implementasjon av denne modellen kommer i Kapittel (3.3 Navigasjonsalgoritmen). Tilleggsinformasjonen om «dwa_local_planner» og navigasjonsstakken finnes under denne referansen [9].

III.3.2.2 Egen navigasjonsmodell

Det andre alternativet er å lage en egen navigasjonsmodell som i prinsippet blir lik noen av modellene til navigasjonsstakken. Her blir også tre grunnleggende komponenter brukt, altså odometrien, cmd_velocity og destinasjonsmålet som ble forklart i forrige Kapittel (3.2.1 Navigasjonsstakken til ROS)

Å utvikle en slik modellen kan vare et alternativ til Navigasjonsstakken siden modellen kan designes med tanke på:

- Skreddersydd for spesifikke behov:
  - Det vil si at en egenlaget navigasjonsmodell, kan tilpasses kravene og behovene for et miljø roboten skal operere i.

- Bedre kontroll:
  - Ved å lage navigasjonsalgoritmen selv har jeg bedre forståelse og kontroll over hvordan roboten beveger seg fram til destinasjonsmålet eller reagerer på uforutsigbare situasjoner eller hindringer

- Tilpassingsevne:
  - Slik modell tillater oss å enkelt tilpasse eller videreutvikle algoritmen etter hvert som kravene eller behovene for oppgaven endres.

Det vil ikke si at det er enklere å implementere en egen navigasjonsalgoritme, siden denne krever en god teknisk ekspertise innen ting som: ROS, sensorikk, datavitenskap og robotikk. Og det kan også bli ressurs og tidskrevende.

For å simplifisere oppgaven kan en enkel «Move-to-point» algoritmen introduseres og om nødvendig videreutvikles til «Move-to-pose». Dette skal videre diskuteres under Kapittel (3.3 Navigasjonsalgoritmen)

4 Omnidireksjonale roboter

I dette kapittelet vil vi diskutere hva en omnidireksjonal robot er, samt dens egenskaper, fordeler og ulemper.

4.1 Definisjon og egenskaper

Omnidireksjonal robot er en robot som har en evne til å bevege seg i alle retninger uten å måtte tilsette lineære eller rotasjons hastigheter i andre retninger. Dette betyr at denne type roboten er en perfekt løsning for parkeringsoppgaver siden roboten kan parkeres uten å en gang å rotere på basen, takket være holonomiske egenskapene til hjulene som er montert under roboten. Figur 16 er et greit eksempel på en omnidireksjonal robot.

I oppgaven simulerer vi en AVG med en omnidireksjonal robot «Robotino» fra Figur 3. Grunnet er at vi ikke har noen AVG tilgjengelig på campus Førde.

4.2 Fordeler og ulemper

Slike roboter brukes i bredt spekter av bransjer med største bruksområder som industri og pakking. Det finnes mange fordeler og ulemper når vi snakker om roboter av denne type, men jeg skal bare nevne kun noen av dem.

Fordeler:

- Presisjon
    - Omnidireksjonale roboter har en veldig høy presisjonsgrad som tillater roboten å handtere komplekse oppgaver som krever nøyaktige bevegelser

- Effektivitet
    - Slike roboter kan oppnå høy hastighet på oppgavene siden der det er mulig kan vi unngå å svinge eller rotere roboten.

Ulemper:

- Kompleksitet
    - Kompleksiteten på maskinvaren og programvaren til slike roboter er vanskeligere å designe og bygge i forhold til andre ikke holonomiske roboter.

- Kostbarhet
    - Kostnaden for en omnidireksjonal robot vil naturligvis være mye høyere på grunn av kompleksiteten og den avanserte teknologiske løsningen.

Figur 16 Eksempel Omni Robot fra OSOYOO Mecanum Wheel Robotino. Referansen [10]

5 API

En API er en samling av abstrakte funksjoner og protokoller som tillater forskjellige programvare å kommunisere og samhandle med hverandre. API-en fungerer som et grensesnitt mellom ulike systemer, og den definerer hvordan informasjon kan utveksles og operasjoner kan utføres. I dag brukes API-er i en rekke forskjellige bruksområder og spiller en viktig rolle i moderne programvareutvikling. Enten det er innen webutvikling, apputvikling, skytjenester eller integrasjon av ulike systemer, muliggjør API-er enkel og effektiv kommunikasjon mellom forskjellige programvarekomponenter.

I dette prosjektet blir API-en brukt som et verktøy for å oppnå kommunikasjon mellom bærbare datamaskinen og ulike enheter som Robotino, Open Manipulator og eventuelt UR5-robotarmen. Gjennom API-en kan datamaskinen sende og motta data, instruksjoner og statusoppdateringer til og fra disse enhetene, noe som er essensielt for å kontrollere og samhandle med dem i prosjektet.

6 PID regulator

PID regulator er en matematisk algoritme som kan brukes til å regulere elektromekaniske maskiner og apparater. Denne typen regulatorer er også mye brukt innen robotikk, derfor kan implementering av en PID regulator være nyttig for dette bachelorprosjektet.

PID regulator består av 3 deler:

- P ledd (Proporsjonal ledd) - Dette er et ledd som bruker pådraget for å jevne ut avviket i systemet
- I ledd (Integral ledd) - Dette er et ledd som fjerner restavviket ved å ta integralet fra (0 til t) til avviket under skalverdien.
- D ledd (Derivative ledd) – D leddet har ansvar for å reagere på hastigheten avviket endrer seg over tid.

$$P = K_P \cdot e \qquad I = \frac{K_p}{T_i}\int_0^t e\,dt \qquad D = K_p \cdot T_d \cdot \frac{de}{dt}$$

$$U = P + I + D$$

Mer informasjon om PID finnes under denne referansen [11].

IV Utførelse

1 Hardware

I denne delen beskriver jeg komponentene til Robotino plattformen og annen maskinvare som kreves for navigering med AR tags. All tilgjengelig informasjon om komponenter og datablad eksisterer også under denne lenken [12].

1.1 Robotino komponenter

Hovedsakelig består Robotino av fem komponenter som:

- Styrings enhet
- Fremdriftssystem
- Sensorer
- Strøm forsyning
- Moduler

Styringsenhet:

En datamaskin som har hovedansvaret for alle kalkulasjoner, kontroll og utveksling av data.

Parameter	***Value***
Typ	*Integrerte PC til COM Express spesifikasjoner*
Operasjons system	*Linux Unbuntu 18.04 LTS (64 Bit)*
CPU	*Intel i5 8th generation. 4 cores*
RAM	*8 GB RAM*
HD	*64 GB SSD*
Motorstyring	*Mikrokontroller med 32 Bit-Mikroprosessor*

Tabell 1 Robotino spesifikasjoner

Fremdriftssystem:

Detter er et system som er ansvarlig for bevegelser relaterte posisjonering. Fremdriftssystem selv består av fem komponenter som jeg skal bare nevne. Om ønske finnes det mer detaljert informasjon her [12].

- Omnidireksjonale hjul
- Motorer
- Inkrementell enkoder
- Gir
- Framdrifts enheter

Sensorer:

Så langt er det kun kameraet som skal brukes. Det blir brukt et eksternt kamera som ikke hører med roboten. Om tiden tillater kan ekstra funksjonalitet iverksettes, for eksempel med bumper og avstandssensorer. Her er oversikt over alle sensorene som er tilgjengelige til den mobile basen.

- Bumper
- Avstandssensor
- Gyro
- RGBD Kamera (Denne følger ikke med.)
- Optiske sensorer
- Induktive sensorer

Strømforsyning:

Foreløpig er det uklart for hvilken strømforsyning blir brukt i Robotino, denne finnes ikke i databladet og likner ikke på det som tilbys fra leverandøren i utgangspunktet.

Moduler:

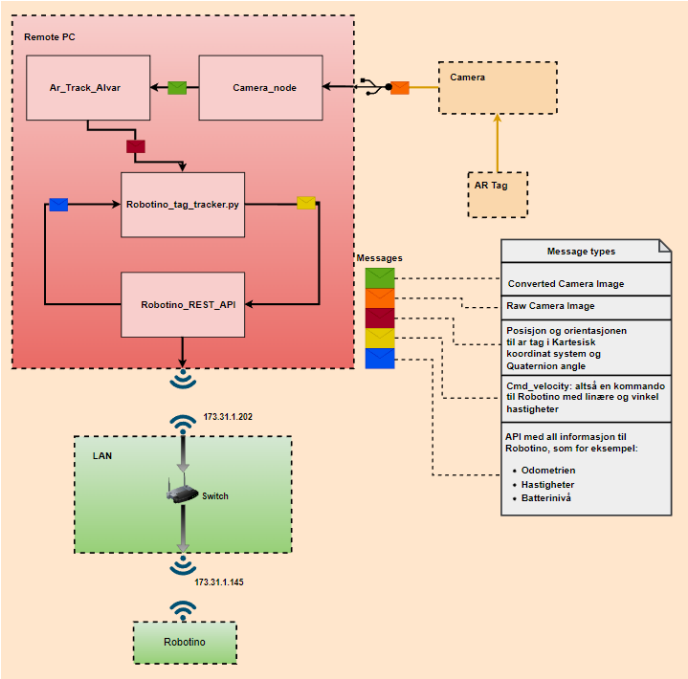
På robot plattformen er det montert en «Tower» modul som ser ut som et bord, der dette bordet kommer til å bli brukt for testing av presisjonen og nøyaktigheten. Mere informasjon om denne kan også finnes under denne referansen [12].

2 Software

I dette kapittelet forklares det hvordan kommunikasjonen ble satt opp mellom alle pakkene og nodene som ble brukt for å få Robotino parkert til dokkestasjonen.

Programvaren til prosjektet består av flere forskjellige noder som kan observeres på Figur 17 eller Vedlegg Kommunikasjonskart.

Alle nodene kommuniserer med hverandre via ROS Publisher/Subscriber strukturen. Dette er standard kommunikasjonsmåte mellom pakkene i ROS som tillater brukeren å bruke en Publisher-node til å sende data fra ei fil og en Subscriber-node som kan lytte på dataen som blir sendt.



Figur 17 Diagram med oversikten over alle modulene

1. Prosessen starter med kameraet som kobles til Remote PC med en USB-kabel.
2. Kamerabildet med ar-markøren inn i bildet plukkes deretter opp av Camera_node, som har i oppgave å konvertere bildet til det formatet ROS støtter.
3. Det konverterte bildet plukkes deretter opp av en Ar_Track_Alvar node som behandler informasjonen fra kameraet, beregner posisjonen og orientasjon til Ar markøren i forhold til kameraet
4. Den behandlede informasjonen pakkes og sendes videre som et objekt med innhold som: Markørens id, navn, kartesisk xyz-posisjon og orientasjon til markøren i Quaternion. Eksempel på et slikt objekt finnes i Figur 19.
5. Objektet som ar_track_alvar-pakken gir fra seg, tas deretter imot av Robotino_tag_tracker.py modulen. Denne Python-modulen er ansvarlig for navigasjon og styring av Robotino. Modulen beregner hvor Robotino er i forhold til destinasjonsmålet og navigerer roboten frem ved å sende en cmd_velocity bevegelseskommando.
6. Denne kommandoen plukkes videre opp av Robotino via Robotino_REST_API pakken, som kommuniserer med Robotino via et LAN-nettverk.

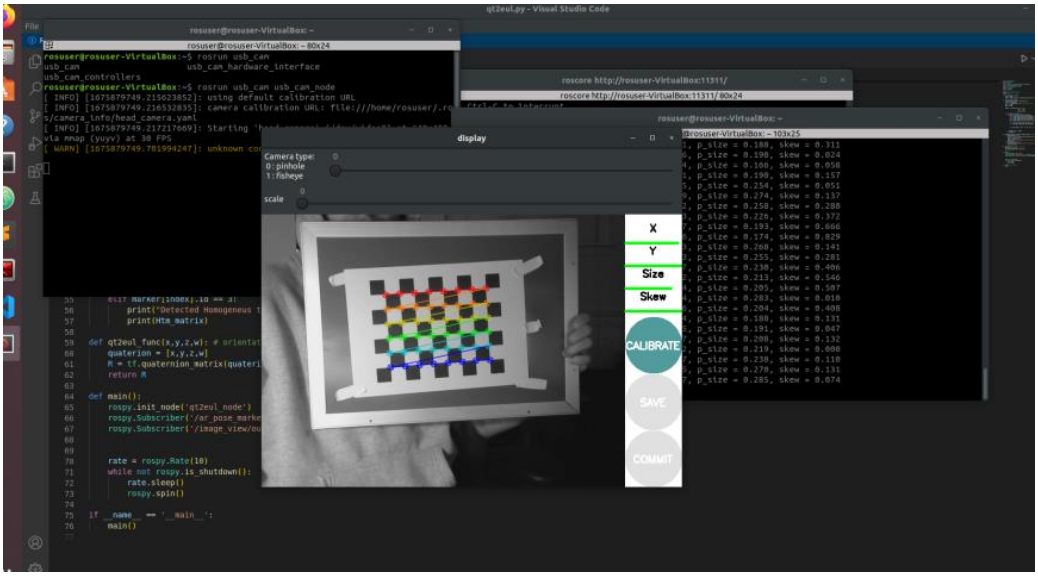
Mer om utviklingen og implementeringen av AR_track_alvar pakken og Robotino_tag_tracker.py python modulen blir forklart i Kapittel (3.0 Utvikling og iverksetting av systemet)

3 Utvikling og iverksetting av systemet

Dette kapittelet handler om utviklingen og implementeringen av ting som AR-markører, Open CV, Navigasjonsalgoritmen, mekanisk dokking og UR5 robotarmen.

3.1 Ar-markører

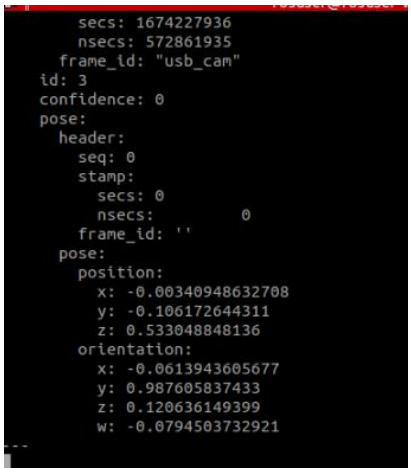
For at ar_track_alvar pakken skulle klare å lese av nøyaktig posisjonen og orientasjon til ar-markøren, må kameraet kalibreres med hensyn til optisk forvrengning. Om ikke denne operasjonen utføres vil alvar pakken ikke kunne lese av riktig posisjonen til markøren. Eksempel på kalibreringsprosessen kan observeres på Figur 18. Og mere informasjon om selve kalibreringsprosessen kan finnes under min Github link [13].



Figur 18 Kalibrering av kameraet

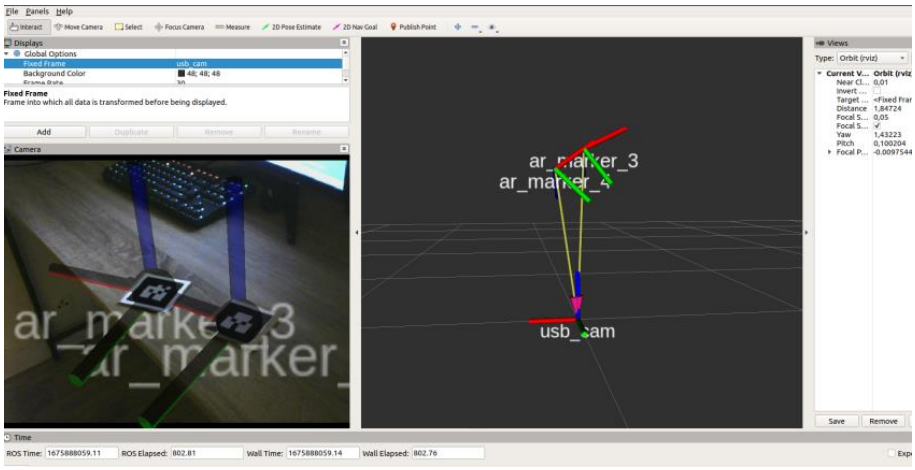
Når kalibreringen er utført vil Alvar-pakken automatisk bruke denne dataen til å konvertere rå bildeformatet fra kameraet hver gang samme type kamera blir koblet til maskinen. Får å teste om pakken leverer riktig posisjon blir det kjørt en manuell kommando via terminalen som lytter på objektet fra Alvar-pakken.

Ved å se på z-koordinater under posisjon fra Figur 19 , kan vi få den faktiske avstanden i meter mellom kameraet og senterpunktet på markøren. Ved å sammenligne disse verdiene med en fysisk måling av avstanden mellom kameraet og markøren, kan vi avgjøre om Alvar-pakken gir oss riktig avstandsinformasjon.



Figur 19 Objektet som blir sendt fra Alvar pakken

Tester også orienteringen til markøren ved å visualisere transformasjonsmatrisene i RViz som blir også vist på Figur 20. Her tester jeg flere Ar-markører siden Alvar-pakken er i stand til å potensielt handtere uendelig antall markører i samme kamerabildetplan.



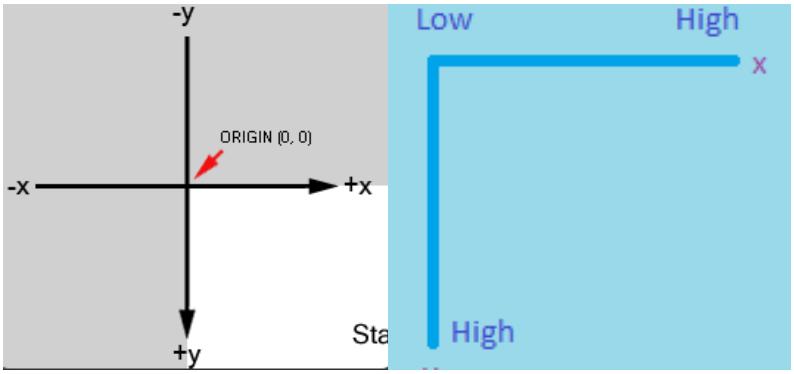
Figur 20 Test i Arviz

Etter å ha kjørt 5 kalibreringsforsøk med kameraet, konkluderte jeg at posisjonen til ar markøren blir gitt med ± 1mm som er godt nokk for denne oppgaven.

3.2 Open CV

Starter med å implementere et singelt punkt som vil alltid være i senteret til Ar markøren.

For å løse denne oppgaven ble det utviklet et Python skript som konverterer fra det 3D inverse kartesiske koordinatsystemet til Ar_track_alvar pakken i Figur 21 til 2D Open CV-koordinatsystemet som bruker fjerde kvadrant i et omvendt kartesisk koordinatsystem fra samme Figur 21



Figur 21 (Venstre)-Inverse Kartesiske og (Høyre)-Kartesisk fjerde kvadrant

Oversikt over metodene som ble implementert for å få denne oppgaven til er:

- map_coordinate ()
- ar2cv_coordinate ()
- calculate_scale_factor ()

```
def ar2cv_coordinate(marker_x_position, marker_y_position, z_distance):
    """
    The function takes the marker's x and y position in the AR coordinate
    System and the distance from
    the camera to the marker, and returns the marker's x and y position in the
    Open CV coordinate system

    :param marker_x_position: The x position of the marker in the AR
    coordinate system
    :param marker_y_position: The y position of the marker in the AR
    coordinate system
    :param z_distance: The distance from the camera to the marker
    :return: The x and y coordinates of the marker in the image.
    """
    common = Common # Creating an instance of the Common class




    scale_factor =
                   calculate_scale_factor(z_distance,common.z_min,common.z_max)
    min_values = [common.negX_at_z_min + (common.negX_at_z_max -
common.negX_at_z_min),
                  common.posX_at_z_min + (common.posX_at_z_max -
common.posX_at_z_min),
                  common.negY_at_z_min + (common.negX_at_z_max -
common.negY_at_z_min),
                  common.posY_at_z_min + (common.posY_at_z_max -
common.posY_at_z_min)]


    x_min = min_values[0] * scale_factor
    x_max = min_values[1] * scale_factor
    y_min = min_values[2] * scale_factor
    y_max = min_values[3] * scale_factor




    x_mapped = int(round(map_coordinate(marker_x_position, x_min, x_max, 0,
    640)))
    y_mapped = int(round(map_coordinate(marker_y_position, y_min, y_max, 0,
    480)))

    return x_mapped,y_mapped
```

Hovedmetoden, **ar2cv_coordinate** fra øvrige koden integrerer de to tidligere nevnte metodene og brukes til å finne det nøyaktige senterpunktet til AR-markøren.

Måten metoden fugerer på er at den tar først hensyn til de negative og positive grenseverdiene for x og y aksene basert på de minimale og maksimale avstandene fra kameraet til AR-markøren. Disse verdiene blir manuelt satt ved å flytte AR-markøren fysisk og logge verdiene som genereres av AR Track-Alvar-pakken. Her det også viktig å merke seg at verdiene som oppgis her kan variere avhengig av kameratypen som brukes og oppløsningen til kameraet, verdiene finnes også under klassen Common().

Deretter beregner metoden skaleringsverdien for z-aksen, som representerer avstanden mellom kameraet og markøren. Når skaleringsfaktoren er beregnet, multipliseres de positive og negative verdiene for x og y aksene i den minimale avstanden mellom kameraet og markøren med skaleringsfaktoren. Dette resulterer i nye grenseverdier for x og y aksene, som deretter kan brukes til å mappe x og y koordinatene til Open CV koordinatsystemet med en standar mapp() funksjon [14].

Når vi har fått det nøyaktige senterpunktet i Open CV-koordinatsystemet, kan vi faktisk bruke Open CV til å visualisere dette punktet på skjermen.

```
final_x, final_y =
ar2cv_coordinate(marker_x_position,marker_y_position,distance_to_marker)
                    cv.circle(image,(final_x,final_y),6, (0, 255, 0), -1)
```

I den øvrige koden kan man se hvordan metoden er brukt, og det vises også at jeg bruker et sirkel metode fra Open CV pakken med radius -1. Dette skyldes at Open CV tolker en negativ radius som en indikasjon på å tegne et punkt i stedet for en sirkel. Derfor vil den endelige figuren se ut som et enkelt punkt på skjermen, noe som er egnet for å representere senterpunktet til AR markøren.

Når det gjelder visualisering av teksten, kan dette enkelt løses ved å bruke metoden **cv.text()** fra Open CV-pakken. Denne metoden lar deg skrive ut tekst på skjermen og gir deg muligheten til å tilpasse skriftstørrelse, farge, posisjon og andre tekstegenskapene.

3.3 Navigasjonsalgoritmen

For å avgjøre om vi skal bruke Navigasjonsstakken til ROS eller lage vår egen navigasjonsalgoritme, må begge alternativene testes.

Etter å ha testet Navigasjonsstakken til ROS i noen uker, konkluderte jeg med at det var veldig vanskelig å iverksette «dwa_local_planner» fra TurtleBot 3 til Robotino. Siden navigasjonspakken krever visse ting som ikke er så enkle å implementere, for eksempel et kart og en Lidar som Robotino ikke har.

For å danne kartet, må vi skanne arbeidsområdet der Robotino skal operere ved hjelp av en Lidar, vi har en Lidar disponibellt som er integrert i på en av Turtlebotene. Dette også betyr at området må være statisk og aldri endre seg. Lidar i seg selv er nødvendig grunnet navigasjonspakken bruker PointCloud til å navigere med SLAM. Uten Lidar er det teoretisk mulig, men veldig vanskelig å få dette til i praksis.

Etter å ha laget et kart ved å skanne området med en av Turtlebotene vi har tilgjengelige på robotlabben og testet dwa_local_planner uten SLAM, konkluderte jeg med at dette var noe jeg ikke var i stand til å løse. Derfor ser alternativ nummer to mer attraktivt ut for meg i dette scenariet.

For å komme i gang med prosjektet, kan vi begynne med å utvikle en enkel algoritme som styrer roboten mot destinasjonsmålet, i dette tilfellet AR markøren ved å gradvis redusere avstanden. Dette vil tillate meg å starte testingen av robotens bevegelse og gi oss en plattform å bygge videre på.

I koden nedenfor vises hvordan jeg iverksatte dette. Først tester jeg om AR-markøren er identifisert og om avstanden til markøren er stor nok. Hvis vi oppfyller disse kriteriene, vil roboten bevege seg mot markøren og enkelt snu inntil avstanden er mindre enn den maksimale tillatte avstanden.

Her blir det også brukt en P-regulator for å kompensere for avviket i vinkelen mellom Roboten og destinasjonsmålet der selve vinkelen blir beregnet med formelen under.

$$\emptyset = \text{atan } 2(x, y)$$

Når robotens avstand er mindre eller er lik kalibreringsavstanden vil roboten avslutte denne fasen.

```
if marker_is_detected == True and distance_to_marker >
MAX_CALIBRATING_DISTANCE:
                twist = Twist()
                twist.linear.x = MAX_LINEAR_VEL
                twist.angular.z = -0.6 * math.atan2(marker_x_position,
distance_to_marker)
                move_cmd.publish(twist)
```

Neste som blir utført av programmet er kalibrering av orientasjon til roboten.

I koden nedenfor vises hvordan jeg iverksatte dette. Programmet sjekker om orientering ble kalibrer eller ikke, sjekker også om kalibreringen av avstanden er utført og om markøren er synlig for kameraet. Om kriteriene er oppfylt vil jeg utnytte holonomitet til Robotino og kjøre roboten sidelengs inn til posisjonen av ar-markøren langs x-aksen på skjermen er i origoen, altså lik 0.

For å beregne rotasjonen til roboten/kameraet, må jeg invertere kvaternionorienteringen til AR-markøren. Dette gjøres for å oppnå orienteringen i kvaternion til kameraet. Når denne inverteringen er utført, kan jeg endelig konvertere kvaternionen til Euler-vinkler og deretter ekstrahere denne

rotasjonen. Slik rotasjonen er også kalt for «yaw» rotasjonen, som er vanligvis rotasjonen rundt z-aksen.

Denne rotasjonen blir brukt i koden under til å kjøre roboten rundt z aksen inntil rotasjonen er lik 0. Altså vi kjører roboten rundt inntil orienteringen til den mobile basen blir lik orienteringen til ar markøren.

```
elif not orientation_calibrated and distance_to_marker <
MAX_CALIBRATING_DISTANCE and marker_is_detected:
                twist = Twist()
                twist.linear.x = 0.0
                twist.linear.y = -marker_x_position * 0.8
                twist.angular.z = -0.2 * yaw
                move_cmd.publish(twist)
                angular_z = twist.angular.z

                if abs(marker_x_position) <= X_MARKER_PLACEMENT and
      abs(angular_z) <= ANGLE_TOLERANCE:
                    orientation_calibrated = True
```

Etter at kalibreringen av orienteringen er fullført, vil roboten motta en instruksjon om å kjøre rett frem til avstanden, i dette tilfellet 19 cm, fra markøren er oppnådd. Når avstanden er nøyaktig 19 cm, vil programmet skrive ut på skjermen at roboten er parkert, og deretter avslutte Python-modulen.

Resultatet vil være at vi har gjennomført tre faser. Der den første fasen er en distanse - kalibrasjonsfasen som også kalles for Move-to-point fasen, hvor avstanden blir kalibrert. Den andre fasen er orienteringskalibreringsfasen, også kjent som Move-to-pose fasen, hvor robotens orientering blir kalibrert. Også den siste fasen er parkeringsfasen, hvor roboten blir parkert. Hver fase har en spesifikk oppgave og bidrar til å oppnå det endelige målet med å parkere roboten. Mere om denne Python-modulen finnes under denne referansen [15].

3.4 Mekanisk dokking

I denne delen av bacheloroppgaven designer jeg min egen dokkingstasjon for å gjøre det visuelt enklere å se at roboten har dokket. Alle delene ble designet i Autodesk Inventor, og deretter ble de 3D-printet på skolen sine printere.

Ved å bruke Autodesk Inventor som design værverktøy, kunne jeg opprette 3D-modeller av dokkingstasjonen. Dette gjorde det mulig for meg å visualisere hvordan dokkingstasjonen ville se ut og passe sammen med roboten. Jeg kunne også gjøre justeringer og optimalisere designet før jeg gikk videre til produksjonsprosessen.

Etter å ha fullført designfasen, ble 3D-modellene eksportert til filer som er kompatible med skolens sine 3D-printere. Deretter ble delene 3D-printet på skolens utstyr, som tillot meg å konvertere designet mitt til fysiske komponenter. Ved å designe og 3D-printe dokkingstasjonen på denne måten, kan jeg skape en tilpasset løsning som passer perfekt til mine behov og gjør det tydeligere visuelt når

roboten er dokket. På Figur 22 kan dere se hvordan sluttresultatet ser ut og om nødvendig all teknisk informasjon om 3D modellene kan det finnes under Vedlegg Kobling Stasjon.

Figur 22 Fysisk dokkestasjon

3.5 UR5 robotarmen

Jeg opplevde at UR5-armen som var tilgjengelig på robotlabben ikke hadde blitt oppdatert siden 2016, og derfor manglet den oppdaterte UR5 API-en. Dette begrenset mulighetene mine for direkte interaksjon med armen. Som et resultat måtte jeg ta i bruk en alternativ tilnærming. Jeg utviklet et enkelt Move-to-pose program ved å bruke det medfølgende grensesnittet fra Figur 23 som var tilgjengelig med robotarmen. Dette grensesnittet tillot meg å sende kommandoer og instruksjoner til armen.

Selv om dette ikke var ideelt, gjorde det meg i stand til å utføre grunnleggende bevegelser og manipulasjoner. Ved å bruke det tilgjengelige grensesnittet klarte jeg å implementere en funksjonell løsning tilpasset de begrensede mulighetene som var tilgjengelig for UR5-roboten.

Ideen var å bruke robotarmen til å plassere et merke på arbeidsområdet med en tusj. For å oppnå dette har jeg utviklet et enkelt program som består av to forskjellige bevegelser. Den første bevegelsen er "moveL", som er en lineær bevegelse som tar roboten til startposisjonen. Den andre bevegelsen er "moveJ", som tillater å kjøre roboten med jevn og begrenset hastighet.

Figur 23 UR5 Grensesnittet. [2]

Gjennom "moveL" bevegelsen, sørger programmet for at roboten kjører til ønsket

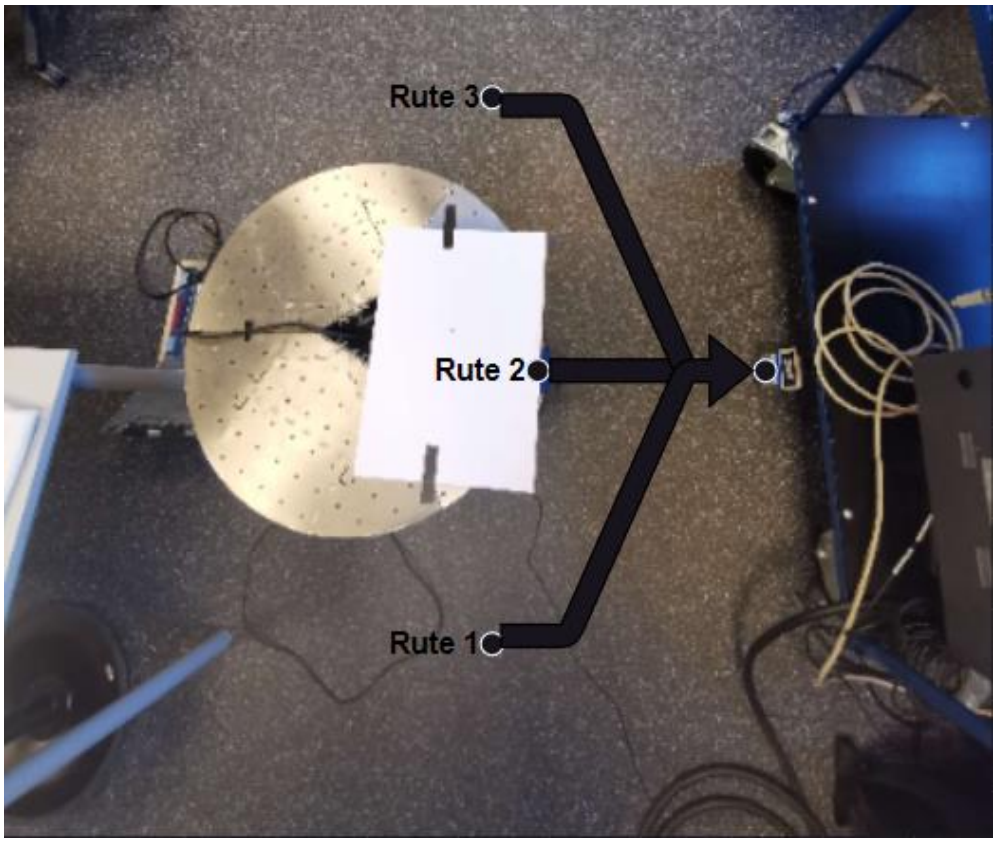
startposisjon. Denne bevegelsen er nødvendig for å sikre at robotarmen er riktig posisjonert før den utfører den faktiske markeringen med tusjen.

Deretter, ved hjelp av "moveJ" bevegelsen, utfører programmet en jevn og kontrollert bevegelse med en hastighet som er satt til 10% av den tilgjengelige maksimale hastigheten. Dette sikrer at markeringen blir gjort på en kontrollert måte og gir nøyaktighet under utførelsen.

Ved å kombinere "moveL" og "moveJ" bevegelsene i programmet, oppnår jeg muligheten til å plassere merket på arbeidsområdet ved hjelp av robotarmen med en tusj.

I tillegg til det, har jeg også designet og 3D-printet en egen griper fra Figur 24 for å sikre at tusjen som blir plassert i griperen har samme posisjon hver gang, uten behov for å kalibrere robotarmen på nytt.

Figur 24 3D printet griper

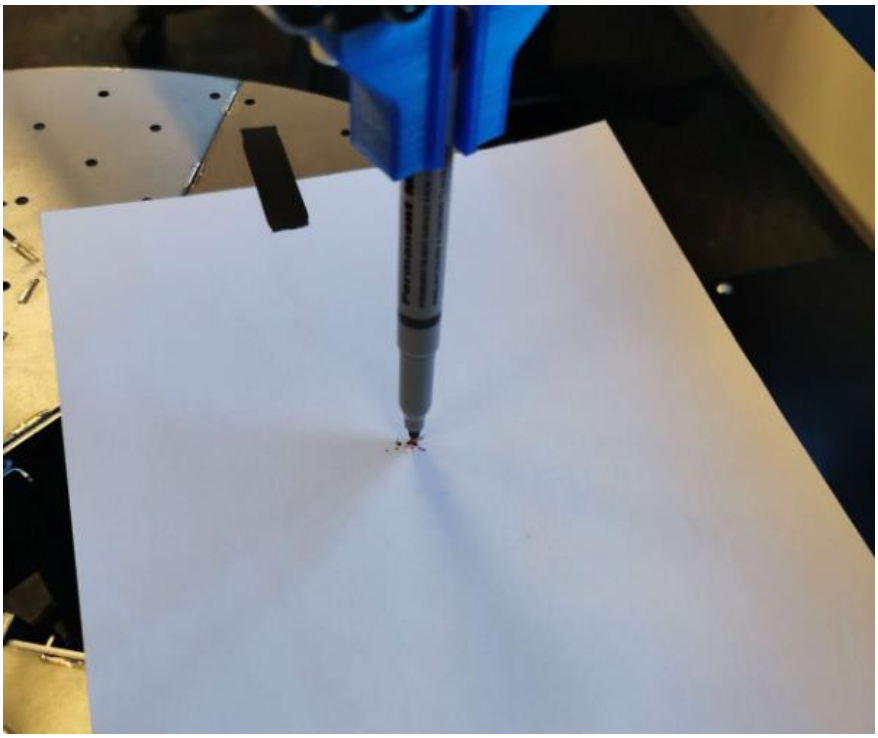


Figur 25 Oversikt over rutene som ble tatt under testene

4 Testing av systemet

Først utførte jeg vanlige tester uten å logge nøyaktighet eller presisjonen på parkeringen. Hovedformålet med disse testene var å finjustere navigasjonsalgoritmen. Jeg gjennomførte flere iterasjoner av testkjøringer for å optimalisere og sikre at robotens bevegelser var korrekte og forutsigbare. Under disse testene var hovedfokuset på å finjustere systemet, se Figur 26. Kort video som demonstrer hvordan systemet fungerer, finnes også under denne referansen [16].

Siden vi ikke har noe referansesystem eller et eksplisitt koordinatsystem, kan vi ikke bruke vanlige metoder for å kvantifisere presisjonen og nøyaktigheten, og selv om det kan bli noe mer utfordrende å vurdere presisjonen og nøyaktigheten til parkeringen blir en ustandardisert måte introdusert i Kapittel (4.1 Presisjon) og (4.2 Nøyaktighet).



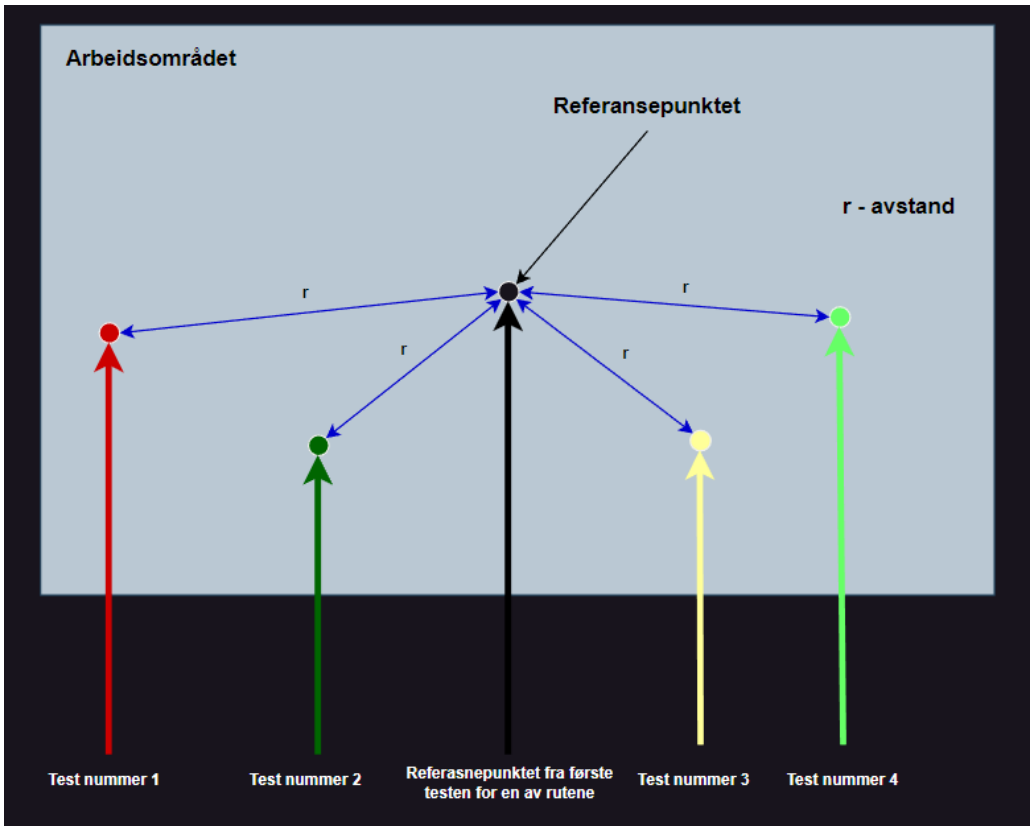
Figur 26 Eksempel på testene

4.1 Presisjon

Etter å ha oppnådd tilfredsstillende resultater med hvordan den mobile basen parkerte, gikk jeg over til å logge presisjonsdataen. Jeg ønsket å få kvantitative målinger av presisjonen til parkeringssystemet. For å oppnå det valgte jeg å utføre multidireksjonelle tester, som vist i Figur 25.

Jeg valgte å kjøre tre forskjellige ruter for å teste presisjonen til parkeringssystemet. Rute 1 hadde en vinkel på 60 grader i forhold til destinasjonsmålet, rute 2 hadde en vinkel på 90 grader, og rute 3 en vinkel på 120 grader. Disse forskjellige rutene tillot meg å evaluere systemets presisjon under forskjellige forhold og vinkler. Hensikten her er å la roboten kjøre fra en av de nevnte rutene og plassere et referansepunkt for hver rute første gangen roboten er parkert.

Ved å fokusere på testene fra for eksempel rute nummer 1 som blir også illustrert på Figur 27, kan vi analysere avstanden mellom referansepunktet og de påfølgende testpunktene som blir tatt for samme rute. Dette gir oss en indikasjon på presisjonen som oppnås for rute 1. Gjennom å gjenta denne analysen for alle tre rutene, kan vi også beregne presisjonen for de andre rutene. Dette gir oss en helhetlig forståelse av presisjonsnivået for de ulike rutene i prosjektet.

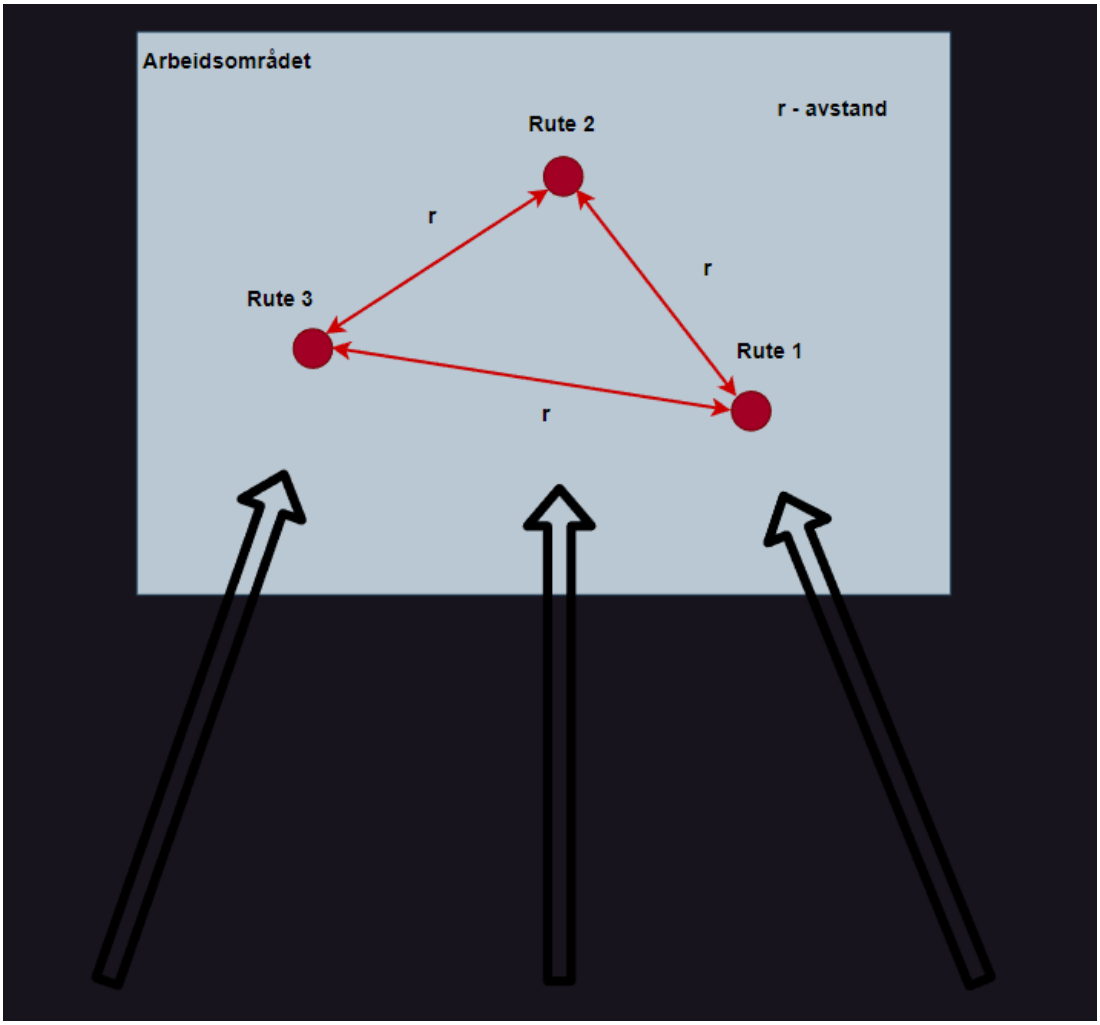


Figur 27 Måling av presisjon

4.2 Nøyaktighet

På samme måte som for presisjonstestene, bruker jeg de tre overnevnte rutene. Denne gangen utfører jeg testene etter hverandre for alle tre rutene. Etter hver parkering måles avstanden mellom for eksempel punktet som ble plassert fra rute 3 og rute 2, samt mellom rute 3 og rute 1, og mellom rute 2 og rute 1. Ved å se på den gjennomsnittlige avstanden fra hver rute kan vi finne de gjennomsnittlige senterpunktene som også blir vist i Figur 28. Disse punktene kan da brukes for å finne den totale nøyaktigheten for parkeringen. Figur 28 er en grafisk representasjon av nøyaktighetsmålingen. På denne måten kan jeg evaluere nøyaktigheten av parkeringen.

Alle testene ble utført på robotlabben ved HVL Førde. Dette laboratoriet ga meg de nødvendige ressursene og miljøet for å utføre pålitelige tester og samle inn presisjons og nøyaktighets data som var nødvendig for evaluering og analyse av systemets ytelse.



Figur 28 Måling av nøyaktigheten

V Resultater

1 Resulterende Presisjon

Fra arbeidsmetoden som ble introdusert i øvrig Kapittel (4.1 Presisjon) resulterer vi med følgende datasett.

Test nummer: xx	x: mm fra venstre posisjon P1	x: mm fra front posisjon P2	x: mm fra høyre posisjon P3
No: 1	1	2	2
No: 2	1	5	2
No: 3	1	3	1
No: 4	2	4	2
No: 5	5	2	1
No: 6	2	2	1
No: 7	2	3	3
No: 8	1	2	5
No: 9	2	4	3
No: 10	2	2	4
No: 11	3	3	3
No: 12	1	1	2
No: 13	1	3	3
No: 14	2	1	2
No: 15	4	1	5
No: 16	4	1	2
No: 17	1	1	2
No: 18	2	3	2
No: 19	1	1	2
No: 20	1	1	3

Tabell 2 Data med presisjonstestene for hver posisjon

Basert på denne dataen ser vi at den maksimale avstanden fra referansepunktet til testpunktene er 5 millimeter. Dette betyr at vi kan bruke denne 5 millimeter-grensen som referanse for å beregne presisjonen på parkeringen for alle tre rutene. Med andre ord, jeg er interessert i å vite hvor mange punkter som havner innenfor denne grensen og hva er sannsynligheten for det.

$$\frac{(om(antall\ verdier) \leq (1\ til\ 5)mm)}{Tot\ antall\ verdier} * 100$$

Ved å ta i bruk den øvrige formelen og justere grenseverdien fra 1 millimeter til 5 millimeter, kommer vi frem til følgende resultater.

Om målet er +- 5mm	RSD Rute 1:	100,00	%presisjon
	RSD Rute 2:	100,00	%presisjon
	RSD Rute 3:	100,00	%presisjon
Om målet er +- 4mm	RSD Rute 1:	95,00	%presisjon
	RSD Rute 2:	95,00	%presisjon
	RSD Rute 3:	90,00	%presisjon
Om målet er +- 3mm	RSD Rute 1:	85,00	%presisjon
	RSD Rute 2:	85,00	%presisjon
	RSD Rute 3:	85,00	%presisjon
Om målet er +- 2mm	RSD Rute 1:	80,00	%presisjon
	RSD Rute 2:	60,00	%presisjon
	RSD Rute 3:	60,00	%presisjon
Om målet er +- 1mm	RSD Rute 1:	45,00	%presisjon
	RSD Rute 2:	35,00	%presisjon
	RSD Rute 3:	15,00	%presisjon

Tabell 3 Resulterende presisjon under forskjellige grenseverdier

Basert på den øvrige tabellen kan vi observere at uansett hvilken rute som er valgt, vil roboten alltid oppnå 100% presisjon innenfor 5 mm-grensen. Imidlertid, hvis vi reduserer presisjonsgrensene, vil sannsynligheten for at roboten treffer innenfor det området også reduseres. Dersom vi også beregner gjennomsnittet av presisjonsmålingene, finner vi ut at gjennomsnittet for presisjonen i alle testene er 2 millimeter.

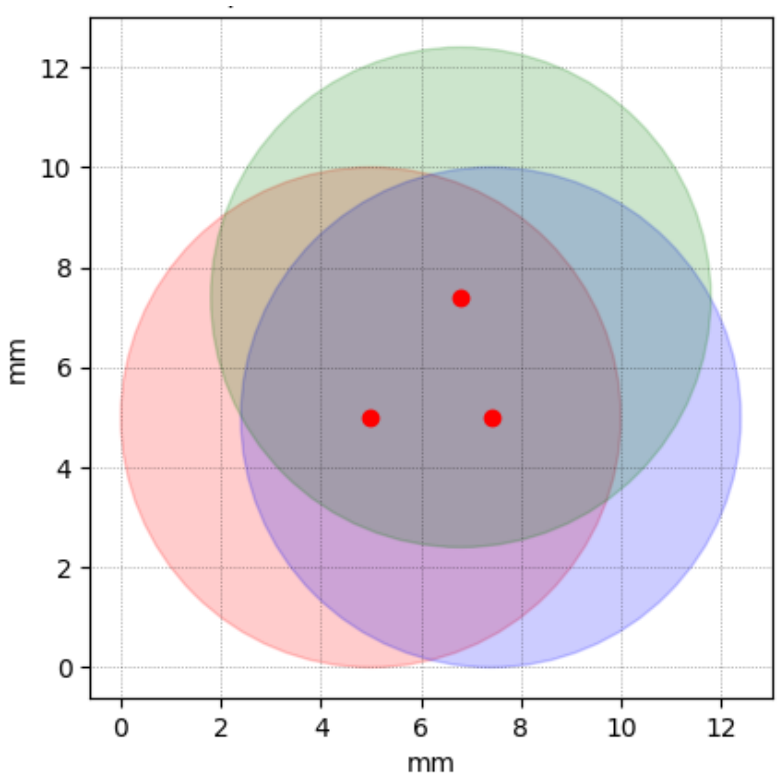
2 Resulterende Nøyaktighet

Fra arbeidsmetoden som ble introdusert i øvrig Kapittel (4.2 Nøyaktighet) resulterer vi med følgende datasett.

Avstanden mellom R3 og R2	2	2	2	2	4	mm
Avstanden mellom R2 og R1	4	2	3	2	1	mm
Avstanden mellom R3 og R1	2	4	5	4	3	mm

Gjennomsnittet mellom R3 og R2.	2,4 mm
Gjennomsnittet mellom R2 og R1	2,4 mm
Gjennomsnittet mellom R3 og R1	3,6 mm

Tabell 4 Gjennomsnittet mellom nøyaktighetsmålingene



Figur 29 Grafisk representasjon av nøyaktigheten

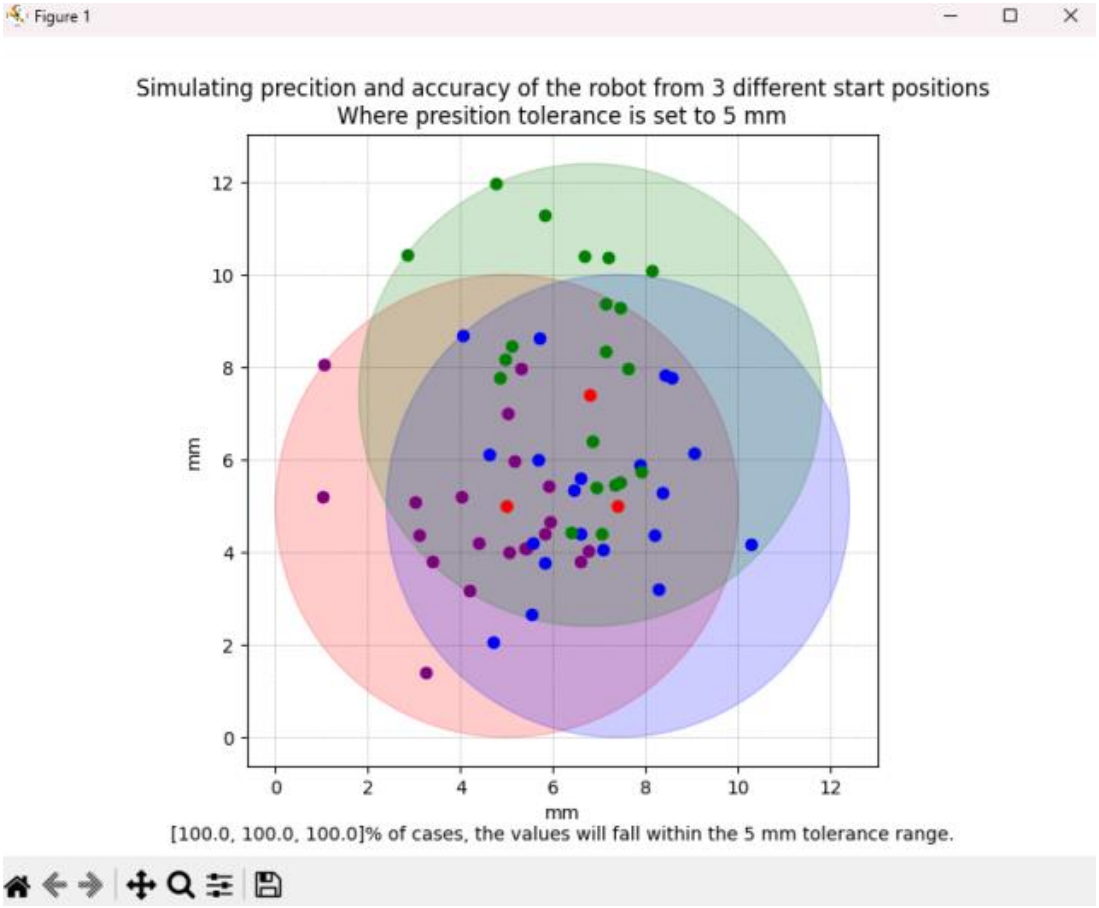
Ved å bruke gjennomsnittstabellen og Python kan jeg visualisere nøyaktigheten på målingene på en grafisk måte, noe som kan gjøre det enklere å observere og analysere resultatene. Ved å plugge gjennomsnittsverdiene for nøyaktigheten i python resulterer vi med en graf som blir vist på Figur 29. Ut fra denne grafen kan vi se at når presisjonsgrensen er satt til 5 millimeter vil vi ende opp med nøyaktigheten på cirka 12 millimeter som tilsvarer 1,2 centimeter.

3 Resultatet

For å oppsummere hva som ble sagt i de to forrige kapitlene, resulterer vi med en mobil plattform som parkerer med en presisjon på ± 5 millimeter i 100 % av tilfellene og ± 3 millimeter i 85 % av tilfellene. Der nøyaktigheten på den mobile basen er 12 millimeter i 100 % av tilfellene.

Ved å plugge alle verdiene inn i python kan dataen visualiseres med følgende Figur 30.

Python dataanalyserings-modulen finnes under denne referansen [15].



Figur 30 Grafisk representasjon av dataen

VI Konklusjon

Denne bacheloroppgaven har vært både utfordrende og spennende, og den krever god teoretisk kunnskap innen ulike områder. Oppgaven viser det store potensialet til AR-markører som en teknologisk løsning for presisjonsparkering av mobile enheter ved hjelp av et enkelt RGB-kamera og en bærbar datamaskin. Samtidig har oppgaven identifisert noen potensielle utfordringer, og gir forslag til hvordan disse kan løses for å forbedre løsningen.

Gjennom arbeidet mitt har jeg oppnådd alle målene for bachelorprosjektet. Jeg mener også at det er gode muligheter for videreutvikling av denne parkeringsløsningen. Det er et bredt spekter av anvendelser og mulige forbedringer som kan utforskes, og resultatene fra dette prosjektet kan danne et solid grunnlag for videre forskning og utvikling innen presisjonsparkeringsteknologi.

Gjennom hele prosjektet har jeg hatt god kommunikasjon med min veileder, som har vært til stor hjelp med å gi meg gode tips og veiledning innen ROS og Python i starten. Selv om jeg har jobbet alene, kunne jeg gjerne tenkt meg å arbeide i en gruppe, da oppgaven var spennende og det ville vært muligheter for å utvide og utforske videre.

Resultatet av prosjektet er en mobil base som parkerer svært presist, med god nøyaktighet, og jeg er svært stolt av dette. I tillegg har robotarmen blitt brukt som et måleverktøy for parkeringsdataene.

Jeg er også svært fornøyd med resultatene fra prosjektet på grunn av at jeg hadde forventet mye verre resultater enn det vi faktisk oppnådde.

Dette har vært et svært lærerikt prosjekt som har gitt meg verdifull innsikt i hvordan man planlegger og gjennomfører store prosjektoppgaver som denne. Jeg er sikker på at jeg vil dra nytte av denne erfaringen både i arbeidslivet og i mitt personlige liv.

VII Diskusjon

1 Utfordringer

I denne bacheloroppgaven var den største utfordringen selve programmeringen i Python og ROS. Selv om jeg hadde tidligere erfaring med ROS gjennom faget ELE306-1 22H Robotikk, hadde jeg i hovedsak fokusert på Matlab. Dette skapte en del utfordringer i begynnelsen, da jeg ikke var kjent med ROS og måtte bruke tid på dette for å bli kjent med strukturen og syntaksen.

Opprinnelig var planen å bruke ROS-navigasjonsstakken for å navigere roboten til dokkestasjonen. Dessverre gikk ikke denne delen som planlagt, da jeg støtte på flere feilmeldinger og hadde vanskeligheter med å løse dem. Etter å ha prøvd å håndtere feilmeldingene uten suksess, måtte jeg gi opp denne tilnærmingen og starte med å utvikle en egen navigasjons-algoritme.

Når det gjelder robotarmen, var planen å montere den på toppen av Robotinoen. Vi hadde opprinnelig valgt Open Manipulator robotarm, som er montert på TurtleBot 3-plattformen. Dessverre møtte jeg også utfordringer her, da ROS versjonen til TurtleBot-plattformen også ikke hadde blitt oppdatert siden 2016. Dette førte til problemer når jeg skulle installere visse pakker som ikke lenger støttet denne versjonen av TurtleBot-plattformen. De fleste pakkene som var tilgjengelige for Open Manipulator var utviklet for ROS melodic, der TurtleBot-plattformen ble sist oppdater med ROS Kinetic som er en eldre versjon av ROS. Selv om det hadde vært mulig å oppdatere TurtleBot 3-plattformen, var det en veldig kompleks oppgave som ville krevd mye tid og ressurser.

2 Forbedringsmuligheter

Selv om roboten utfører parkeringen som den skal, er det fortsatt rom for forbedring av systemet. En av de første forbedringene jeg ønsker å implementere er reguleringen av navigasjonsalgoritmen. For øyeblikket bruker algoritmen bare en P-regulator for å kompensere for avvik i orientering. Her ønsker jeg å prøve å implementere en komplett PID-regulator som kan muligens gi en raskere og bedre navigering sammenlignet med en P-regulator alene. Selv om jeg ikke er 100% sikker på om en full PID-regulator vil resultere i bedre navigasjon, vil det være mulighet for å teste og evaluere dette.

Det jeg også ønsker å forbedre i dette prosjektet er alt som er relatert til maskinvaren. Basert på resultatene vi har oppnådd, kan vi se at vi oppnår god presisjon og en akseptabel nøyaktighet i parkeringen. Imidlertid mistenker jeg at avviket i presisjon og nøyaktighet kan skyldes faktorer som et billig kamera og utdatert programvare til robotarmen, som av og til kan føre til grove bevegelser.

Det er også muligheten for å øke stivheten til alle komponentene. For eksempel bruker vi for øyeblikket en isolasjonsteip for å holde kameraet på plass som er ikke et ideelt løsning. Ved å bruke komponenter som oppfyller industrielle standarder som kan sørge for riktig festing og montering, kan vi ytterligere øke presisjonen og nøyaktigheten i systemet.

For å oppnå nøyaktig og presis parkering, kan HVL vurdere å implementere kalibrering av robotarmen ved hjelp av AR-markører og et kamera. Kameraet kan monteres på griperen til robotarmen og brukes til å gjenkjenne AR-markørene. Dette kan være mulig å implementere på en

nyere modell av UR5 robotarm, siden denne har tilgjengelig API-funksjonalitet som kan motta en Pose-melding fra datamaskinen. Deretter kan man bruke en "Move to pose"- bevegelse som allerede er implementert i UR5-grensesnittet for å utføre presise bevegelser basert på kalibreringsdataene. Dette kan bidra til å minimere avviket mellom robotarmens posisjon og ønsket posisjon, og dermed øke nøyaktigheten og presisjonen på systemet.

Om det er også ønskelig, kan en digital modell lages for å simulere den mobile basen. I denne sammenhengen har jeg utviklet et C++-bibliotek [17] som inneholder funksjoner for fremover kinematikk (FK) og invers kinematikk (IK), som er klare til bruk.

Ved å bruke dette biblioteket kan man simulere bevegelsene og posisjonene til den mobile basen på en nøyaktig og effektiv måte. Dette kan være en verdifull ressurs for testing, optimalisering og videreutvikling av parkeringsløsningen, da det tillater å evaluere forskjellige scenarier og justeringer uten behov for fysisk tilstedeværelse av den mobile basen.

Python-filen Robotino_tag_tracker.py kan også struktureres som en klasse for å legge til abstraksjon, modularitet, utvidbarhet og fleksibilitet til modulen. Ved å implementere klassen kan man organisere koden på en mer objektorientert måte, som gjør det enklere å håndtere og vedlikeholde. Dette vil også bidra til bedre lesbarhet, gjenbrukbarhet og mulighet for å utvide funksjonaliteten ved behov. Ved å bruke klassen kan man også oppnå en mer fleksibel og modulær struktur som gjør det enklere å integrere og samhandle med andre deler av prosjektet eller eksterne biblioteker.

VIII Prosjektadministrasjon

1 Organisering

- Oppdragsgiver: HVL Campus Førde
- Kontaktperson: Marcin Andrzej Fojcik
- Veileder: Gizem Ates Venås
- Prosjektgruppe: Mikael Walde

Under bachelorprosjektet fikk jeg også diskutert oppgaven med forskere under en reise til AIUT/Polen. Denne turen ga meg nyttig informasjon og tips relevante til oppgaven og rapporten generelt.

2 GitHub

Det ble opprettet en GitHub-repository [13] for å organisere og opprettholde koden og pakkene som ble utviklet og oppdatert i løpet av prosjektet.. I repositoryen er det også inkludert en trinnvis veiledning for installasjon og bruk av pakken, slik at andre kan teste eller videreutvikle prosjektet. Koden ble utviklet i samsvar med Python Code Style, noe som resulterer i ryddig og oversiktlig kode som legger til rette for enkel videreutvikling.

3 Tid

Når det gjelder tidsbruken, ble det opprinnelig estimert å bruke 500 timer på bachelorprosjektet. Til slutt endte jeg opp med 446 timer, som er mindre enn det som ble planlagt. Se Vedlegg Gantt og Vedlegg Arbeidsoppgaver med Timeføring.

4 Kostnader

De eneste direkte kostnadene jeg kan relatere meg til er reisen til Polen. Disse kostnadene ble dekket og refundert av HVL. Andre kostnader i prosjektet inkluderer utstyret som ble stilt til rådighet av HVL Robotics lab i Førde. Dette utstyret ble lånt eller gitt til meg uten ekstra kostnad for å kunne gjennomføre prosjektet. Se Vedlegg Budsjett.

5 Dokumentstyring

For bachelorprosjektet ble det opprettet en felles mappe via Google Drive, der alle relevante filer ble lagret. Bruken av Google Drive ble valgt med tanke på enkel tilgang til dokumentene fra forskjellige datamaskiner. Dette er spesielt nyttig når jeg er på skolen og tester ting med en bærbar PC, og når jeg er hjemme og bruker den stasjonære maskinen. Google Drive ble også brukt som en backup-løsning for å sikre at PDF eller Word filene mine ikke gikk tapt hvis en av maskinene skulle ryke.

6 Arbeidsmetoder

Fra begynnelsen av prosjektet bestemte jeg meg for å organisere arbeidet ved å opprette oppgaver som skulle fullføres i løpet av bachelortiden. For hver dag jeg jobbet med en bestemt oppgave, registrerte jeg antall timer jeg jobbet den dagen. Selv om dette kanskje ikke var den mest effektive måten å organisere arbeidet på, kom jeg på dette litt sent ute i bachelortiden til å endre på det. Etter hvert som jeg arbeidet, la jeg til flere oppgaver som skulle fullføres. Totalt sett endte jeg opp med et litt lavere antall timer enn det som var forventet. Dette har imidlertid ikke påvirket sluttresultatet av rapporten. Oversikten over alle arbeidsoppgavene og timene som ble brukt for å løse dem finnes under Vedlegg Arbeidsoppgaver med Timeføring.

7 Risikovurdering

En risikovurdering brukes til å identifisere og evaluere risikoene i et prosjekt. Dette gjøres ved å identifisere alle mulige faremomenter som kan oppstå. Når faremomentene er identifisert, kan man vurdere konsekvensene av hver fare og sannsynligheten for at den faktisk oppstår. Ved å kombinere konsekvens og sannsynlighet kan risikonivået for hver fase vurderes.

Siden jeg bruker UR5 robotarmen må risikovurdering utføres ifølge robotens manual. Manualen anbefaler å følge at operatøren bruker retningslinjene for ISO 10218-2 paragraf 4.3 for å gjennomføre risikovurderingen. Robotino som en enhet faller også under samme ISO-standarden som kan brukes til å risiko vurdere systemet.

- ISO 10218-2 er en samling med standarder for roboter og robottekniske innretninger.

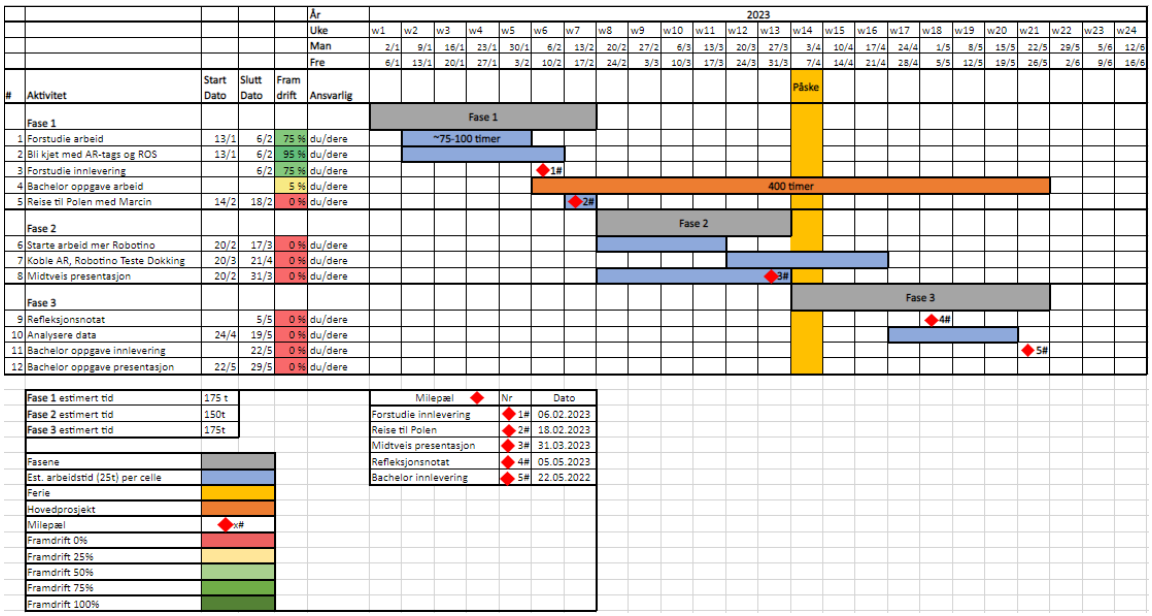
Under Vedlegg Risikovurdering finnes det risikovurderingen som ble gjort for prosjektet, denne inkluderer Robotino og UR5 robotarmen.

8 Prosjektmøter

Grunnet at jeg jobber alene har jeg ingen å kjøre prosjektmøter med. Men av og til har vi hatt møtene med min veileder. Møtereferater fra slike møter medfører under Vedlegg Møtereferat.

IX Vedlegg

1 Vedlegg Gantt



2 Vedlegg Budsjett

Budsjett for Bachelor oppgave 2023

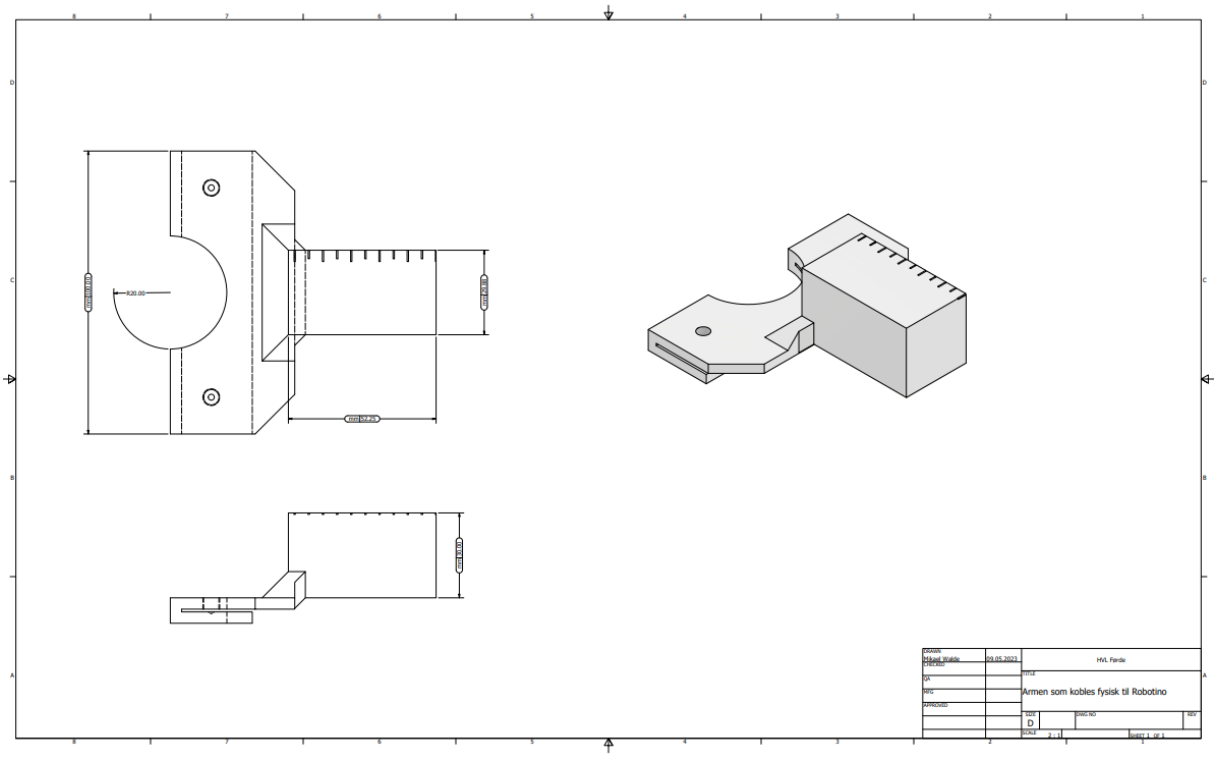
Oppgave nr	BO23EF-03
Antall studenter	1
Bedrift	HVL
HVL Veileder	Gizem Ates Venås

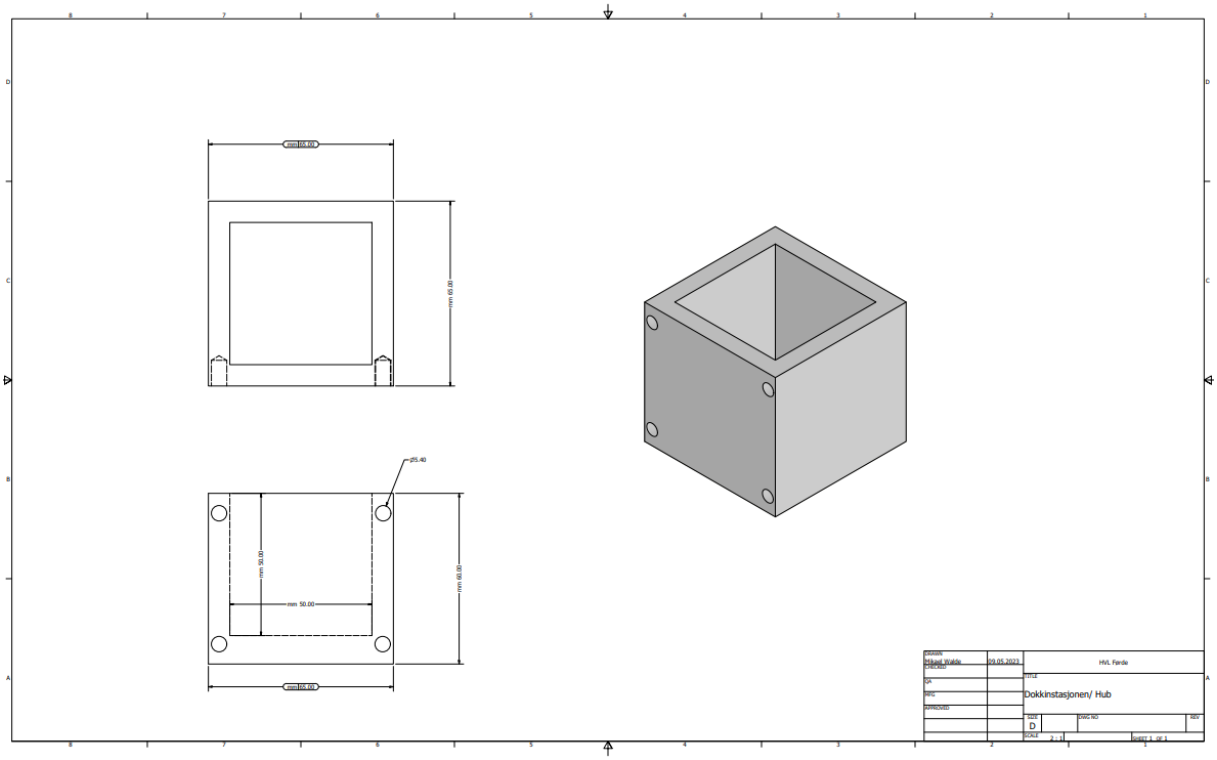
		Kroner	Hvem skal betale ?		
#	**Beskrivelse av utgift**	**Kost**	**Studenter**	**Bedrift**	**HVL**
1	Reise til Polen	2040	1	HVL	
2	UR5 Robot	300000		HVL	
3	Web camera Logitech C920 HD PRO	900		HVL	
4	Robotino	65000		HVl	
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					
	Totalt	**367940**	**1**	**0**	**0**

3 Vedlegg Arbeidsoppgaver med Timeføring

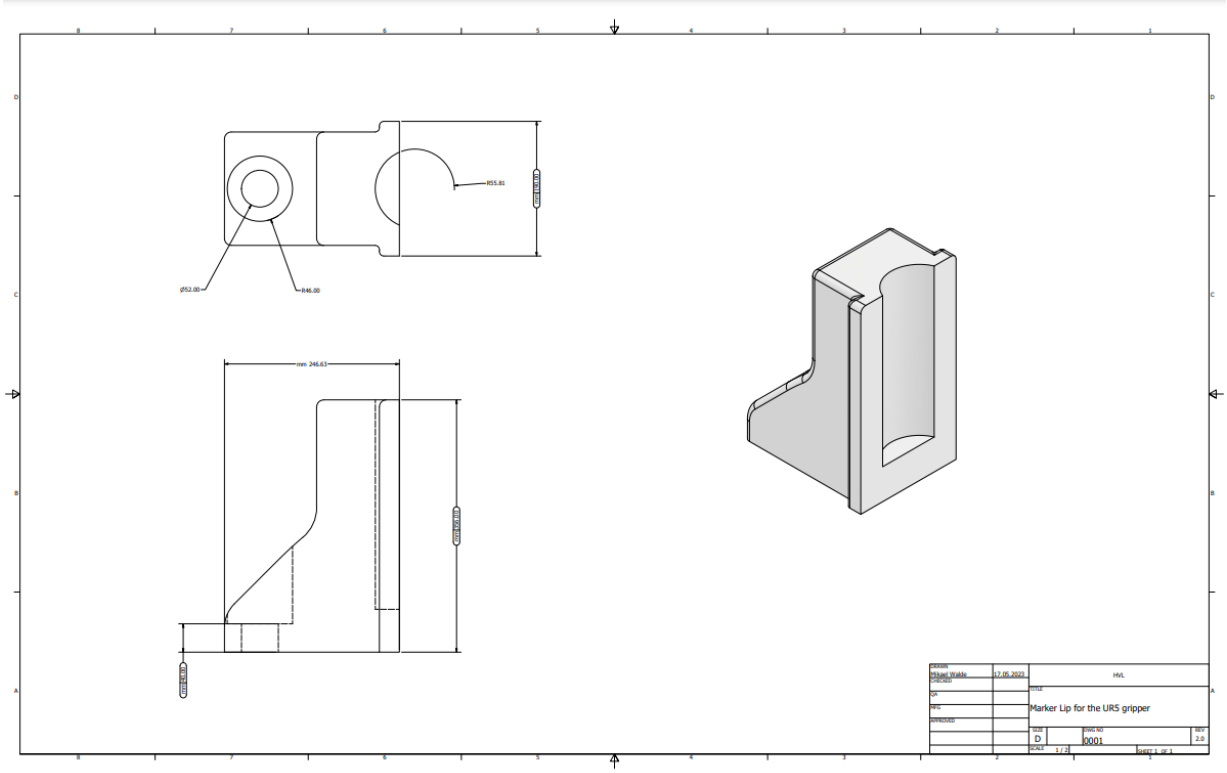
Rabeidsoppgave	Timer brukt på arbeidsoppgaven
Begynne å bli kjent med teorien rundt AR markører.	25
Teste deteksjon av AR markører og konvertere ar pose til Rotasjonsmatrisene	10
Bli kjent med ROS	40
Skrive ferdig Forprosjektrapport	30
Starte med å Open manipulator	40
Midtveispresentasjon	10
Oppstart arbeid med robotino	25
Oppstart arbeid med UR5 robotarm	25
Teste ut Navigasjonstakken	30
Begynne med simpel Move to pose	30
Koble AR til robotino, teste dokking	30
Teste dokking med robotarmen	15
Refleksonsnotatet	10
Github Dokumentasjon	16
Dataanalyse	20
Hoved Rapporten	90
Totalt	446

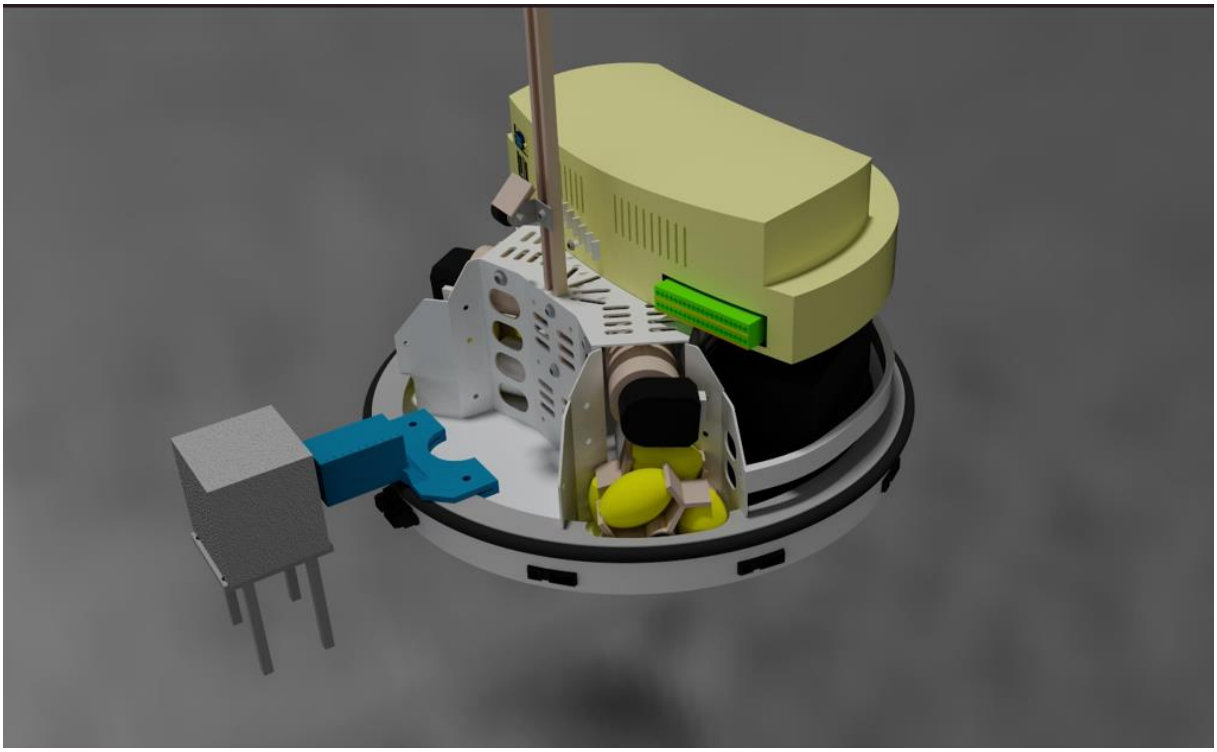
4 Vedlegg Kobling Stasjon





5 Vedlegg Gripper





6 Vedlegg Analyse

Test nummer : X	x: mm fra venstre posisjon P1	x: mm fra front posisjon P2	x: mm fra høyre posisjon P3
No: 1	1	2	2
No: 2	1	5	2
No: 3	1	3	1
No: 4	2	4	2
No: 5	5	2	1
No: 6	2	2	1
No: 7	2	3	3
No: 8	1	2	5
No: 9	2	4	3
No: 10	2	2	4
No: 11	3	3	3
No: 12	1	1	2
No: 13	1	3	3
No: 14	2	1	2
No: 15	4	1	5
No: 16	4	1	2
No: 17	1	1	2
No: 18	2	3	2
No: 19	1	1	2
No: 20	1	1	3

Gjennomsnitt for Posisjon1:	1,95
Gjennomsnitt for Posisjon2:	2,25
Gjennomsnitt for Posisjon3:	2,5

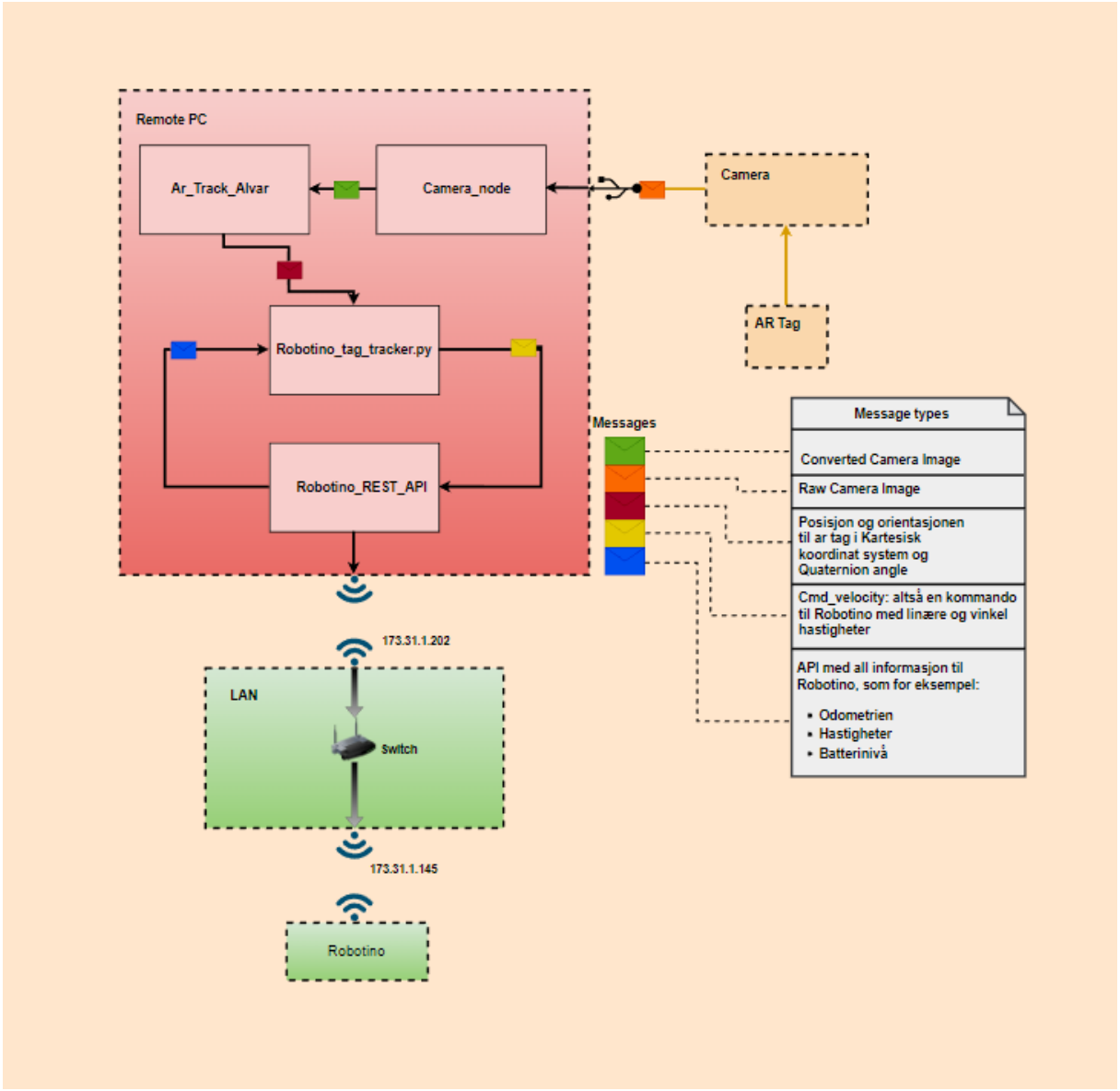
Median for posisjon1:	2
Median for posisjon2:	2
Median for posisjon3:	2

Standaravviket for posisjon1:	1,160818677
Standaravviket for posisjon2:	1,177921899
Standaravviket for posisjon3:	1,118033989

Om målet er +-5mm	RSD Rute1:	100,00	%presisjon
	RSD Rute2:	100,00	%presisjon
	RSD Rute3:	100,00	%presisjon
Om målet er +-4mm	RSD Rute1:	95,00	%presisjon
	RSD Rute2:	95,00	%presisjon
	RSD Rute3:	90,00	%presisjon
Om målet er +-3mm	RSD Rute1:	85,00	%presisjon
	RSD Rute2:	85,00	%presisjon
	RSD Rute3:	85,00	%presisjon
Om målet er +-2mm	RSD Rute1:	80,00	%presisjon
	RSD Rute2:	60,00	%presisjon
	RSD Rute3:	60,00	%presisjon
Om målet er +-1mm	RSD Rute1:	45,00	%presisjon
	RSD Rute2:	35,00	%presisjon
	RSD Rute3:	15,00	%presisjon

	Best
	Bra
	Normal
	Nøytral
	Dårlig

7 Vedlegg Kommunikasjonskart



8 Vedlegg Møtereferat

Høgskulen på Vestlandet

Bachelor møte

Deltagere:

Gizem Atez Venås

Mikael Walde

Sted:

HVL Førde Grupperom Hulda

Dato:

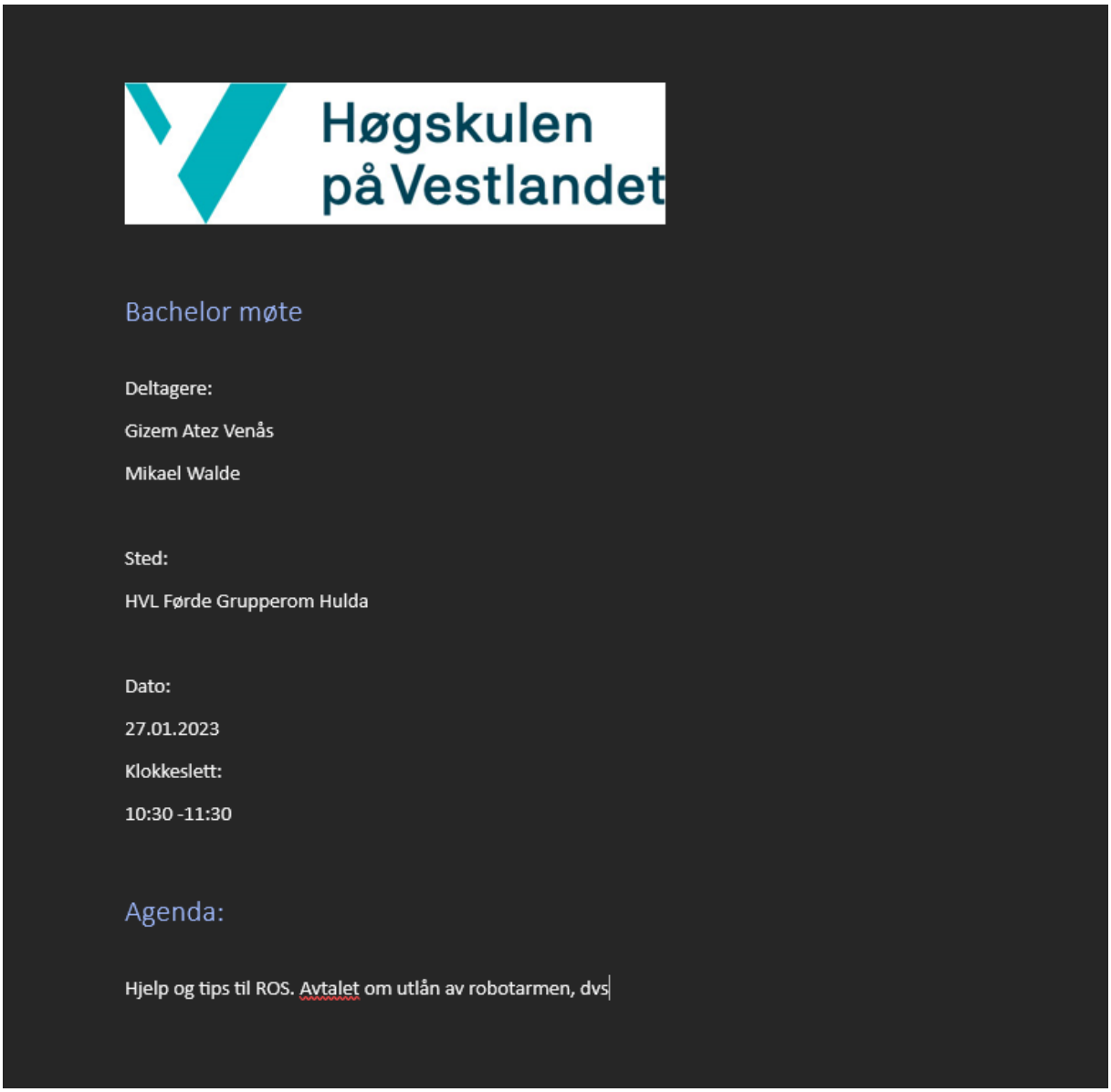
27.01.2023

Klokkeslett:

10:00 -11:30

Agenda:

Oppstart av arbeidet, planen for hvordan bacheloren skulle utføres. Eventuell info angående reise til Polen.


Høgskulen
på Vestlandet
Bachelor møte
Deltagere:
Gizem Atez Venås
Mikael Walde
Sted:
HVL Førde Grupperom Hulda
Dato:
27.01.2023
Klokkeslett:
10:30 -11:30
Agenda:
Hjelp og tips til ROS. Avtalet om utlån av robotarmen, dvs


Høgskulen
på Vestlandet
Bachelor møte
Deltagere:
Gizem Atez Venås
Mikael Walde
Sted:
HVL Førde Grupperom Hulda
Dato:
22.02.2023
Klokkeslett:
10:00 -11:00
Agenda:
Diskusjon og spørsmål rundt AR markørene. Diskusjonen angående Polen turen og veien videre.


Høgskulen
på Vestlandet
Bachelor møte
Deltagere:
Gizem Atez Venås
Mikael Walde
Sted:
HVL Førde Robotlabben
Dato:
15.05.2023
Klokkeslett:
13:00 -14:00
Agenda:
Gjennomgang av Sluttrapporten

9 Vedlegg Risikovurdering

Sannsynlighet	Konsekvens					
		1	2	3	4	5
	1	1	2	3	4	5
	2	2	4	6	8	10
	3	3	5	9	12	15
	4	4	6	12	16	20
	5	5	10	15	20	25

Risikomatrise

No	Risiko	Årsak	sannsynlighet	Konsekvens	Risikofaktor	Tiltak
1	Kollisjon og klemfare	Kollisjon med mennesker Klemme legemer i et ledd	4	5	20	Egen sikkerhetsinstruksen må utarbeideides
2	Skade på omgivelser	Robot kan kollidere med vegg eller komponenter	3	4	12	Passe på at koden inneholder safety funksjonalitet.
3	Skade på utstyr	Feil bruk	2	3	6	Bli kjent med manualen
4	Elektrisk støt	Elektrisk støt fra strømførende komponenter	2	5	10	Unngå å arbeide AUS

Risikovurdering

X Referanser

[1] auit, «AUIT.com,» 2022. [Internett]. Available: https://aiut.com/en/solutions/automation-and-robotics/mobile-production-stations/.

[2] U. robotics, «Universal robotics,» [Internett]. Available: https://www.universal-robots.com/no/academy/?utm_source=Google&utm_medium=cpc&utm_cja=Academy&utm_leadsource=Paid%20Search&utm_campaign=HQ_NO_Always-On2021&utm_content=textad&utm_term=universal%20robots%20academy&gclid=CjwKCAjw04yjBhApEiwAJcvNoR4kiObZUO2U.

[3] M. Korada, «AR Tag Detection,» [Internett]. Available: https://madhu.work/project/ar_tag_detection/.

[4] [Internett]. Available: https://www.logistikkinside.no/bring-distribusjon-posten/posten-tester-robot-henting-pa-first-mile/714338.

[5] «ROS.org ar_track_alvar,» 19 07 2016. [Internett]. Available: http://wiki.ros.org/ar_track_alvar.

[6] A. Abdulov og A. Abramenkov, «IEEE Xplore,» 2015. [Internett]. Available: https://ieeexplore-ieee-org.galanga.hvl.no/document/9112035.

[7] A. Kawecki, P. Dąbrowski, S. Januszko, M. Rećko og K. Dzierżek, «IEEE Explore,» 2017. [Internett]. Available: https://ieeexplore-ieee-org.galanga.hvl.no/document/9738534.

[8] OpenCV, «Opencv.org,» 2023. [Internett]. Available: https://opencv.org/about/.

[9] ROS, «navigation - ROS Wiki,» 14 09 2020. [Internett]. Available: http://wiki.ros.org/navigation.

[10] Osoyoo. [Internett]. Available: https://osoyoo.store/products/4wd-omni-wheel-robotic-mecanum-wheel-robot-car-platform-chassis-with-dc-speed-encoder-motor-for-arduino-raspberry-pi?variant=31634957336687.

[11] Wikipedia, «PID Controller,» 2023. [Internett]. Available: https://en.wikipedia.org/wiki/PID_controller.

[12] F. Didactic, «Robotino Hardware Control Unit,» [Internett]. Available: https://ip.festo-didactic.com/InfoPortal/Robotino/Hardware/Controller/EN/index.html#.

[13] M. Walde, «2023_Bachelor_Project_AGV_Docking,» [Internett]. Available: https://github.com/meldew/2023_Bachelor_Project_AGV_Docking.

[14] Arduino, «Arduino Map function,» 2023. [Internett]. Available: https://reference.arduino.cc/reference/en/language/functions/math/map/.

Referanser

[15] M.W, «Robotino_tag_tracker.py,» 2023. [Internett]. Available: https://github.com/meldew/2023_Bachelor_Project_AGV_Docking/blob/master/src/Robotino_tag_tracker.py.

[16] M.W, «Autonomous precision parking with AR,» [Internett]. Available: https://www.youtube.com/watch?v=3h_kH3zjrUo&ab_channel=MikaelWalde.

[17] M.W, «Diff_models.h,» 2023. [Internett]. Available: https://github.com/meldew/2023_Bachelor_Project_AGV_Docking/blob/master/other_files/Diff_models.h.

[18] L. A. Mateos, W. Wang, B. Gheneti, F. Duarte, C. Ratti og D. Rus, «IEEE Xplore,» 2022. [Internett]. Available: https://ieeexplore.ieee.org/document/8793525/authors#authors.

[19] «Transformasjonsmatrise Wikepedia,» [Internett]. Available: https://no.wikipedia.org/wiki/Transformasjonsmatrise.

[20] M. Walde, «Github,» 2023. [Internett]. Available: https://github.com/meldew/2023_Bachelor_Project_AGV_Docking.

[21] M. Walde, «data_analysis,» 2023. [Internett]. Available: https://github.com/meldew/2023_Bachelor_Project_AGV_Docking/blob/master/other_files/data_analysis.py.