Høgskulen på Vestlandet
Bergen, Vår 2021

Datainnhenting med Clearpath Jackal og computer vision

Data collection with Clearpath Jackal and computer vision

Fredrik Skjerve Jensen & Ingvar Filip Auestad

Veileder: Adis Hodzic

Bachelorutreding i automasjon med robotikk

Høgskulen på Vestlandet

Forord

Denne oppgaven er gjennomført våren 2021 ved Høgskulen på Vestlandet. Det er den avsluttende oppgaven for vårt bachelorstudium i automatisering med robotikk. Oppgaven er utformet av to studenter med tidligere bakgrunn fra fagskole og fagbrev.

November 2020 startet arbeidet med å finne aktuelle prosjekter. Oppgaven vi bestemte oss for var ressurssporing med bruk av en mobil robot, et prosjekt vi fikk gjennom Cognite AS. Vi følte at oppgaven var nytenkende og interessant, samtidig gav den mulighet for ny kunnskap og erfaring som kan brukes i fremtidige prosjekt.

Rapporten har til hensikt å dokumentere vårt arbeid gjennom prosjektoppgaven. På grunn av forsinkelser og COVID-19 utgikk de fysiske testturene. Dette resulterte i at vi fokuserte mer på programmering og videreutvikling av programmene som ble utviklet gjennom prosjektet. Det har gitt oss et stort læringsutbytte i programmeringsspråket Python som var et nytt språk for oss. Vi har fått en dypere forståelse for robotikk-systemer, og hvordan de kan integreres i moderne skyløsninger for industriell data.

Vi vil først og fremst rette en stor takk til vår kontaktperson i Cognite, Kevin Karlsholm Kaldvansvik, for god hjelp og rettledning gjennom hele prosjektet. Takk til vår veileder i Cognite, Johan Hatleskog, for hans ekspertise innen robotens oppsett og CDF (Cognite Data Fusion). Samt takk til vår veileder ved HVL, Adis Hodzic, for hans oppfølging og veiledning.

Høgskulen på Vestlandet

Bergen, mai 2021

Fredrik Skjerve Jensen

Ingvar Filip Auestad

Sammendrag

Oppgaven er en del av et utviklingsprosjekt for autonom ressurssporing på et industriområde. Per dags dato er det en manuell jobb å holde oversikt over hvor forskjellige ressurser er plassert. Cognites prosjekt har som mål å undersøke om det er mulig å digitalisere dette ved hjelp av en mobil robot. For å løse oppgaven er det benyttet en Clearpath Jackal, en hjulbasert robot utviklet av Clearpath Robotics. Sensorene vi skal fokusere på er GPS, IMU og kamera. Se ordliste for forklaring på forkortelser.

Hovedelementene som er utarbeidet gjennom semesteret er Python programmer for utpakking av sensor data, opplasting av utpakket data til CDF og filtrering av GPS-koordinater til waypoint navigering. Ved datainnsamling på Clearpath Jackal lages det en rosbag(`bag`). Den muliggjør innhenting av all data på en felles tidslinje. Dermed har en oversikt over nøyaktig tidspunkt for målingene, og kan enklere sammenligne data. Kombinasjonen av rosbag og sensor tillater digital posisjonering av ressursene som spores.

Utpakking av data gjennomføres i to seksjoner: Skriving til `csv` for generell sensor data, og skriving til `jpeg` og `npy` for kameradata. Det genereres en filstruktur basert på innholdet i rosbagen som leses, og navnet på filen. Se figur 6.1 for eksempel.

Opplasting baserer seg på de utpakkede filene. Det har tre hovedelement: Oppdatering av Timeseries, opplasting av kameradata og opplasting av generell data. Det er opprettet Timeseries i CDF for IMU- og GPS-data. Disse oppdateres med ny informasjon fra de relevante filene. Bilder og dybdefiler lastes opp med tilknytting til kamera som de tilhører. De resterende filene lastes opp med tilknytting til roboten. Filene blir merket med navnet på bagfilen de tilhører, det bedrer sorteringen i CDF miljøet.

Uthenting av GPS-koordinater gjøres direkte fra rosbag. Den analyseres for et konkret topic, der GPS-data blir lagret. Denne informasjonen hentes ut, og filtreres etter antall meter mellom hvert punkt, som blir oppgitt av bruker. Dataen blir lagret som en `json` fil for enklere bruk i waypoint navigering.

Til tross for endringer og utfordringer i prosjektet mener vi at oppgaven løser problemstillingen på en tilfredsstillende måte. Vi har implementert systemer for data-behandling og -opplasting, og tilpasset det til ROS miljøet på Jackal.

Summary

This thesis is part of a development project for autonomous resource tracking in an industrial area. As it stands today it is a manual job to keep track of resources and their location. Cognites project intends to solve this with a mobile robot. The plan is to use Clearpath Jackal, a wheel-based robot developed by Clearpath Robotics. The focus of our bachelor thesis is treatment of sensor-data from GPS, IMU and camera related to Clearpath Jackal.

Primarily we have developed a set of python scripts for extraction of data, uploading data to CDF, and filtering GPS-coordinates for waypoint navigation. Clearpath Jackal collects data in a rosbag(`bag`). This facilitates data-collection on a shared timeline, which allows for an easy overview of when all data is collected. The combination of rosbag and sensor data allows one to discern the location of resources that needs tracking. Extraction of data is a two-step process: Writing to `csv` for general sensor data and writing to `jpg` and `npy` for camera data. A file structure is generated based on the contents of the rosbag and its name. See figure 6.1 for illustration.

Uploading to CDF is based on the extracted data. There are three main elements: Updating Timeseries, uploading camera data, and uploading general data. Timeseries has been created in CDF for IMU and GPS data. These are updated with new information from the relevant files. Pictures and Depth data is uploaded under the relevant camera asset. All remaining data files are uploaded under the robot asset. Files are registered with the name of the bag they derive from; this helps with sorting in CDF.

GPS waypoints are extracted directly from a rosbag. It is analysed for a specific rostopic where GPS data is stored. The data is extracted and filtered by meters between each point, which is defined by the user. The data is then stored as a `json` file for later use in waypoint navigation.

Despite changes and challenges during the project we attest that we solved the main issue in a satisfactory way. Systems were implemented for data extraction and data upload and customized for Clearpath Jackals ROS environment.

Keywords – ROS, Python, datalogging, data-notasjon, skylagring, robotikk, kamerakalibrering

Innhold

Figurliste

Ordliste

Forkortelse	**Navn**	**Forklaring**
API	Application Programming Interface	Programmeringsgrensesnitt: sørger for kommunikasjon mellom applikasjoner
bag	rosbag	Filtype i ros som samler sensor data på en felles tidslinje
CDF	Cognite Data Fusion	Cognites industiell dataplattform
CVAT	Computer Vision Annotation Tool	Data-annotasjons program
GPS	Global Positioning System	Navigasjonssystem som benytter sattelitter
IMU	Inertial measurement unit	Måleinstrument bestående av accelerometer, gyroskop og magnetometer
OpenCV	Open Source Computer Vision Library	Programvarebibliotek til bildebehandling for computer vision
OS	Operating System	Oprativsystemet er den grunnlegende programmvaren på en datamaskin
ROS	Robot Operating System	Samling av programmeringsbibliotek for utvikling av robot-programvare
SDK	Software Development Kit	Programvareutviklingsverktøy levert av utvikler for et produkt
SSH	Secure Shell	Dataprogram og nettverksprotokoll for kommunikasjon mellom maskiner
Topic	rostopic	Databuss for sending av sensor data Unik for hver sensor
VPN	Virtual Private Network	Virtelt Privat Nettverk: kryptert forbindelse over internett

1 Innledning

Innledningen gir leser en forståelse for hva oppgaven går ut på. Først presenteres vår oppdragsgiver og oppgaven. Deretter er problemsstillingen presentert, etterfulgt av bedriftens hensikt med prosjektet. Til slutt kan en lese om våre mål for oppgaven.

1.1 Oppdragsgiver

Vår oppdragsgiver er Cognite AS. Det er et datterselskap av Aker Capital AS, og har hovedfokus på å samle, behandle og analysere data i industrien. De har en visjon om å digitalisere den industrielle verden ved hjelp av deres nyutviklede programvare Cognite Data Fusion(CDF)[1]. CDF er en plattform hvor en løpende samler all relevant data på samme plass, slik at den er lett tilgjengelig og forståelig.
Cognite ble grunnlagt i 2016 for å møte kravene for industriell digitalisering.

1.2 Oppgaveforklaring

Denne rapporten omhandler vår bacheloroppgave, og er en del av et større prosjekt. Oppgavebeskrivelsen vi fikk fra Cognite beskriver bakgrunnen og grunnlaget for prosjektet, og var som følger:

"*Cognite AS har et eksisterende engasjement med Kværner Stord, hvor vi har tracket understøttelser og containere med roboten Spot, som vist i denne videoen [2]. Dette prosjektet bygger videre på disse resultatene ved å sette en robot i ukentlig drift ved Kværner Stord for å kartlegge verftsområdet.*"

I figur 1.1 kan dere se bilde fra Spot-prosjektet. Den viser Boston Dynamics Spot, den går forbi en understøttelse med ID-markør, av typen AprilTag.

Opprinnelig var vårt hovedfokus å kjøre roboten på Kværner Stord sitt verft for å samle et datagrunnlag. Dette skulle brukes i utviklingen av en autonom ressurssporings løsning. Med autonom ressurssporing mener vi en robot som kjører rundt av seg selv, og lokaliserer ulike ressurser på verftets område. Noen eksempel på ressurser er: containere, understøttelser, tilhengere osv.

Figur 1.1: Illustrasjonsfoto fra Cognite video[2]

Cognites prosjekt skal benytte en hjul-basert robot med flere kamera og sensorer for å hente inn dataen som trengs. Vi skal behandle bilder og sensor data fra roboten, og lagre det som et datasett. Datasettet skal så lastes opp til CDF for videre behandling og enklere tilgang for andre applikasjoner.

På grunn av forsinkelser ble ikke roboten klar for kjøring innen planlagt tid. En oppblussing av COVID-19 førte også til at verftet på Stord stengte for eksternt personell. Denne kombinasjonen førte til at vi ikke fikk kjørt roboten, som opprinnelig var planlagt. Resultatet ble at vi endret hovedfokus fra datainnhenting, til databehandling og opplasting til CDF. Hovedsakelig har vi fokusert på data fra bilder, Global Positioning System(GPS) og Inertial Measurement Unit(IMU), samt en generell løsning for andre datatyper.

1.3 Problemstilling

Opprinnelig var mye av fokuset basert på kjøring av Clearpath Jackal og datainnhenting på Stord. Da var problemstillingen som vist:

"*Kværner Stord har behov for en autonom sporingsløsning på sitt verft. Dette er fordi dagens manuelle løsning krever mye tid og ressurser. Løsningen er mobil robot som skal kjøre regelmessige runder på verftet. Vår jobb er å hente ut data fra roboten, og laste det opp til CDF. Hvordan skal dette gjennomføres?*"

Som nevnt tidligere fikk vi ikke adgang til verftet. Fokuset i oppgaven ble da naturlig flyttet over til databehandling og opplasting. Derfor har vi også utarbeidet en ny problemstilling. Den reflekterer endringen i prosjektet og arbeidet som er utført.

"*Cognite AS skal lage en autonom robot for kartlegging av ressurser på norske industri områder. I denne sammenheng er det behov for løsninger innen databehandling og dataopplasting. Hvordan skal informasjonen fra roboten lagres og formateres, slik at den kan lastes opp til Cognite Data Fusion for videre analyse?*"

1.4 Bedriftens hensikt med prosjektet

Bedriftens hensikt med prosjektet er å samle et sterkt datagrunnlag, og demonstrere verdiuttak for Kværner Stord. Dette vil inngå i en planlagt publikasjon sammen med veileder i bedrift, Johan Hatleskog, og masterstudent Kevin Kaldvansvik ved NTNU.

Verftet på Stord er relativt stort og uoversiktlig, med verdifulle ressurser over hele området. Per dags dato er det ingen som har full oversikt over hvor alle de forskjellige ressursene er plassert. Dersom en har behov for en eksakt ressurs må en få tak i noen som vet hvor de er, for så å hente den. Når alle må gjøre dette krever det mye tid, dette er tid som kunne blitt brukt på mer verdifullt arbeid.

Et problem ved verftet er at en ressurs blir feilplassert eller kommer på avveie. Skjer dette må man ut å lete etter det, og notere faktisk posisjon. Tanken med prosjektet er å fjerne behovet for denne arbeidsoppgaven, med en robot i konstant drift for kartlegging og registrering av ressurser. Her vil vi benytte en mobil robot med kamera for å lokalisere identifiserende markører som plasseres på ønskede ressurser. Markøren, kombinert med GPS og robotens interne data skal brukes for å gi et mål på koordinatene til ressursen. Denne informasjonen lastes opp til CDF, hvor den skal brukes til å lage et digital kart over alle de tilgjengelige ressursene. Kartet skal være tilgjengelig for alle arbeidere gjennom en mobil app, og skal gi en bedre flyt i det daglige arbeidet på verftet.

1.5 Mål for hovedoppgaven

Etter endringene som førte til at fokuset ble databehandling og dataopplasting, ble målet vårt og kunne tilby en brukervennlig og robust løsning til vår oppdragsgiver. Håpet er at

dette kan brukes videre i det endelige produktet.

Videre ønsker vi å få innsyn og erfaring i dagens skyløsninger i industrien. Vi håper også å utvikle våre programmeringsferdigheter, og gjerne lære oss et nytt programmeringsspråk.

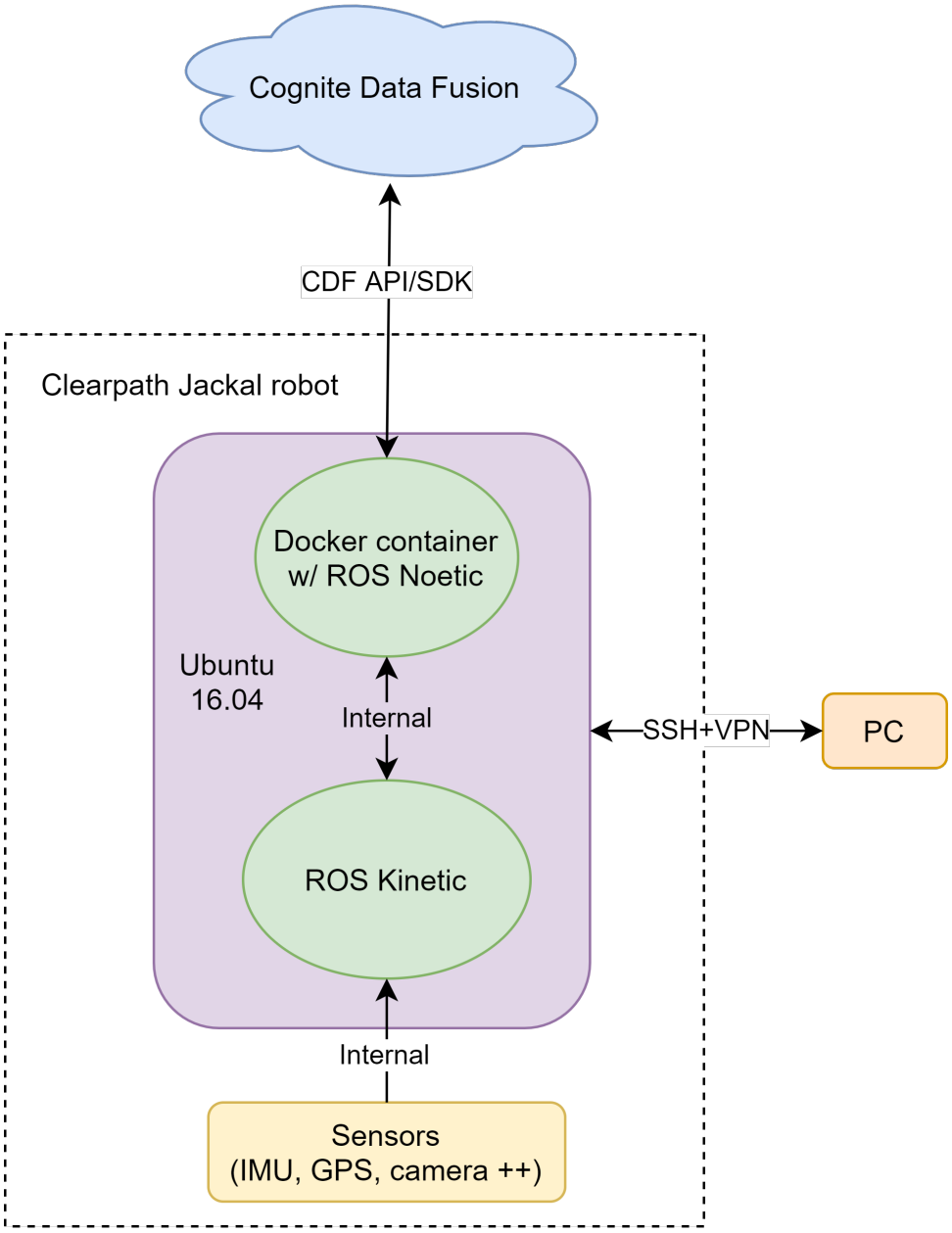
2 Kravspesifikasjon

Kravene vi har fått fra oppdragsgiver vedrørende oppgaven er:

- Behandle data og programvare som blir laget etter oppgitt taushetsavtale.
- Oppgaven skal løses ved å benytte en hjul-basert robot. Den skal kjøres i ett industriområde, og detektere markerte ressurser som skal kartlegges.
- Informasjonen fra sensorene samles i en datapakke, som lastes opp til CDF.
- Filene må være formatert på en slik måte at den kan brukes i andre prosjekter i CDF.
- Kameraene til Jackal må kalibreres.
- Bilder av containere med digital tag.
- Data som skal hentes inn:
  - Video og bilde fra dybdekamera og 360 kamera
  - GPS lokasjon
  - IMU data

3 Teori

Her vil vi kort presentere teori om ulike element som er relevant for oppgaven. Som nevnt i kapittel 1.5 vil vi utvikle oss i programmering og databehandling. Derfor presenterer vi konsepter og løsninger som er relevante for de temaene. I tillegg kommer teori om element som er viktige for oppgaven. I figur 3.1 kan en se en skisse av hvordan noen av elementen i teorien henger sammen.



Figur 3.1: Kommunikasjons skisse for prosjekt. Viser også hvor noe av teorien som blir presentert er brukt.

3.1 Software

Operativsystem

Ubuntu er et operativsystem (OS) basert på Linux. Det er en populær og vidt støttet linuxdistribusjon. Det finnes mange utgaver, de mest brukte er 16.04, 20.04 og 22.04. Det kommer jevnlig oppdateringer, men innimellom har de versjoner som garanterer langtidsstøtte for sikkerhetsoppdateringer og vedlikehold. For 16.04 ender denne støtteperioden juni 2021.

Secure Secret Shell

Secure Secret Shell (SSH) er et dataprogram og en nettverksprotokoll. Det blir brukt til å kommunisere med en annen datamaskin gjennom kommandovinduet. Dette gjør det mulig å styre en annen datamaskin over nett. Kommunikasjonen mellom maskinene blir kryptert. For et ekstra lag med sikkerhet kan en bruke et Virtual Private Network (VPN), det oppretter en sikker "tunell" for informasjonen som blir sendt. SSH blir brukt til kommunikasjon mellom datamaskin og robot.

Programmeringsspråk

Et programmeringsspråk omhandler ulike kommandoer som får datamaskinen til å utføre en bestemt oppgave. Syntaksen mellom forskjellige programmeringsspråk er unik og ett minste avvik fra "grammatikken" kan gjøre at programmet feiler. Eksempel på ulike programmeringsspråk er: Python, C, C++, Java, og mange flere. Python er et høynivå tekstbasert programmeringsspråk. Det er utviklet som et visuelt, ryddig og leselig språk. Det støtter bruk av ferdige bibliotek og pakker, noe som gjør det enkelt å gjenbruke kode.

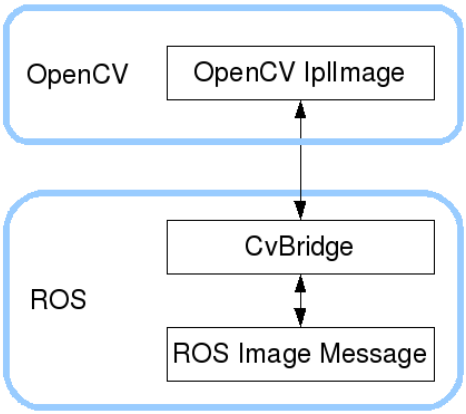
Robot Operating System

Robot Operating System (ROS) er et rammeverk for programmering og utvikling av roboter. Det er ett av de mest populære verktøyene for robot utviklere, og tillater stor variasjon i kompleksitet og funksjonalitet. Det baserer seg på ett nettverk av noder og topics, se figur 3.2. Nodene sender og mottar data over ett eller flere topics, og kan jobbe uavhengig av hverandre. Topics kommuniserer hovedsakelig over TCP/IP-protokollen, en

av de mest brukte kommunikasjonsprotokollene for nettverkskommunikasjon. Den benytter tilbakemelding, som betyr at mottaker informerer sender om at dataen er mottatt, dette hjelper på pålitelighet og nøyaktig kommunikasjon.

Figur 3.2: Hovedprinsippet til ROS. Hver node er en uavhengig prosess. Publisher sender informasjon over topics, subscriber mottar informasjon fra ett eller flere topic [3].

For å hente ut bilder fra ROS brukes CvBridge. Det er et bibliotek som gir brukeren et grensesnitt mellom ROS og OpenCV (figur 3.3). OpenCV er et bibliotek for behandling av bilder for computer vision.



Figur 3.3: Kommunikasjonsprinsipp for CvBridge. Lager en bro mellom ROS og OpenCV [4].

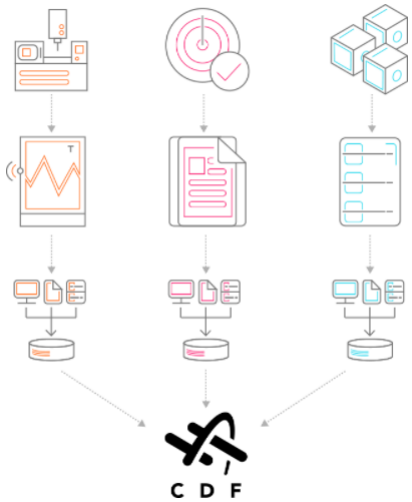
Rosbag er en funksjon i ROS for logging av sensor data under drift. Den tillater en å synkronisere dataflyten til alle sensorene i en felles tidslinje. Dette gjør det mulig å analysere informasjonen i samme tidspunkt.

En Rosservice er en annen måte noder kan kommuniserer med hverandre på. Servicen tillater noder å sende en forespørsel og motta svar. Den består av tre deler: En `srv` fil som deklarerer input og output for servicen, de to andre er client og server script. Server scriptet er hoved koden til servicen, det er et Python script som kjører kontinuerlig i bakgrunnen.

Man kan bruke Client for å kalle serveren, eller kalle den direkte med `rosservice call`.

Cognite Data Fusion

CDF er et produkt fra Cognite AS. Det er en kombinert database og databehandlingspakke, som lar bedrifter utnytte data og informasjon de allerede har i en mye større grad. Det inkluderer prinsipper som digital tvilling, maskinlæring, nevrale nettverk og lignende. Det er utarbeidet for å samle all relevant data på en plass (skisse vist i figur: 3.4). Det gjør det mulig å koble sammen målinger, utstyr, historikk osv. Da kan en se helheten av et anlegg basert på målingene man tar. Det kan og brukes for historikk og analyse av utstyr, som kan bidra til planleggingen av forebyggende vedlikehold.



Figur 3.4: CDF prinsipp: Samler data fra forskjellige systemer og prosesser i en felles database. Kan utføre databehandling på flere nivå, og generere statistikk og informasjon om systemets helhet. [5]

Docker container

En Docker container er et databehandlingsmiljø, den kan jobbe på tvers av plattformer og programvare-versjoner. Det vil si at den løser problematikken med OS begrensninger på versjoner som støttes. For eksempel: Dersom et datamiljø kun støtter kjøring av Python 2 og en har behov for å kjøre en Python 3 versjon. Da kan en lage en Docker container på datamaskinen der Python 3 kan kjøre. For mer informasjon se [6].

Data-annotering

Data-annotering er kategorisering og markering av data for AI applikasjoner, primært brukt for objekt deteksjon i sammenheng med computer vision. Eksempelvis kan en definere "sann" posisjon til et objekt i et bilde.

Mennesker har egenskapen til å identifisere et objekt i et bilde, for eksempel kan vi si at figur 3.5 inneholder biler og trafikklys, og en kan si hvor de er i bildet. Dette kan ikke en datamaskin gjøre, hvert fall ikke før den lærer å gjøre det. Maskinlæring baserer seg på at maskinen selv lærer seg å gjenkjenne disse objektene, men da må man gi den opplæringsbilder med informasjon om hvor og hva som er i bildet. For at algoritmen skal være robust og klare å detektere forskjellige objekt av samme klassifisering trengs det et bredt datasett. Dette datasettet må lages manuelt, det gjør en ved å ta bilder av flere forskjellige objekter for så å annotere "sann" lokasjon på hvor objektet er i bildet.

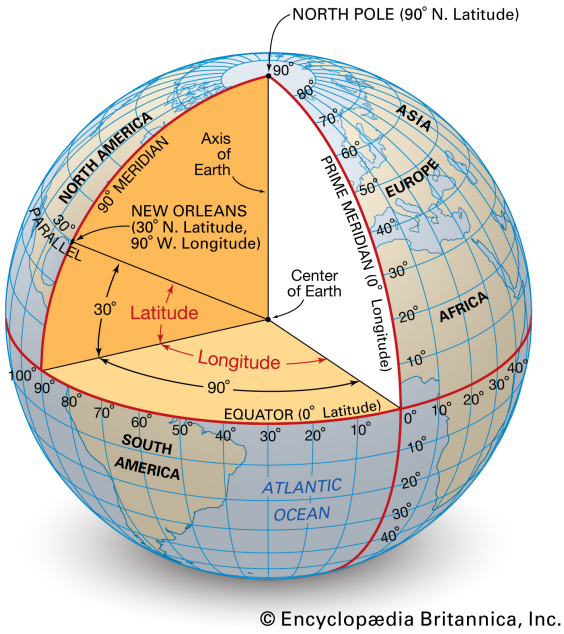
Figur 3.5: Data annotasjon med Bounding boxes [7]. Viser hvordan objekt blir markert i bilde.

Det er flere forskjellige data annotasjons metoder, en av de vanligste er Bounding boxes. Den fungerer ved at en tegner en rektangulær boks over lokasjonen til de ønskede objektet, Da blir X og Y koordinatene til det venstre øverste hjørne lagret samt koordinatene til det nedre høyre hjørnet av rektangelet. Dette vil representer hvor objektet er lokalisert i bildet, slik at algoritmen kan identifisere og huske figuren som er markert. Det er viktig at man dekker hele objektet med den rektangulære boksen. Et eksempel på hvordan Bounding boxes metoden markerer objektene er vist i figur 3.5, dette er ved annotasjon for opplæring av autonome kjøretøy.

3.2 Sensorikk

Global Positioning System

GPS (Global Positioning System) er et navigasjonssystem som benytter seg av kommunikasjon med satellitter for å fastslå nøyaktig posisjon. Kort forklart fungerer dette ved at man måler avstanden fra minimum fire satellitter samtidig. Avstandsmålingen blir beregnet med tiden radiosignalet bruker fra satellitten til mottakeren. Det er derfor svært viktig at en bruker en felles tidsreferanse for at avstandsmålingene skal bli rett. Den fjerde satellitt sørger for at mottaker og satellittene bruker samme tidsreferanse. Er posisjonen og avstanden mellom satellittene kjent har vi nok informasjon til å fastslå nøyaktig posisjon til mottaker.

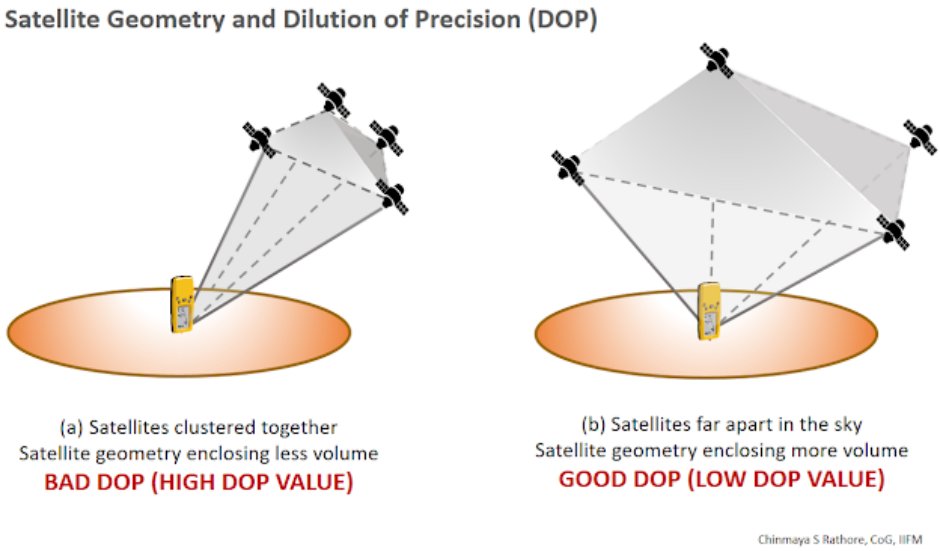


Figur 3.6: Latitude og longitude skissert på jordkloden[8]

To vanlige format for GPS-data er desimalgrader og NMEA. Desimalgrader er på grad format, og kan se slik ut: (38.8897°, -77.0089°). NMEA setning er mer kompleks, men inneholder med informasjon. For mer info se [9].

Generelt består et GPS-koordinat av breddegrad, lengdegrad og høyde. Breddegrad (latitude) er målet fra nord til sør. Lengdegrad (Longtitude) er målet fra øst til vest. Høyde (Altitude) er et mål på høyden ut fra jordkloden i meter. Se figur 3.6 for skisse av jordkloden med latitude og longitude.

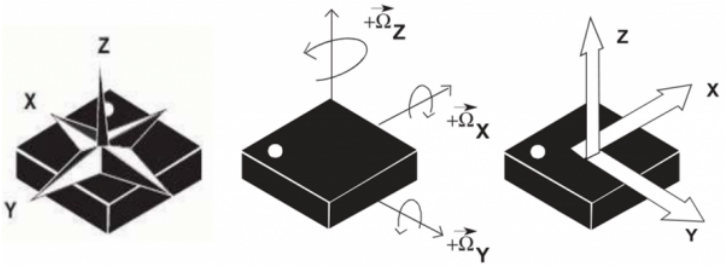
Fra en GPS sensor får en mål på latitude, longitude og altitude for å fastslå posisjonen. En får også en usikkerhet knyttet til målingene. Posisjon Covariance (heretter kovarians) er et mål på usikkerheten til målingene. For GPS er en type kovarians "Dilution of Precision Errors"(DOP), det er målt basert på avstanden mellom satellittene. Som en kan se i figur 3.7 vil en større avstand mellom satellittene gi bedre triangulering, som igjen gir mindre usikkerhet i målingen, dette representeres med en lav DOP verdi.



Figur 3.7: Usikkerhet i GPS måling basert på DOP verdi. Større avstand mellom satellittene gir en bedre måling. [10]

Inertial measurement unit

IMU (Inertial measurement unit) er en elektronisk enhet som måler kraft, vinkelhastighet og orientering. Dette gjøres ved hjelp av akselerometer, magnetometer og gyroskop. I figur 3.8 er hver sensor i en IMU representert.



Figur 3.8: Skisse på hva en IMU sensor inneholder med aksene til målingen. Fra venstre: magnetometer, gyroskop og akselerometer [11]

Måleenheten til magnetometeret er oppgitt i Gauss(G), den måler magnetfelt og

representerer orienteringen til sensoren. Akselerometer målingen blir oppgitt i G-krefter(g), der 1 g er 9.81 m/s^2. Måleenheten til gyroskopet er grader per sekund (°/s) og representerer vinkelhastighet. Hver måling som blir tatt representeres i et kartesisk koordinatsystem, med verdier i x, y og z retning.

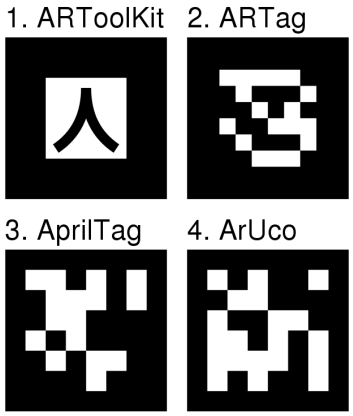
Kamera

Innen robotikk og computer vision brukes det ofte stereokamera, det er kamera med flere sensorer, som kan ta RGB bilde med pixel-verdier fra 0-255, dybdebilder og mer. Et dybdebilde gir oss informasjon om avstanden til objektene en tar bilde av. Samler en denne informasjonen tillater det oss å gjenskape et bilde i en 3D-representasjon, som gir mye mer detaljer og info enn et tradisjonelt bilde.

Et av de mest populære og anerkjente dybdekameraene er Intel RealSense. De har en integrert prosessor som ufører dybdeberegningene. Intern kalibrering sørger for nøyaktige bilder, og en open-source dokumentasjon gjør det enkelt å jobbe med. Kamera vist i figur 5.3 er en Intel RealSense D435.

3.3 Fiducial markør

En fiducial markør blir brukt som referanse i et bilde. Som regel er det for å få informasjon om kameraets positur(pose) i forhold til markøren, det vil si et mål på posisjon og orienteringen til markøren fra kameraet. De brukes også som en identifikator, da er hver tag unik og vil representere en ID. Det finnes forskjellige fiducial markører for computer vision, noen eksempel er: ARToolKit, ARTag, AprilTag og ArUco. Disse er vist i figur 3.9.

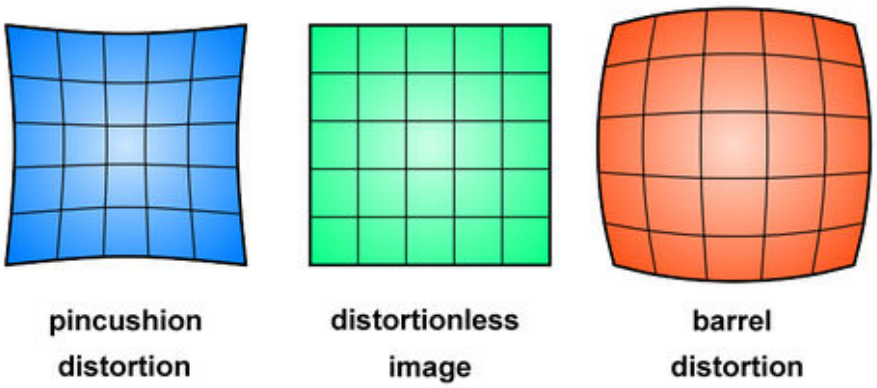


Figur 3.9: Eksempel bilde på diverse fiducial markører. [12]

AprilTag er designet for enklere og mer robust deteksjon, der synsvinkel og lysstyrke kan variere. Den er laget for å kunne gi god lokaliseringsnøyaktighet. Computer vision og deteksjonsalgoritmer brukes for å finne taggen i et bilde. Den kan så analyseres for å hente ut ID-nummer og plasseringen til taggen.

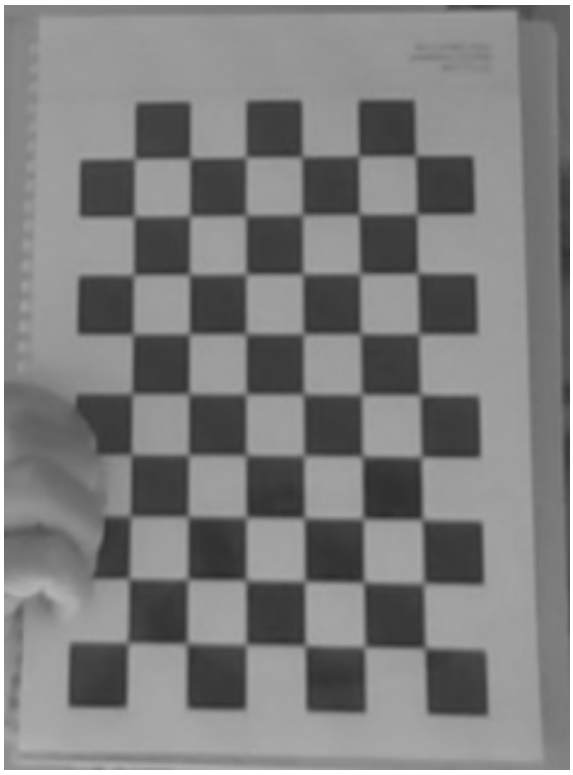
3.4 Kamerakalibrering

Dagens kamera kan være litt unøyaktige og dette kan skape forvrengninger av bilde. Det er i hovedsak to forskjellige forvrengninger: radial forvrengning (barrel-distortion) og tangential forvrengning (pincushion-distortion). Den radiale forvrengingen gjør at de rette linjene vil virke buede på bilde. Tangensial forvrengning oppstår når kameralinsen ikke er perfekt justert. Resultatet av tangential forvrengning er at noen områder i bilde kan se nærmere ut enn det faktisk er. I figur 3.10 ser man hvordan forskjellige deler av bilde forskyves. Dette kan rettes opp ved kamerakalibrering.

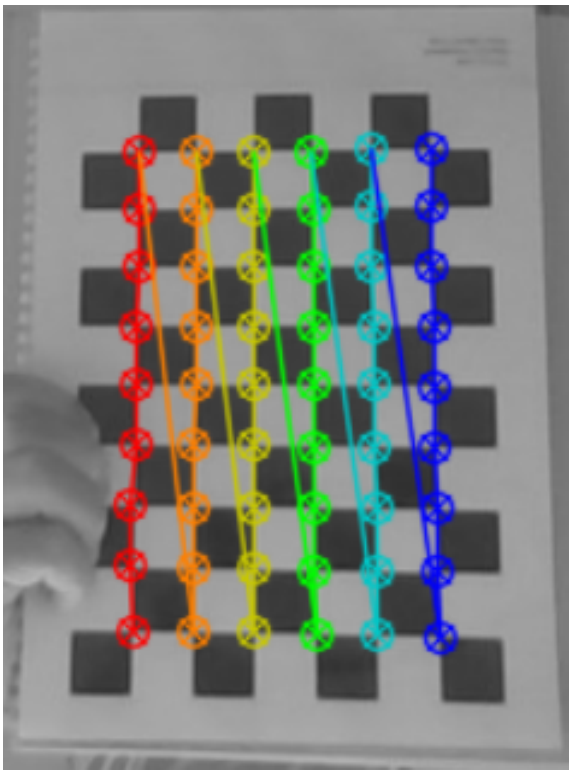


Figur 3.10: Eksempel på tangential og radial forvrengning[13]

Det er flere forskjellige kalibreringsmetoder, metoden som blir presentert videre heter Zhangs metode [14]. Denne benytter en flat overflate med rutemønster som kalibreringsbilde. Eksempel på et kalibreringsbilde er vist i figur bilde 3.11a, i figur 3.11b kan en se bilde fra hvordan datamaskin detekterer hjørnene i rutemønsteret. Selve kalibreringen kan gjøres på flere vis, en populær metode er å lage et program med Python og biblioteket OpenCV.



(a) Rutemønster for kamera kalibrering kalibrering



(b) Rutemønster med hjørnedeteksjon i Python

Figur 3.11: Rutemønster for kamerakalibrering. Denne er 9x6 rutemønster og er basert på å detektere hjørnene til rutene.

Overføringen av et bilde fra den reelle verden til et 2D plan foregår slik:

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = KR \begin{bmatrix} I_3 & | & -X_O \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{3.1}$$

Der

$$K = \begin{bmatrix} fs_x & fs_\theta & u_c \\ 0 & fs_y & v_c \\ 0 & 0 & 1 \end{bmatrix}, R \begin{bmatrix} I_3 & | & -X_O \end{bmatrix} = \left[\begin{array}{ccc|c} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{array}\right] \tag{3.2}$$

R er en rotasjonsmatrise og viser til vinklingen til kamera, X_O er en transformasjonsmatrise og viser til kameraets posisjon, den er negativ for å ta hensyn til kameraets faktiske senter. K er kalibreringsmatrisen, den inneholder kamerakonstanter og linseverdier. Konstanten

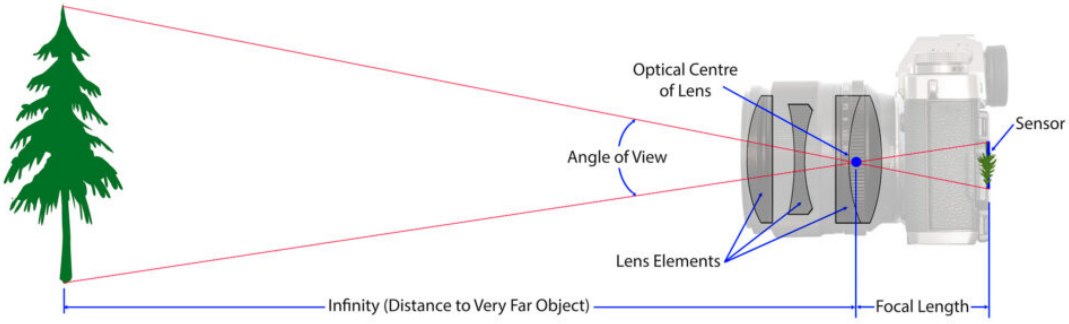
s_θ representerer diagonal forvrenging, og er generelt sett tilnærmet lik null[14, s. 4]. λ er en skaleringsfaktor. I utregningene settes denne til 1.

Et kamera har både indre og ytre parameter. De indre parameterne som en får ved kalibrering er vist i kameramatrisen 3.3. OpenCV setter s_θ til null, ettersom den har et neglisjerbart utslag på resultatet.

$$Kameramatrise = \begin{bmatrix} fs_x & 0 & u_c \\ 0 & fs_y & v_c \\ 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

Her vil fs_x og fs_y vise oss informasjon om brennvidde (Focal length). Dette er et indre mål på lengden fra optisk senter til bildesensor. Det optiske senter er punktet på kameralinsen som alle stråler blir samlet i, for så bli reflektert til bildesensor. Bildesensoren omformer lysstråler til elektriske signaler.

u_c og v_c er to indre parametre som en får ut fra kalibreringen, de representere optisk senter. Ideelt sett bør dette punktet være nært sentrum av bilde, for å maksimere synsfeltet. Realiteten er ikke slik, men ved kalibrering av det optiske sentere finjusteres posisjonsplasseringen. I figur 3.12 er blant annet brennvidden og optisk senter vist.



Figur 3.12: Diagram som representere de forskjellige målene en kameralinse bruker.[15]

Kameramatrisen benyttes for å fjerne forvrengning som oppstår i bilde på grunn av kameraets oppbygging. Hvert enkelt kamera er forskjellig, og en trenger en tilpasset matrise for å oppnå best mulig resultat.

Radial distortion

$$x_{corr} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{3.4}$$

$$y_{corr} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{3.5}$$

Tangential distortion

$$x_{corr} = x + [2p_1 xy + p_2(r^2 + 2x^2)] \tag{3.6}$$

$$y_{corr} = y + [p_1(r^2 + 2y^2) + 2p_2 xy] \tag{3.7}$$

Der x og y er koordinatet til pikselen som skal korrigeres, og r er distansen fra origo. Fra Kamerakalibreringen får en et mål på forvrengningen i bildene. Forvrengning-parametere (distortion coefficient) representeres som en vektor i OpenCV og ser slik ut:

$$distCoeffs = \begin{bmatrix} k_1 & k_2 & p_1 & p_2 & k_3 \end{bmatrix} \tag{3.8}$$

Kameramatrisen og distorsjonskoeffisientene brukes i computer vision for å få mer nøyaktig resultat. De kan også brukes til å justere pikslene i bilde for å fjerne forvrengningen. Da vil bilde være mer "likt" virkeligheten.

4 Løsning

I dette kapitelet vil en kunne lese om hvilke valg som gjort, og hvordan Jackal er utstyrt. Deretter kommer en oversikt over hvordan de forskjellige elementene i oppgaven henger sammen. Videre er programvaren vi har brukt presentert. Til slutt kan en lese om hvilke vurderinger som er gjort for oppgaven.

4.1 Clearpath Jackal

Løsningen til Cognite er en mobil robot, de har valgt å bruke Clearpath Robotics Jackal. Den er robust og enkel å jobbe med, og kan utvides med sensorer og utstyr etter behov.

Dette prosjektet bruker følgene konfigurasjon:

- Kommunikasjon
  - Teltonika Networks RUT240 Router (for 4G)
  - Integrert WiFi & Bluetooth
- Data & OS
  - Arduino (for kontroll av lys)
  - Ubuntu 16.04
  - ROS Kinetic
- Kamera
  - Intel Realsense D435
  - Hikvision Mini PTZ camera - Model: DS-2DE3304W-DE
  - Ricoh Theta V - 360 kamera
- GPS
  - Ublox RTK GPS med TCP-tilkobling til Statens Kartverk for RTK korreksjoner
  - Integrert GPS fra Clearpath
- IMU Integrert i Jackal: Phidget Spacial 3/3/3

– Gyroskop, Akselerometer og magnetometer

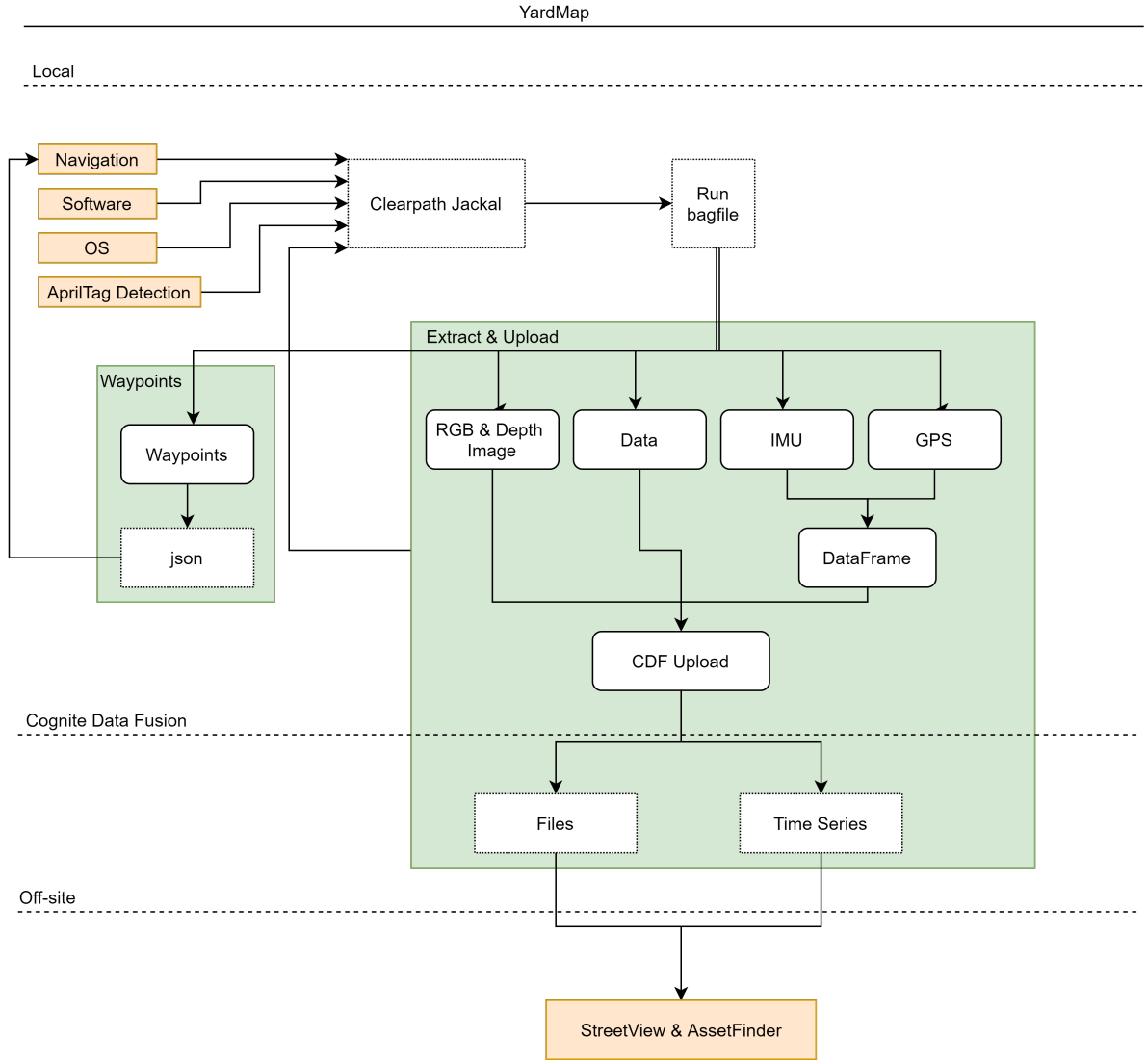
Kameraet vi i hovedsak skal benytte er Intel RealSense D435 som vist i figur 5.3. Det er montert to slike kamera på roboten, ett helt fremme nede og ett en meter over bakken som er vendt mot høyre. Dette kan en se i figur 4.1, som viser Clearpath Jackal med utstyr.

Figur 4.1: Clearpath Jackal utstyrt av Cognite AS

4.2 Oppgaveoversikt

En oversikt over programvaren til Clearpath Jackal kan sees i figur 4.2. Her er de forskjellige systemene og programmene separerte, og sammenhengen er vist. Vår del av prosjektet er innrammet i grønt, og omhandler databehandling og opplastingen. Det er en sekvensiell løsning, som gjennomfører et sett med funksjoner for å tilrettelegge for filopplasting til CDF. Ekskludert fra CDF har vi laget en rosservice for uthenting av waypoints fra

GPS-data. Den skal brukes for autonom navigering med Jackal.



Figur 4.2: Generell oversikt over elementene knyttet til prosjektet. Det innrammet i grønt er vårt fokus, og er det vi vil finne en løsning på.

4.3 Programvare

For å utvikle og teste programvare som kjøres på Clearpath Jackal har vi installert Linux på vår datamaskin. Vi bruker Ubuntu 16.04 med ROS Kinetic, og Ubuntu 20.04 med ROS Noetic.

Clearpath Jackal er utviklet med tanke på Ubuntu 16.04. Det betyr at vi primært må jobbe med ROS Kinetic, ettersom det er ROS versjonen som tilhører 16.04. Derfor må vi også bruke Python 2, da Kinetic ikke støtter Python 3. Her støtte vi på et problem, ettersom Cognites CDF-SDK(Software Development Kit) ikke er bakover-kompatibel med

Python 2, spesifikt krever det Python 3.5 eller høyere [16].

En løsning er å kjøre en Docker-container med Ubuntu 20.04 og ROS Noetic, som støtter Python 3. Docker tillater oss å simulere et programvare-miljø i flere forskjellige OS. I dette tilfelle vil vi simulere en Ubuntu 20.04 container på en datamaskin som kjører Ubuntu 16.04. Da kan vi få tilgang til alle filene som trengs, og samtidig kjøre Python 3 programmene. Se figur 3.1 for en grafisk fremvisning av løsningen.

Data som samles fra sensorene til Clearpath Jackal lagres i en `rosbag`. Det er en proprietær filtype fra ROS, og kan hovedsakelig kun brukes i et ROS-miljø. Vår løsning er da å hente ut informasjonen gjennom et program som kjører på Jackal, for så å skrive den til konvensjonelle filtyper (`jpg, csv, npy`) som lagres lokalt. Derfra er det enklere å laste filene opp til CDF for videre bruk.

4.4 Vurderinger

Vi bruker CDF, da det er Cognites egne skyløsning, og et av målene med prosjektet er å fremheve det. Vi er også nødt til å bruke Ubuntu 16.04 med ROS Kinetic, ettersom Clearpath Jackal er basert på det. ROS er også industristandard for robotikk-prosjekter, så det er et naturlig valg. Cognite har CDF-SDK tilgjengelig i JavaScript og Python. Her valgte vi Python, ettersom det var den løsningen som våre kontaktpersoner i Cognite hadde kjennskap til og anbefalte. Det er også språket som ble brukt tidligere i prosjektet.

Koden kunne vært skrevet i flere forskjellige redigeringsprogram. Programvaren vi har valgt å bruke er Visual Studio Code (VS Code). Det er en gratis programvare som vi hadde kjennskap til fra før. Vi valgte VS Code over andre programmer, da Code er en "lettvekter" sammenlignet med disse. Samtidig har det funksjonalitet som feilsøking, syntaksfremheving, kodefullføring og integrering med GitHub.

Det er benyttet OpenCV bibliotek for Python i forbindelse med kamerakalibrering og bildebehandling. Det er et populært verktøy for bildebehandling, og har god dokumentasjon og eksempelkode.

5 Metode

I dette kapittelet kan en lese om datainnsamling og databehandling. I datainnsamling kan du lese om kjøring av Jackal og dataflyt til CDF. Så presenteres ROS og simulering, før vi skriver om de forskjellige datatypene.

Databehandlingen inneholder utpakking av data fra `bag` filer og opplasting til CDF. Deretter presenterer vi arbeidet med kamerakalibrering og data-annotering.

5.1 Datainnsamling

5.1.1 Kjøring av Clearpath Jackal

Det ble ikke testkjøring på Stord, som opprinnelig var planen, men det er allikevel utarbeidet en plan for hvordan testingen kan foregå.

Før man starter selve kjøringen bør man filme en liten snutt med rutemønsteret i fokus, i forskjellige positurer. Dette er for bruk i kamerakalibrering. Se kapitel 5.2.3 for mer informasjon. Identifikasjonsmarkøren som skal benyttes i prosjektet heter AprilTag. Ved deteksjon av AprilTag kan algoritmen ta forbehold om feilmargin, dette kan være til hjelp for å oppnå nøyaktig deteksjon i situasjoner som ikke er optimale. Siden roboten skal operere utendørs, under varierende værforhold vil dette bidra til en mer robust løsning.

Datainnsamlingen på verftet kan foregå under manuell eller autonom kjøring. Her har vi presentert noen eksempel på ruter, og hvilken informasjon vi vil få ut av dem.

Manuell kjøring

Langs rute: Styre Clearpath Jackal med joystick, kjører langs en rute der det er plassert AprilTags på diverse ressurser. Tanken er at dette vil gi et syn på hvordan avstand påvirker deteksjon av AprilTags. Vi ville også sett på eventuelle dødsoner, og hvilken innvirkning det vil ha.

Rundt container eller understøttelse: Kjøre Clearpath Jackal rundt en ressurs med AprilTags i forskjellige høyder. Her ønsket vi å se på hvor stort synsfelt Intel RealSense D435 har, og hvordan høyde påvirker forvrengningen til kameraet.

Autonom kjøring

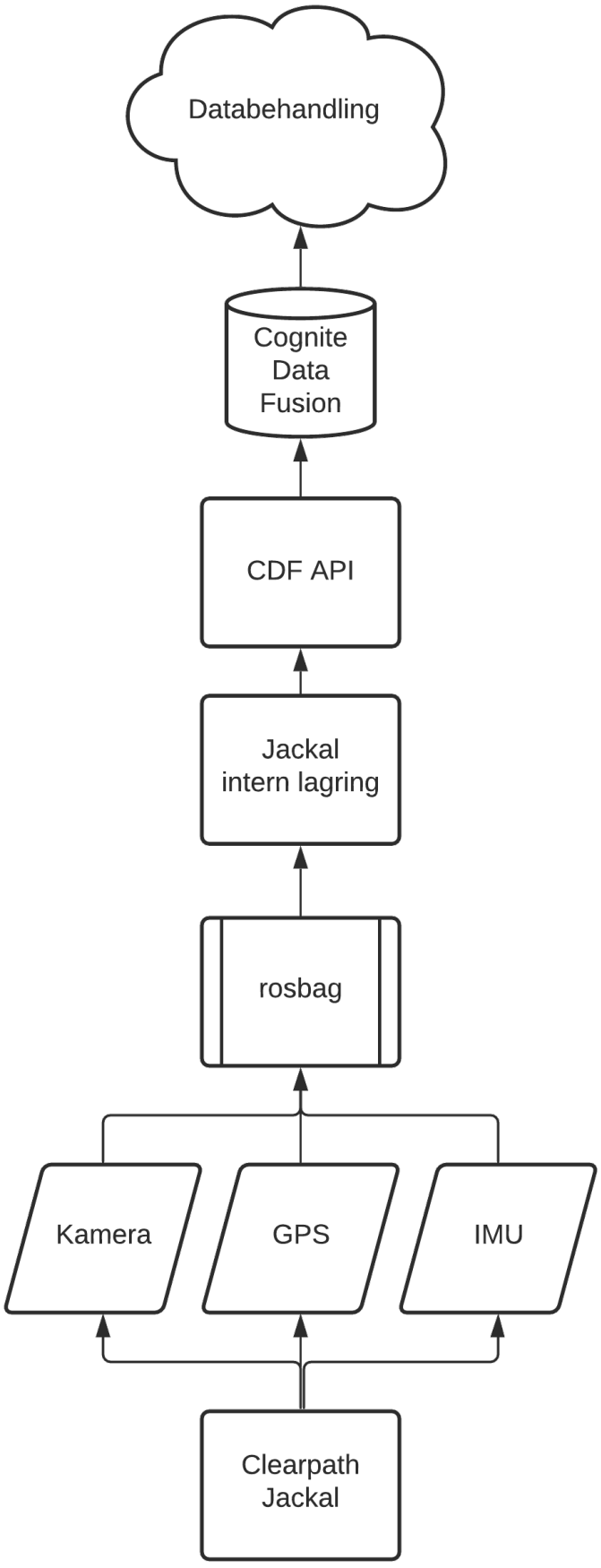
Jackal settes opp for autonom drift. Det plasseres ut AprilTags langs den mulige ruten. Dersom Jackal hadde blitt klar for autonom kjøring ville det vært interessant å se hvordan den klarer seg selv, og notere hva den gjør annerledes enn ved manuell kjøring.

5.1.2 Dataflyt

Dataflyten fra Clearpath Jackal til CDF er vist på figur 5.1. De forskjellige punktene er kort forklart under. Kommunikasjon med roboten går gjennom en SSH-tilkobling, denne er sikret gjennom en VPN.

- Clearpath Jackal: Den mobile robot som blir benyttet i prosjektet.
- Sensorikk: Alle sensorene er tilkoblet Jackal, gjennom ROS får vi tilgang til dataen de sender og mottar.
- rosbag: En funksjon i ROS som logger data i på en felles tidslinje. Den "abonnerer" på de sensorene man er interessert i, og lagrer alt i en `bag` fil.
- Jackals intern lagring: rosbagen blir lagret og pakket ut på Jackals interne harddisk.
- CDF API: API'en gir oss tilgang til CDF og funksjonene vi ønsker å benytte. Det inkluderer opplasting av filer, og oppdatering av timeseries. Tilgang til CDF på Jackal skjer gjennom en docker container.
- Cognite Data Fusion: Databehandlingssystemet som skal brukes for videre utvikling.
- Databehandling: Her sammensettes informasjonen til et helhetlig produkt.

Hvor hyppig en lagrer data fra sensorene kan variere. Det blir i området millisekund til sekund. Kamera har en hastighet på ca. 30Hz, GPS ligger mellom 1 og 10Hz, og IMU ligger på 100Hz og oppover. Fra dette ser vi at vi vil logge video og GPS så hurtig som mulig, men med IMU kan det være gunstig å ikke ta alle signalene. Under testing har vi valgt å logge alt, da det er den enkleste måte, og vi var interessert i å samle så mye data som mulig.



Figur 5.1: Dataflyt fra robot til CDF

5.1.3 rosbag

ROS kan brukes til å logge alle topics som kjøres under drift. Disse lagres i en `bag` fil. Denne må behandles innad i ROS miljøet for å hente ut nødvendig data. En kan velge om man vil logge all tilgjengelig data, eller spesifikke topic.

```
# records all available data
rosbag record -a
# records two given topics
rosbag record /realsense/color/image_raw /realsense/depth/image_rect_raw
```

rosbag har tilgang til alle topics som publiseres i ROS. Vi er hovedsakelig interessert i /realsense/color/image_raw , realsense/depth/image_rect_raw , /navsat/fix/ og /imu/data. Det er Fargebilde, Dybdebilde, GPS og IMU, respektivt.

Programmet vi har laget henter også ut alle andre topics fra en rosbag Disse blir lagret til generiske `csv` filer med et standardisert script. Ett problem med dette er håndteringen av store mengder data, spesielt bilder og punktskyer (dybdeinformasjon fra RealSense). Om denne dataen blir lagret som `csv` blir størrelsen over 100x større enn selve bildet. ROS vet ikke hvilket filformat som passer best til enhver topic. Vår løsning er å lage en liste med alle topics som ikke skal skrives til fil. Denne er formatert som en `yaml` fil, se appendiks A6 for eksempel. Et alternativ hadde vært å analysere hvert topic, for å finne ut hva det er, men det ville medført en kompleksitet i programmet som vi ikke ønsket. Det ville også redusert effektiviteten til programmet drastisk.

5.1.4 Simulering i Gazebo

Gazebo er et simuleringsverktøy som er integrert i ROS. Det gjør det mulig å simulere en hel del forskjellige roboter i varierende miljø. Clearpath har satt opp en løsning for simulering i Gazebo[17], der applikasjonen Rviz brukes for å styre roboten. Rviz representerer robotens informasjon om seg selv, og Gazebo viser robotens syn på verden rundt den.

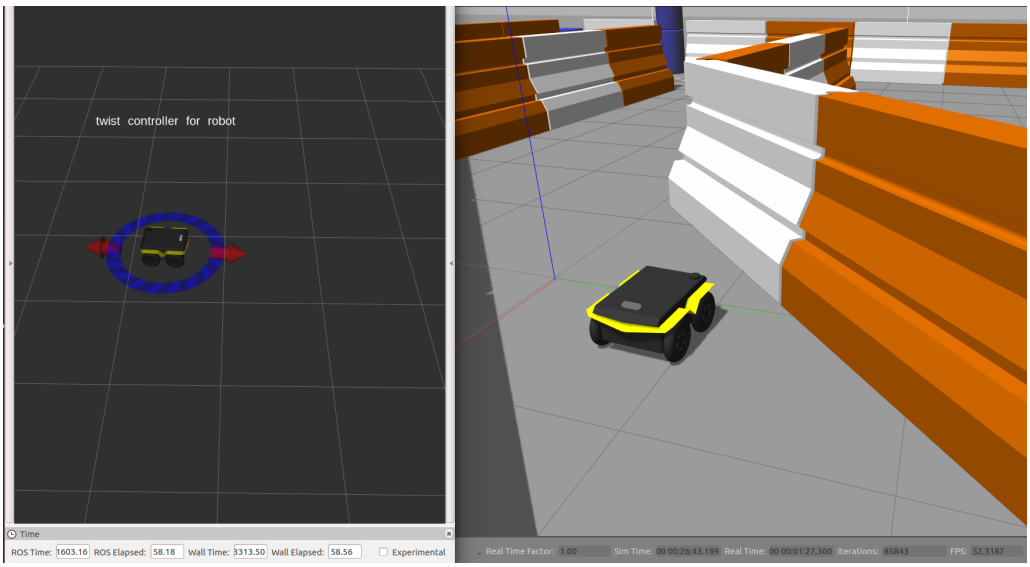
Clearpaths dokumentasjon inkluderer løsning for montering av Intel RealSense D435 på Jackal. Det tillater oss å generere og hente ut bilder og data fra simuleringen. Dataen fra Clearpath Jackal blir på samme måte som i simuleringene samlet i en rosbag. Gjennom simulering fikk vi testet hvilke kommandoer som behøves for å starte en bagfil med ønsket data fra topics.

For å starte simulering i Gazebo med Clearpath Jackal kjører man disse kommandoene i konsollet på Ubuntu:

```
export JACKAL_URDF_EXTRAS = $HOME/../realsense.urdf.xacro
roslaunch jackal_gazebo jackal_world.launch platform:=jackal
roslaunch jackal_viz view_robot.launch
```

Etter simuleringen fikk vi rosbager med data som var nyttig til testing av utpakking og databehandling. I figur 5.2 kan en se et oversiktsbilde av simuleringen. I kapittel 6.1.2 er RGB- og dybdebilder fra simulering vist i figurene: 6.3a og 6.3b.

Bruken av simulering i oppgaven har gjort det enklere å kjøre hyppige tester. Det har bidratt til en bedre forståelse for ROS, og har hjulpet på den bratte læringskurven som kommer med det.



Figur 5.2: Simulering av Jackal. Rviz styring til venstre, Gazebo simulering til høyre.

5.1.5 Global positioning system

GPS dataen som blir lagret i rosbag inneholder flere målingssett med tittel og verdi. Eksempel fra utsnitt av rådata: “latitude: 59.904...”, “secs: 1600174550”. For lagring til timeseries, eller opprettelse av waypoints, benytter vi enkelte av disse settene for å hente ut dataen vi ønsker. Settene vi er interessert i er latitude, longitude, altitude og timestamp. I waypoint program brukes kun latitude, longitude.

Rådataen fra GPS som vi får fra rosbag ser slik ut:

```
topic:
    /navsat/fix

message:
    header:
      seq: 2628
      stamp:
        secs: 1600174550
        nsecs: 987762078
      frame_id: "navsat_link"
    status:
      status: 0
      service: 1
    latitude: 59.9045106667
    longitude: 10.6259649667
    altitude: 89.071
    position_covariance: [1.3224999999999998, 0.0, 0.0,
                          0.0, 1.3224999999999998, 0.0,
                          0.0, 0.0, 5.289999999999999]
    position_covariance_type: 1

timestamp:
    1600174550991753311
```

Verdiene for `latitude` og `longitude` er oppgitt i grader, `altitude` er oppgitt i meter over havet. `position_covariance` er en estimert kovarians i kvadratmeter (m^2). Ifølge [18, s. 7] er en god måling på dette mellom 1-5, noe Jackals målinger på latitude og longitude er. `position_covariance_type: 1` vil si at kovariansen er estimert fra usikkerheten til posisjonen til satellittene, se figur 3.7. /navsat/fix er navnet på rostopic-et dataen kommer fra. 1600174550991753311 er tiden målingen ble gjort, dette tilsvarer secs og nsecs. Dette blir oppgitt i Epoch time, det er basert på antall sekunder som har forløpt siden midnatt (GMT) 1 januar 1970 [19]. Timestamp konvertert til dato og tid i GMT format er: Tirsdag 15. September 2020 12:55:50.991

På neste side vises et utsnitt av waypoint uthenting. Informasjonen hentes ut gjennom en for-løkke i Python. Rådataen er lagret i variabelen "bagContents". Her lagrer vi all data fra topic-et /navsat/fix til tabellen "topicTable". Deretter leser vi ut verdiene som tilhører latitude og longitude. Disse lagres til nye tabeller, som brukes for waypoints.

```
# separates message into individual components in a list
for subtopic, message, time in bagContents:
    if subtopic == "/navsat/fix":
        messageString = str(message)
        # split message for each newline
        messageList = string.split(messageString, '\n')

        for nameValuePair in messageList:
            # split header and message value.{"example":"1.93"} to ["example","1.93"]
            splitPair = string.split(nameValuePair, ':')

            for i in range(len(splitPair)):
                splitPair[i] = string.strip(splitPair[i])
            topicTable.append(splitPair)

for x in topicTable:
    if x[0] == "latitude":
        latitude.append(x[1])
    if x[0] == "longitude":
        longitude.append(x[1])
```

5.1.5.1 Waypoint fra rosbag

Clearpath Jackal er satt opp til å kunne følge waypoints som den får via en `json` fil. Vi har laget en python-basert rosservice som leser fra en `bag` fil som inneholder GPS data, og skriver de til en slik `json` fil. Programmet henter GPS data fra rostopic-et /navsat/fix, som gir oss latitude, longitude og altitude verdier. GPS dataen kan komme fra en tidligere testtur med Jackal eller lignende. Navigasjon med waypoints gjør at en enkelt kan gjenskape ruten som ble kjørt. Dette gjør det også mye enklere å få en repeterbar rute som kan brukes til testing og reell kjøring.

Dersom vi hadde brukt alle målingene fra GPS informasjonen til waypoints ville det blitt ekstremt mange punkt den skulle kjørt innom. Se figur 6.5a på hvordan dette vil se ut i et kart. Usikkerheten til hver måling i sammenheng med svært mange waypoints kunne gjort at det ble frem og tilbake("hakkete") kjøring. For å kunne få mer flyt i kjøringen la vi inn filtrering med hensyn på distanse. Her kan en velge hvor mange meter en vil ha mellom hvert waypoint. Se figur 6.5 for resultat.

For å visualisere waypoints har vi brukt nettsiden GPSvisualizer [20]. I rosservice som lager waypoints har vi lagt inn skriving til tekstfil(`txt`). Denne tekstfilen er formatert slik

at den kan lastes rett opp til GPSvisualizer som plotter waypoints på kart.

Koden for rosservice vist i appendiks: A7 for Server og A8 for Client .

5.1.6 Inertial measurement unit data

Fra IMU sensoren på Clearpath Jackal får vi orientering, vinkelhastighet og lineær akselerasjon. Denne dataen skal brukes sammen med GPS dataen for å fastslå posisjonen til Clearpath Jackal.

Rådata som blir lagret i rosbagen fra IMU sensoren er vist under:

```
topic:
    /imu/data

message:
    header:
      seq: 87358
      stamp:
        secs: 1600174551
        nsecs:    762078
      frame_id: "imu_link"
    orientation:
      x: 0.0142741284381
      y: -0.00799684200653
      z: -0.999320852418
      w: -0.0330171727798
    orientation_covariance: [0.0, 0.0, 0.0,
                             0.0, 0.0, 0.0,
                             0.0, 0.0, 0.0]
    angular_velocity:
      x: -0.0148480484069
      y: 0.032048183548
      z: 0.00404170245627
    angular_velocity_covariance: [1.218467815533586e-07, 0.0, 0.0,
                                  0.0, 1.218467815533586e-07, 0.0,
                                  0.0, 0.0, 1.218467815533586e-07]
    linear_acceleration:
      x: -0.55620148097
      y: -0.296066525521
      z: 9.55834881147
    linear_acceleration_covariance: [8.661248102725949e-06, 0.0, 0.0,
                                     0.0, 8.661248102725949e-06, 0.0,
                                     0.0, 0.0, 8.661248102725949e-06]

timestamp:
    1600174551003585993
```

De første resultatene en kan se her er tiden målingen ble utført. secs: 1600174551 nsecs:762078 representerer med vår tidsform: GMT: Tuesday 15. September 2020 12:55:51.762. Orientation(x,y,z,w) er målingen fra magnetometeret. Angular velocity(x,y,z) er målingene fra gyroskop og linear acceleration(x,y,z) er målingene fra akselerometer. Hver av sensorene har en måling kovarians, som er et mål på usikkerheten til målingene.

5.1.7 Intel RealSense

Intel RealSense D435 (se figur 5.3 er et stereokamera som tar både RGB- og dybdebilder. Fra dybdebildene kan en anslå avstanden fra kamera til objektene på bilde. Kombinasjonen av RGB- og dybdebilde muliggjør en 3D-representasjon av motivet en tar bilde av.

Figur 5.3: Intel® RealSense™ Depth Camera D435[21]

Bildedata fra RealSense blir sendt på samme måte som IMU og GPS. Da er ett av målingssettene en matrise med RGB verdier fra 0-255. Gjennom bruk av CvBridge og OpenCV kan denne informasjonen skrives til et 8-bit bilde. Dybdebilder virker på samme måte, men der skrives dataen til et 8-bit numpy array. Figur 6.3b viser dybdeinformasjon representert i et bilde. Vi har valt å lagre dybdeinformasjon som `npy` i motsetning til `jpg`. Det er fordi man mister en del informasjon når det konverteres til `jpg`. I tillegg er det enklere å jobbe videre med en `npy` fil, da det er et standard filformat i NumPy og OpenCV.

5.2 Databehandling

5.2.1 Behandling av `bag` filer

Bag filen som kommer fra Clearpath Jackal må behandles før informasjonen kan lastes opp til CDF. For dette har vi laget et program som pakker ut bagfilen til `csv`, `jpeg` og `npy` filer. En `bag` fil må behandles i et miljø som støtter ROS, det fungerer dårlig i Windows. Dette har vi løst ved å bruke en maskin med Ubuntu 16.04 og ROS Kinetic. Se appendiks A3 for installasjonsveiledning.

For å kunne kjøre utpakkingen i terminalen i samarbeid med ROS, bruker vi en rosservice. Da kan programmet kjøre i bakgrunnen, og man kan kalle det etter behov. Det er en praktisk løsning for Clearpath Jackal, da man kan bruke den hver gang en inspeksjonsrunde er utført.
Koden vi bruker i terminal for å kjøre rosservice er vist under:

```
# Start extract rosservice
rosrun jackal_edge bag_extract_server.py
# Call service with path to bag
rosservice call /bag_extract "/home/user/../myBag.bag"
```

Dette er for kjøring av rosservice for utpakking av `bag` fil. Videre i kapittelet kan en lese om hvordan selve utpakkingen av bagfilen gjøres for tekst- og bilde-data.

5.2.1.1 Sensordata som tekst

Som vist i kapittel 5.1.5 lagrer ROS data som en tabell med målingssett. Hver sensor har sitt egen topic og hver måling i sensoren har sin egen header(tittel). Når vi skal hente ut data fra `bag` filen bruker vi topic navnet til sensoren og header til målingene for å skrive dette til en fil. Filen vi lagrer dette til er en `csv` fil.

En `csv` (comma-separated values) fil er et filformat som er mye brukt siden det støttes av mange programmer. Det er et enkelt format der en skriver verdiene fra sensor som en lang tekst-streng og bruker komma (,) for å skille mellom de ulike målingene. Linjene i tekstfilen representere ulike rader. Filen blir laget for å lagre tabellarisk data i tekst form. Se delkapittel 6.1.1 for eksempel på `csv` data.

Det blir laget en `csv` fil for hvert topic. Filen inneholder alle målingene som sendes over

det topicet, som regel all data fra en sensor. For eksempel fra IMU sensoren blir navnet på filen: `-imu-data.csv`, den inneholder 16 forskjellige titteler. Av de 16 tittelene er vi interessert i 10 av dem: timestamp, Orientation (x,y,z), angular-velocity (x,y,z) og linear acceleration (x,y,z). For lagring til for eksempel timeseries bruker vi tittel for å hente ut dataen vi ønsker. Mer om timeseries kommer i delkapittel: 5.2.2.3.

For å forhindre at uønskede emner(topics) blir skrevet til `csv` fil har vi laget en konfigurasjonsfil i `yaml` format, se appendiks A6. I den kan en definere hvilke emner som en ikke er interesserte i å skrive til `csv` fil. Emnene som skal lagres som `jpg` og `npy` er også definert i konfigurasjonsfilen som en liste, dette er RGB bilder og dybdebildene for hvert kamera.

5.2.1.2 Bilder

Uthenting av bilder skjer ved hjelp av CvBridge. Som vi gjør på skriving av `csv` fil leser vi et topic fra rosbag, og bruker en for-løkke for å dele den i topic, message, og timestamp. Herfra kan vi gi CvBridge hele "message" variabelen, og funksjonen vil konvertere det til et format som kan brukes av OpenCV og NumPy. Vi bruker to lister for å kontrollere om topic-et er RGB- eller dybdebilde. Filnavnet generes ut ifra rosbag, topic og timestamp (se kode under). Det sørger for at vi aldri lager filer med det samme navnet, som kan skape problemer.

```
head, tail = os.path.split(folder) # tail i just .bag name
top = rostopic.name
if listOfImages.__contains__(topic):
    file = string.replace(tail, '/', '-') + '_' + top + '_' +
    str(msg.header.stamp) + '.jpeg'
else:
    file = string.replace(tail, '/', '-') + '_' + top + '_' +
    str(msg.header.stamp) + '.npy'
```

Det blir også laget en `txt` fil for timestamp sammen med bilde navnet. Dette for å holde oversikt over tiden bilde ble tatt slik at en kan spore det tilbake til samme tidspunkt på GPS og IMU dataen. Dette gjør at en kan fastslå posisjonen til Clearpath Jackal i tidspunktet der bilde ble tatt. Posisjonen til Jackal, kombinert med informasjonen fra dybdekameraet gjør at en kan fastslå posisjonen til objektet en observerer.

Bilde typene vi henter ut er RGB og dybde bilder. RGB bildene blir lagret som `jpeg`

og dybde bildene blir lagret som `npy`. Hvordan de blir hentet ut er forklart i starten av delkapittelet. Selve lagringen av bildene er vist i kodeutsnittene under. OpenCV brukes for å lagre RGB bildene, NumPy brukes for å lagre dybdefiler. Variabelen "msg" i koden kommer fra for-løkken som går gjennom hver melding i topic.

Lagring av RGB bilder i `jpeg` format:

```
cv_img = bridge.imgmsg_to_cv2(msg,msg.encoding)
cv.imwrite(os.path.join(imFolder, file), cv_img)
```

Lagringen av dybdebildene i `npy` format:

```
cv_image = bridge.imgmsg_to_cv2(msg,msg.encoding)
numpy_image= np.array(cv_image,dtype=np.uint8)
np.save(os.path.join(imFolder, file), numpy_image)
```

5.2.2 Opplasting til CDF

All data som blir samlet fra Clearpath Jackal skal lastes opp til CDF. Det er et av hovedformålene med oppgaven. Dette gjøres gjennom Cognite API[22] og Cognite Python SDK [23].

I starten av prosjektet brukte vi Postman [24] for å teste API-requests, og få en bedre forståelse for oppbyggingen og strukturen. Etter å ha fått ett innblikk i Postmann og kommandoene som blir brukt der startet vi å jobbe med Cognite Python SDK. Her utformet vi kode som stegvis samlet løsninger for å lage Assets, Timeseries og opplasting av filer osv. Dette gav oss en fin innføring i hvordan de forskjellige elementene fungerer og hva som er nødvending for å opprette/laste opp Assets/Timeseries/filer.

Prosjektet har sitt eget område i CDF, som har vi tilgang til gjennom et web-grensesnitt, og private API-keys. Det gir oss muligheten til å skape Assets, laste opp filer, kontrollere data, og mye mer (for mer informasjon, se Cognites dokumentasjon [23]). Hver fil, Asset og Timeserie får hver sin unike ID laget av CDF.

For å kunne opprette/laste opp Assets, Timeseries og filer trengs det gyldig pålogging til prosjektet en vil koble seg til. Påloggingen vil se slik ut i Python:

```
from cognite.client import CogniteClient
c = CogniteClient(api_key=os.getenv("COGNITE_API_KEY"),
project="example-project", client_name="<your_client_name>")
```

For å sørge for at API-keyen er personlig og ikke kom på avveie legges den inn som en systemvariabel. `os.getenv("COGNITE_API_KEY")` henter API-Key fra datamaskinens systemvariabler.

5.2.2.1 Assets

I CDF kan en lage ressurstyper digitalt som representerer den virkelige verden. Disse ressursene kalles Assets. Eksempel på dette er ventiler, pumper osv. Tilhørende utstyr til disse ressursene kan linkes sammen for å lettere identifisere dataen som er relevante for ressursen. En kan også lage under-Assets som tilhører den overordnede Asseten slik at en enklere kan holde oversikt over systemene.

Hver enkel Assets har sin egen unike ID, men vi har også mulighet for å tilegne en ekstern id, og linke den opp mot andre Assets.

5.2.2.2 Filer

I CDF kan en lagre filer som er relaterte til en eller flere Assets. For å laste opp filer må en definere hvilken fil en vil laste opp og hvilken asset den skal bli koblet til. Et eksempel på opplasting av et bilde til en Asset er vist under:

```
c.files.upload(C:/Users/filplassering, asset_ids=[123456789],mime_type=image/jpeg)
```

En kan også definere hvilken filtype filen er med `mimie_type=` som vist over, dette gjør det mulig å forhåndsvise enkelte filen i CDF. Det er også mulig og laste opp flere filer samtidig, da oppgir en bare mappen filene ligger i.

De filene vi i hovedsak har lastet opp til CDF er bilde(`jpeg`, `npy`) og `csv` filer. Bildene blir opplastet til sin respektive Asset(Realsense Front eller Realsense Right).

5.2.2.3 Timeseries

I CDF er det laget en egen ressurstype som heter Timeseries, dette er en datatype som tar inn datamålinger og visualiserer de med hensyn på tiden målingen ble utført. Timeseries er laget for å enklere kunne analysere og gjøre avgjørelser basert på dataen.

For å laste opp data til timeseries på CDF brukes det et bibliotek som heter Pandas. Her brukes funksjonen dataframe som er en todimensjonal tabell med merkede kolonner og rader. Programmet vi har laget henter data fra `csv` fil for så og skrive det til en Pandas.dataframe. Første kolonner er index, det er timestamp data som konverteres fra epoch til standardtid. Andre kolonne er sensor data, med en tittel. Tittelen er CDF-ID nummeret til Timeserien, sensor dataen er message data som tilhører indexen.

```
t = tab["rosbagTimestamp"]
tArray = []
for value, idC in l.items():
    for Y in t:  # epoch2date (YYYY-mm-dd HH:MM:SS.ffffff)
        tArray.append(datetime.utcfromtimestamp(
            Y/1000000000).strftime('%Y-%m-%d %H:%M:%S.%f'))

    data = tab[value]
    df = pd.DataFrame({idC: data.values}, index=tArray)

    c.datapoints.insert_dataframe(df)
```

Vi har laget totalt tolv Timeseries i CDF, ni av de er IMU data og de tre resterende er GPS data. De tre for GPS data er Latitude, Lontitude og Altitude. IMU timeseriene inneholder x, y og z for vinkel hastighet, lineær akselerasjon og orientering. For eksempel på timeseries se kapitel 6.7 for IMU og kapitel 6.6 for GPS.

5.2.3 Kamerakalibrering

Ved starten av hver datainnsamling skal det filmes et rutemønster i forskjellige positurer. Fra denne delen av videoen kan en trekke ut bildene av rutemønsteret for kalibrering. Koden for kalibrering er avhengig av bilder som input, hvert bilde må være av et rutemønster som den detekterer hjørnene til.

Fra kalibreringen vil en få ut kameramatrise og forvrengningskoeffisienter, de gir et mål på hvor nøyaktig/unøyaktig kamera er. En får også ut rotasjonsmatrise og transformasjonsmatrise på rutemønsteret avhengig av bildene. Til slutt blir det regnet ut

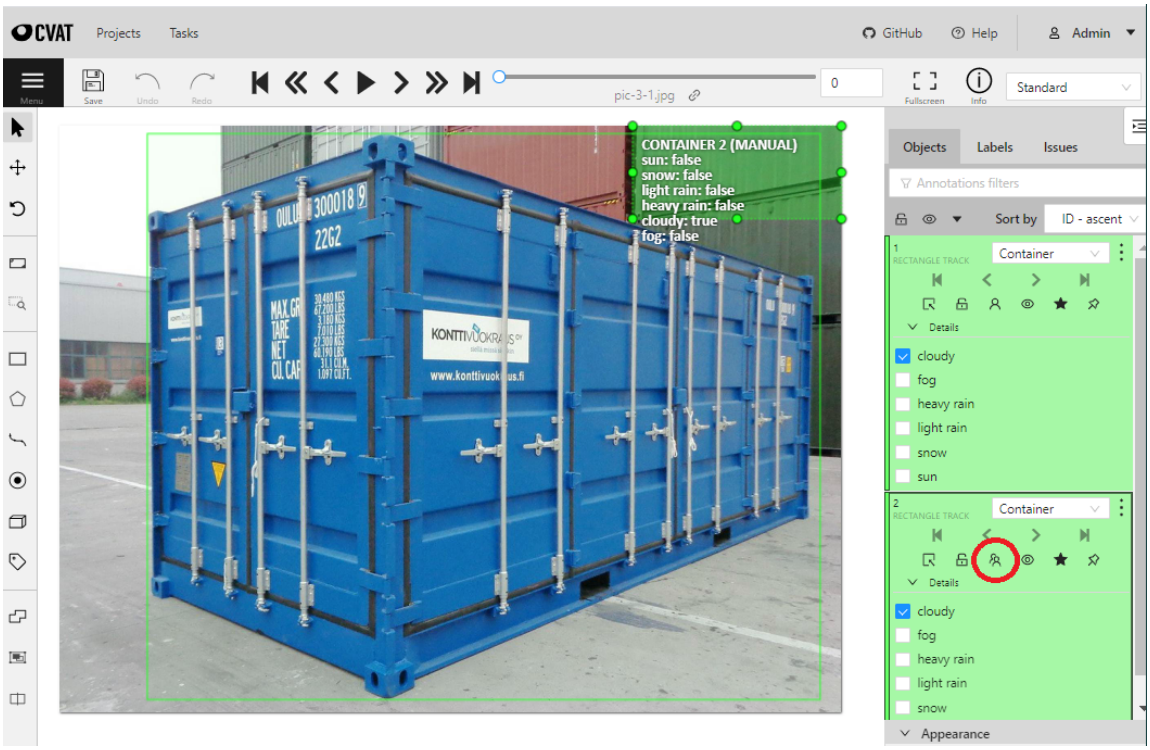
en feilmargin på hvor gode parameterne er. Hvordan disse parameterne blir funnet kan en lese om i kapittel 3.4.

Tidlig i prosjektet testet vi kalibreringskoden med web-kamera. Resultat fra disse testene er vist i kapittel 6.4. Planen var å bruke koden sammen med Intel RealSense kameraet montert på Clearpath Jackal. Ettersom vi ikke fikk tilgang til Jackal har vi heller ikke testet koden med Intel RealSense. HVL hadde heller ikke noen tilsvarende kamera, så det ble ikke testet mer.

Python kode som er utviklet for kalibreringen kan en lese i appendiks A13. Kode er utviklet ved hjelp av OpenCV sin brukerveiledning [25].

5.2.4 Data-annotering

En del av oppgaven er å generere et datasett for masterstudent Kevin. Han har i oppgave å lage en algoritme for å gjenkjenne containere. For at algoritmen skal være robust trengs det et bredt datasett. Dette datasettet skal inneholde bilde av forskjellige containere i varierende værforhold, vinkler og posisjoner. Bildene skal så annoteres, de vil si: markere hvor containeren er plassert i bilde.



Figur 5.4: Her vises registrering av to containere i CVAT. Hver container blir registrert med værforhold, og synlighet(rød sirkel).

Annotasjons-programmet vi bruker for å markere bilde heter CVAT, og kjører gjennom med Docker desktop. Her marker vi posisjonen til container i bildet. Vi registrer også værforhold og tildekking. Dette gjør en etter en har markert container, hvordan CVAT ser ut er vist i figur 5.4.

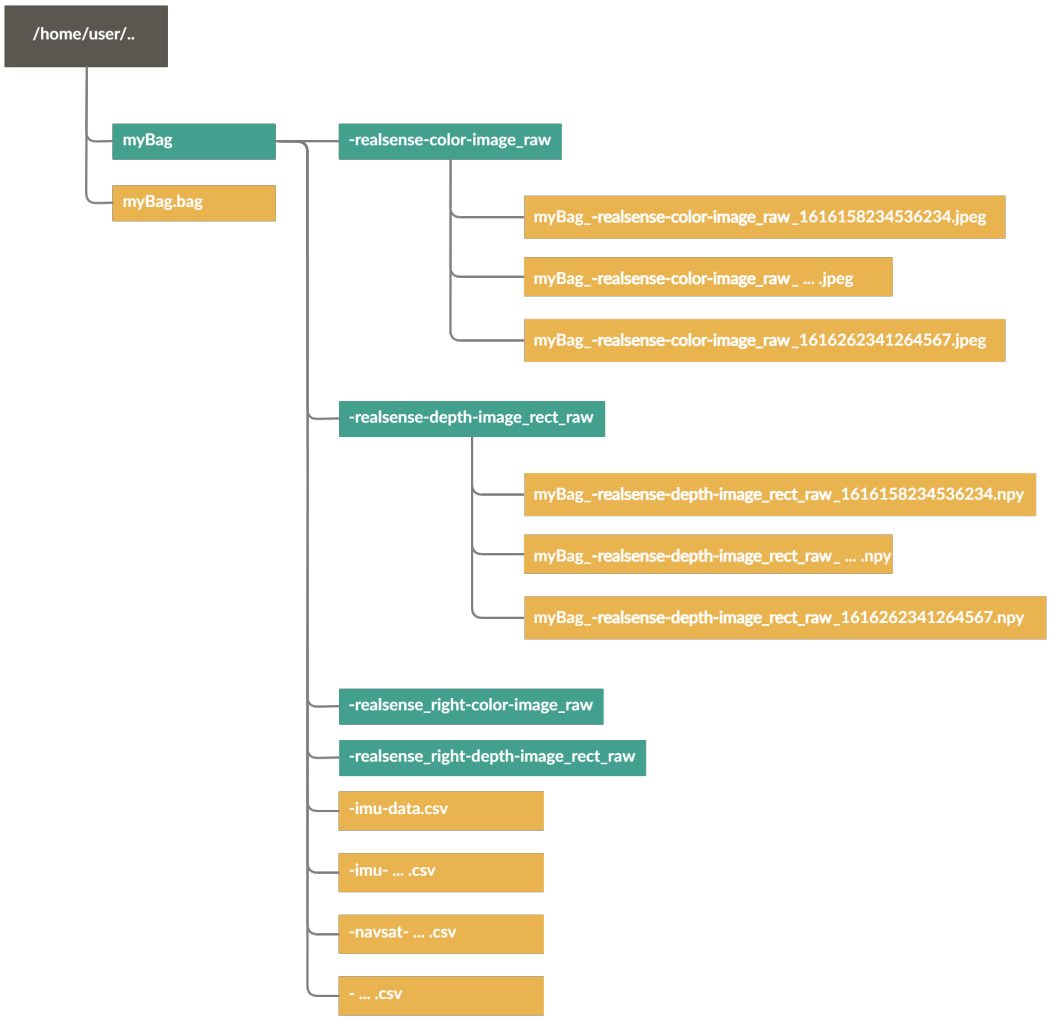
Data annotasjonen kan formateres etter forskjellige standarder, den vi bruker heter PASCAL VOC, etter ønske fra Kevin. Alt lagres som en `zip` fil, med bildene samt informasjonen som bilde er markert med. Informasjons filen som blir generert fra data annoteringen til hvert bilde vil være av filtypen `xml`.

Gjennom prosjektarbeidet ble det utformet en data annotasjons "pakke". Det er bilder som er tatt på fritiden av containere som vi har kommet over. Det er totalt åtte bilder av containere som er ferdig annoterte og lastet opp. For å kunne se informasjonen som blir generert ved annotasjonen se siste side av appendiks A14.

6 Resultat

6.1 Utpakkingeen av en `bag` fil

Organiseringen av utpakkingen er vist i figur 6.1. Mappen- og fil-navn er dynamisk generert fra innholdet i `myBag.bag`. Koden for utpakking er vist i appendiks A4 for Server og A5 for Client.



Figur 6.1: Fil-tre som genereres av BagExtractService. De grønne seksjonen representerer en mappe, de oransje representerer en fil.

6.1.1 Resultat av Tekst/sensor data

Her ser dere hvordan tekstformatet blir i en csv fil, her er et kort utdrag fra GPS data.

```
rosbagTimestamp,header,seq,stamp,secs,nsecs,frame_id,status,status,service,latitude,longitude,altitude
1600174526990542139,,2388,,1600174526,992091562,"""navsat_link""",,0,1,59.9044389,10.626104,85.395
1600174527092415209,,2389,,1600174527,93091562,"""navsat_link""",,0,1,59.9044411667,10.62610045,85.493
```

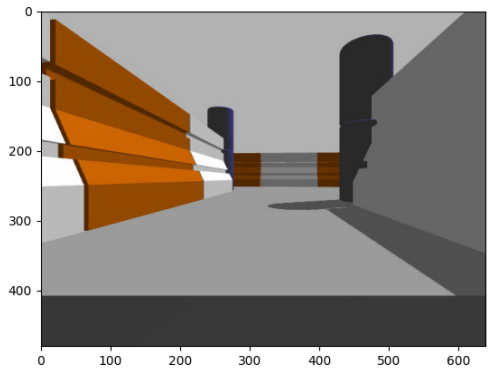
Åpner man en csv fil i Microsoft Excel får man en penere representasjon av dataen. Den splitter informasjonen etter hvert komma (,) og setter det i individuelle celler. Hver rad er en måling fra sensoren. Dette er vist i figur 6.2.

rosbagTimestamp	header	seq	stamp	secs	nsecs	frame_id	status	status	service	latitude	longitude	altitude	position_covariance
1600174526990540000		2388		1600174526	992091562	"navsat_link"		0	1	59.9044389	10.626104	85.395	[1.2768999999999997, 0.
1600174527092410000		2389		1600174527	93091562	"navsat_link"		0	1	59.90444117	10.62610045	85.493	[1.2768999999999997, 0.
1600174527176740000		2390		1600174527	176091562	"navsat_link"		0	1	59.90444328	10.62609723	85.581	[1.2768999999999997, 0.
1600174527278840000		2391		1600174527	280091562	"navsat_link"		0	1	59.90444542	10.62609403	85.664	[1.2768999999999997, 0.
1600174527392580000		2392		1600174527	384091562	"navsat_link"		0	1	59.90444845	10.62608913	85.766	[1.2768999999999997, 0.
1600174527482810000		2393		1600174527	480091562	"navsat_link"		0	1	59.90444845	10.62608828	85.756	[1.2768999999999997, 0.

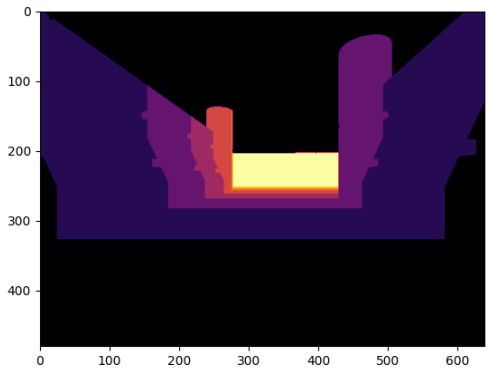
Figur 6.2: csv data representert i Microsoft Excel. Eksempel fra rostopic /navsat/fix.

6.1.2 RGB- og dybdebilde av simulering

Resultat av bag utpakking gir mapper som inneholder bilder for hvert kamera (se figur 6.1). RGB bilde fra en Gazebo-simulering er vist i figur 6.3a. I figur 6.3b er det samme motivet vist som dybdebilde.



(a) RGB bilde av simulering



(b) Dybdebilde av simulering

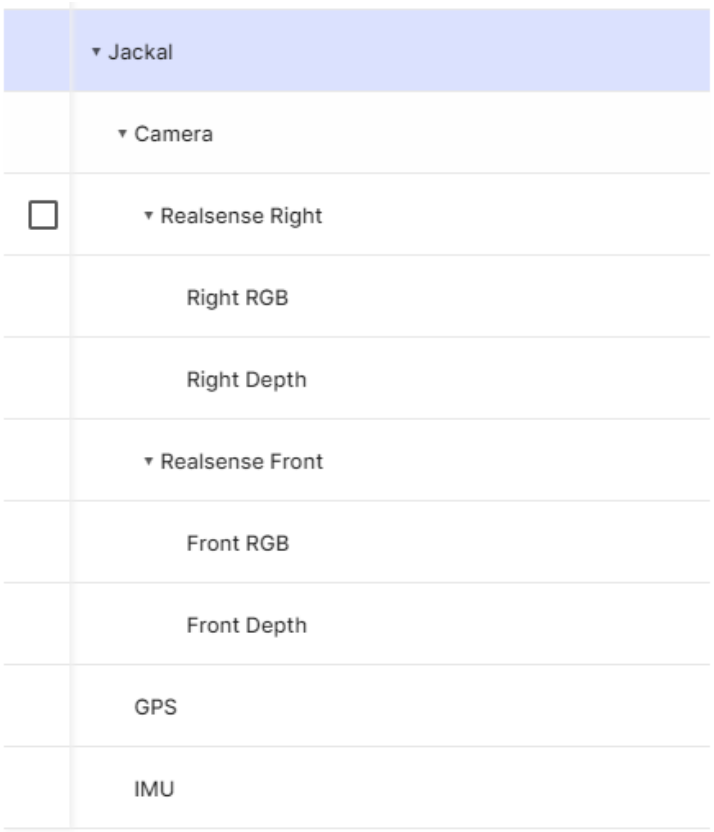
Figur 6.3: RGB- og dybde-bilde av simulering

6.1.3 CDF Assets og organisering

Her har vi presentert kommandoene som må til for å lage en Asset. Under er et utsnitt fra opprettingen av Asseten "Camera" og linken mot Jackal, som er vår hovedasset.

```
my_asset = Asset(name="Camera",parent_id=1234567890)
c.assets.create(my_asset)
```

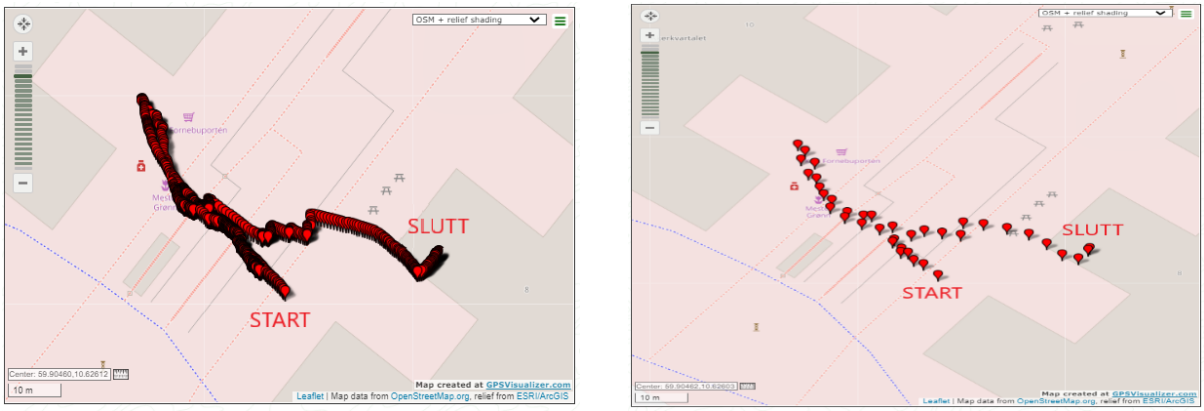
Hvordan vi har valgt å organisere CDF er presentert videre. Vi har en hovedasset som er "Jackal", den har tre under-assets som er Camera, GPS og IMU. "Camera" har også to under-assets som tilhører hvert sitt kamera på Jackal, disse heter "Realsense Front" og "Realsense Right". Disse har separate Assets for bildetypene vi laster opp, de er navnsatt Front/Right etterfulgt av RGB/Depth. Asset hierarkiet i CDF er vist i figur: 6.4.



Figur 6.4: Et utsnitt fra CDF på hvordan vi har valt og organisere Assets

6.2 Visualisering av waypoints og GPS-data

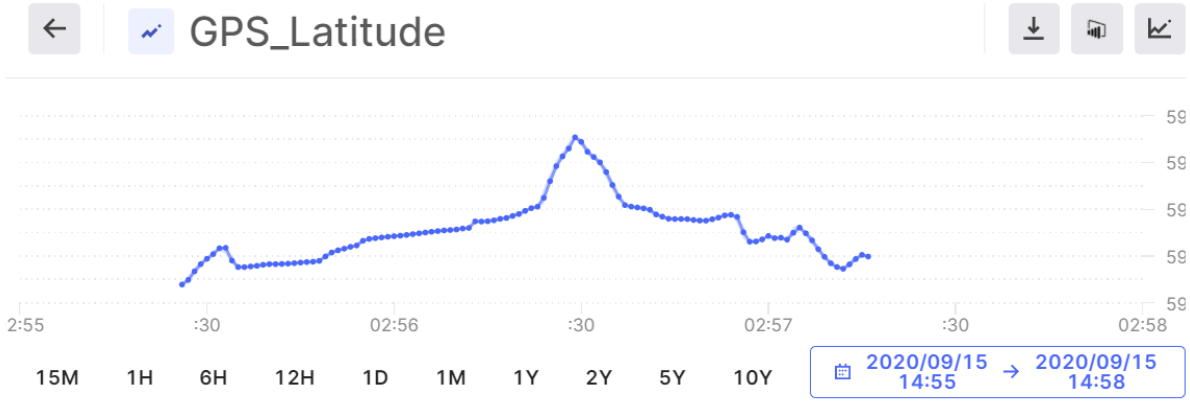
I figur 6.5a er alle GPS punktene skrevet til waypoints. I figur 6.5b er det introdusert et filter på fire meter mellom hvert waypoint. Datasettet er hentet fra testekjøring ved Cognites kontor i Oslo.



(a) Alle waypoints **(b)** Filtrert waypoints med 4 meter mellom

Figur 6.5: GPS waypoints visualisert ved bruk av GPSvisualizer [20]. Målingene er fra testkjøring ved Cognites kontor i Oslo.

Timeseries fra GPS latitude er vist i figur 6.6, den er fra kjøring av Clearpath Jackal september 2020. I CDF er Longitude og Altitude også representert som Timeseries.

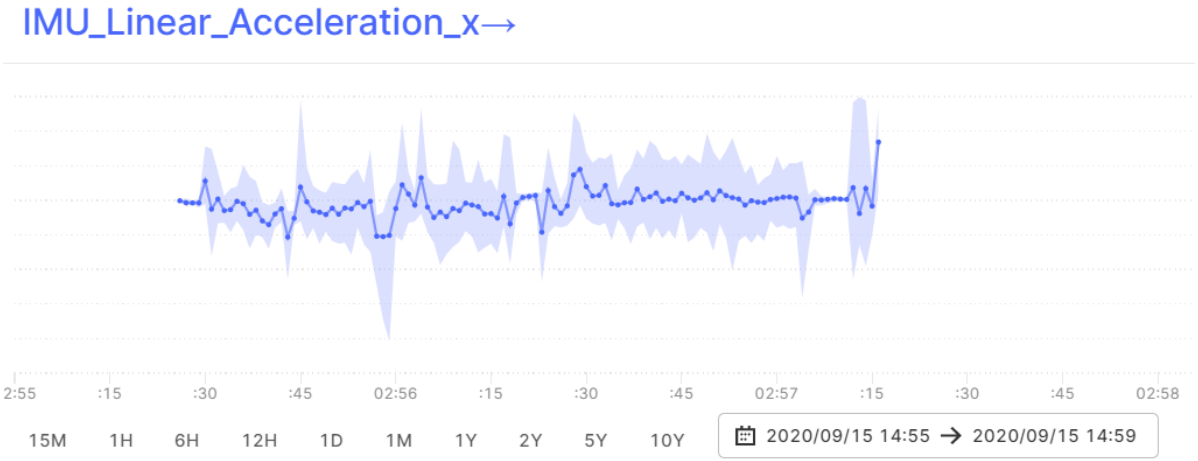


Figur 6.6: Timeseries av latitude målinger fra kjøring av Jackal september 2020

6.3 Timeserie plot av IMU-data

I figur 6.7 er målingene fra IMU sensor vist i Timeseries. Disse målingene er fra samme kjøring av Clearpath Jackal september 2020, som tidligere nevnt. Grunnet en bugg i CDF ser man ikke y-aksen på grafen. I CDF er all data for x,y,z representert for linear acceleration, angular velocity og orientation.

(a) Angular Velocity data fra IMU.



(b) Linear Acceleration data fra IMU.

Figur 6.7: IMU-data representert i CDF som Timeseries. Her fra en kjøring med Jackal fra september 2020.

6.4 Kamerakalibrering

Den testingen vi fikk med kamerakalibrering var med vårt web-kamera. Et av bildene dette resultatet er basert på er figur 3.11b. Resultatet av kalibreringen er vist under:

```
Kameramatrise:
    [[525.03637334   0.         294.14078651]
     [  0.         526.94051942 269.16462742]
     [  0.           0.           1.        ]]

Distortionkoeffisient (k1 k2 p1 p2 k3) :
    [[ 0.01662835  0.00184059  0.00668501 -0.0119708  -0.21287339]]

Rotasjonsmatisen:
    [array([[-0.09853857],[-0.36795853],[-1.69869585]]),
    ...
    array([[-0.28994177],[ 0.20078938],[-1.47894395]])]

Transformasjonsmatrisen:
    [array([[-6.33840154],[ 4.70666859],[15.87749769]]),
    ...
    array([[-0.8266105 ],[ 1.13154056],[28.15177495]])]

Total error: 0.120150804227871
```

Det første en ser her er kameramatrisen og distorsjonskoeffisientene. Videre ser vi en rotasjonsmatrisen og transformasjonsmatrisene for hvert bilde. Rotasjonsmatrisen og transformasjonsmatrisen beskriver hvordan rutemønsteret er posisjonert og orientert i forhold til kamera. Her valgte vi og bare å vise den første og siste matrise.

Total error er et mål på hvor god parameteren er. Den bruker kameramatrisen, distorsjonskoeffisientene, transformasjonsmatrisene og rotasjonsmatrisene til å beregne den totale feilmarginen. Tallet gir deg et gjennomsnittsmål på pikselfeilen i bildene. Mindre enn 0.2 er et bra mål på denne. Vår er 0.12 som vil si at kalibreringen er bra.

6.5 Data-annotering

Som en ser i figur 6.8 skal containeren merkes med "sann" posisjon, de skal også merkes med værforhold, og om hele containeren er synlig eller ikke. For en mer utfyllende fremgangsmåte av data annotasjon se appendiks A14. Der vises også et eksempel på PASCAL-VOC format.

Figur 6.8: CVAT layout for datanotasjon

7 Diskusjon

Gjennom oppgaven har vi vært gjennom mange løsninger og sett på behovet for forskjellige systemer. Vi har støtt på en del utfordringer på veien og noen av disse er presentert videre. Det har også blitt noen endringer i prosjektet og dette har resultert i en utvidelse av oppgaven.

Analyse av resultat

Løsingen vår er python kode strukturert med en rekke funksjoner. Vi har prøvd å holde oss til en standard navn-konvensjon, og har navngitt variabler med tanke på lesbarhet og forståelse. Noe av koden er litt tungvint utført, og vi har noen "brute-force" metoder for å løse enkelte problemer. For eksempel bruken av konkrete ID-nummer og rostopic-navn i koden. Dette kunne blir løst på en mer allsidig måte med bruk av API-GET requests for ID-nummer, og mer detaljert analyse av rosbag for å hente ut datatype og topic navn. Vi har vært flittige med bruk av try-exept funksjoner, dette for å fange opp feil input fra bruker.

Det er ingen fasit på hvordan denne oppgaven kan løses. Gjennom prosjektet har vi testet flere metoder, og utviklet en løsning vi mener er praktisk og effektiv.

Vi løser problemet med å bruke funksjoner og løsninger som tilhører ROS-miljøet. Det gjør det enklere å integrere løsningen på Clearpath Jackal, eller eventuelle andre roboter som bruker ROS Kinetic. Løsningen er utarbeidet i samarbeid med Cognite, så valgene samstemmer med deres designfilosofi.

Utfordringer underveis

I starten av prosjektet hadde vi lite erfaring med Python, det har vært utfordrende og lærerikt å jobbe med. Det har gått med en del tid til å forstå og lære seg syntaksen og metodene som brukes.

Med CDF-API hadde vi noen utfordringer med opplastingen til CDF. Dette var i starten av prosjektet og var først og fremst knyttet til usikkerheten rundt hvordan CDF virket. Det ble mye lesing på dokumentasjon og testing for å forstå hvordan opplastingen utføres. Vi fikk til slutt en forståelse av hvordan Assets, Timeseries og filopplasting fungerte og

hvordan det henger sammen.

For å gjøre programmene mer brukervennlig har vi laget de som en rosservice. Under opprettelsen av disse servicene var det en del problemer knyttet til kompilering og kjøring. Dette var hovedsakelig grunnet manglende erfaring og kunnskap om ROS.

Kommunikasjonen med Clearpath Jackal går gjennom SSH tilkobling og er sikret gjennom VPN. Her har vi hatt problemer med ustabil tilkobling. Det har forhindret testing av programmer på Jackal. En versjon av Ubuntu og ROS på vår PC har hjulpet på dette, da vi kunne teste lokalt.

Vi har sett på utfordringene med Ubuntu, og hvordan versjoner og støtte kan skape problemer i et sammensatt system. Vi startet på en løsning med Docker container, men kom ikke helt i mål med det.

Uforutsette endringer

Vi har sett på kamerakalibrering, og nødvendigheter av det i et moderne computer vision produkt. Det ble ikke en så stor del av oppgaven som vi forventet, delvis på grunn av manglende tilgang til kamera. Etter analyse av dokumentasjon [26] ser det ut til at Intel RealSense er særdeles nøyaktig, og at det mest sannsynlig ikke er det behovet som vi hadde sett for oss. Med dette tatt i betraktning har vi bestemt oss for å ikke implementere kalibrering i løsningen, da vi ikke ser det som nødvendig.

Et annet tema som ikke ble en så stor del av oppgaven som vi hadde forventet er AprilTag. Dette er i første omgang på grunn av fravær av testkjøringene med Jackal. Her hadde vi forventet å hente ut bilder av AprilTags, og sende de til en deteksjonsalgoritme.

Utvidelse

Uthenting av waypoints fra rosbag var opprinnelig ikke en del av oppgaven, men ble gjort etter ønske fra bedrift. Vi ble bedt om å lage en rosservice som skriver filtrete waypoints til en `json` fil. Denne oppgaven passet bra som en utvidelse av prosjektet, ettersom det ikke ble noe praktisk kjøring.

COVID-19

Gjennom semesteret satte COVID-19 en stopper på de fysiske testene for vårt prosjekt. Dette har vi prøvd etter beste evne og jobbe rundt.

8 Konklusjon

Fra problemstillingen ville vi svare på spørsmålet "*Hvordan skal informasjonen fra roboten lagres og formateres, slik at den kan lastes opp til Cognite Data Fusion for videre analyse?*" Gjennom oppgaven har vi utviklet en generell metode for utpakking av data på Clearpath Jackal. Vi har også foreslått en løsning for opplasting av denne dataen til CDF. Dermed mener vi at løsningen svarer på problemstillingen på en tilfredsstillende måte, og vi sitter igjen med et produkt som kan brukes av Cognite.

Python har vist seg å være et praktisk språk å jobbe med. Det er enkelt å sette seg inn i, og støtter de aller fleste funksjoner som brukes i robotikk. Integreringen i ROS og CDF har gjort det enkelt å implementere en komplett løsning, uten behovet for mellomledd og andre tjenester.

Fra et datasynspunkt er løsningen med rosbag og opplasting til CDF et lovende konsept. Om Clearpath Jackal egner seg som en autonom sporingsløsing er for tidlig å si. Vi kan konkludere med at databehandling og dataopplastingen gjennom CDF ser lovende ut. Omfattende testing og analyse av data gjenstår i prosjektet, men det er på god vei, og grunnarbeidet er gjennomført.

8.1 Fremtidig arbeid

Testing på Clearpath Jackal

Programmene er testet i et tilsvarende ROS-miljø som det er på Jackal. De burde derimot også testes på Jackal, for å sørge for kompatibilitet og stabilitet.

For å få testet opplasting kreves det bruk av Docker container. Det er for å løse problematikken knyttet til begrensninger av Python versjonene.

Ricoh Theta V 360 kamera

Ricoh Theta V 360 kamera var originalt en del av kravspesifikasjonen fra Cognite. Det ble derimot tidlig i prosjektet informert om at vi kunne se bort fra det, da Intel RealSense ble prioritert. Skulle man ønske å implementere 360 kameraet kan det være en ide å lage en rosservice for det.

Automatisk opplasting

Foreløpig er opplasting til CDF en semi-manuell prosess. Dette skulle gjerne vært helautonomt. Her kan en god løsning være å konvertere all kode til Noetic. Da vil det være enklere å lage en komplett løsning for hele systemet, uten å veksle mellom ROS versjoner.

Fysisk datainnhenting fra industriområde

En stor del av oppgaven var å lage et bredt datasett gjennom kjøring av Jackal på Stord. Dette skulle Cognite bruke som test-data for utviklingen av en autonom sporingsløsning. Denne jobben er ikke utført, og bør gjennomføres som en del av prosjektets helhet.

Referanser

[1] Cognite. (). "Cognite data fusion | cognite," adresse: https://www.cognite.com/en/cognite-data-fusion (sjekket 5. feb. 2021).

[2] ——, *Cognite + Kvaerner | Robots improving on-site logistics*, 4. sep. 2020. adresse: https://www.youtube.com/watch?v=fkZYSjVNHFc (sjekket 5. feb. 2021).

[3] (). "Exchange Data with ROS Publishers and Subscribers - MATLAB & Simulink - MathWorks Nordic," adresse: https://se.mathworks.com/help/ros/ug/exchange-data-with-ros-publishers-and-subscribers.html (sjekket 27. mai 2021).

[4] (). "cv_bridge/Tutorials/ConvertingBetweenROSImagesAndOpenCVImagesPython - ROS Wiki," adresse: http://wiki.ros.org/cv_bridge/Tutorials/ConvertingBetweenROSImagesAndOpenCVImagesPython (sjekket 27. mai 2021).

[5] ——, (). "Data liberation & integration | cognite," adresse: https://www.cognite.com/en/cognite-data-fusion/data-liberation-and-integration (sjekket 12. feb. 2021).

[6] (). "What is a container? | app containerization | docker," adresse: https://www.docker.com/resources/what-container (sjekket 26. mai 2021).

[7] (). "Bounding box, polygon, cuboids annotation for machine learning," adresse: https://www.infosearchbpo.com/bounding-box-annotation.php (sjekket 13. mai 2021).

[8] (). "Latitude and longitude | definition, examples, diagrams, & facts," Encyclopedia Britannica, adresse: https://www.britannica.com/science/latitude (sjekket 12. mai 2021).

[9] (). "GPS - NMEA sentence information," adresse: http://aprs.gids.nl/nmea/#gll (sjekket 5. feb. 2021).

[10] (16. feb. 2017). "Graticule: Planning a GPS Survey Part 2 – Dilution of Precision Errors," Graticule, adresse: http://graticules.blogspot.com/2017/02/planning-gps-survey-part-2-dilution-of.html (sjekket 12. mai 2021).

[11] (). "LSM9DS1 Breakout Hookup Guide - learn.sparkfun.com," adresse: https://learn.sparkfun.com/tutorials/lsm9ds1-breakout-hookup-guide/lsm9ds1-overview (sjekket 12. mai 2021).

[12] B. Benligiray, C. Topal og C. Akinlar, "STag: A stable fiducial marker system," *Image and Vision Computing*, årg. 89, s. 158–169, 1. sep. 2019, ISSN: 0262-8856. DOI:

10.1016/j.imavis.2019.06.007. adresse: https://www.sciencedirect.com/science/article/pii/S0262885619300903 (sjekket 14. mai 2021).

[13] C. Stachniss, "Camera calibration: Zhang's method," s. 14,

[14] W. Burger, *Zhang's Camera Calibration Algorithm: In-Depth Tutorial and Implementation*. 16. mai 2016.

[15] Elizabeth. (7. apr. 2019). "What is focal length in photography?" Photography Life, adresse: https://photographylife.com/what-is-focal-length-in-photography (sjekket 10. feb. 2021).

[16] E. Vollset, *cognite-sdk: Client library for Cognite Data Fusion (CDF)*, versjon 2.19.0. adresse: https://cognite-sdk-python.readthedocs-hosted.com (sjekket 29. mai 2021).

[17] (). "Simulating Jackal — Jackal Tutorials 0.5.4 documentation," adresse: https://www.clearpathrobotics.com/assets/guides/kinetic/jackal/simulation.html (sjekket 26. mai 2021).

[18] O. Isik, J.-H. Hong, I. Petrunin og A. Tsourdos, "Integrity Analysis for GPS-Based Navigation of UAVs in Urban Environment," *Robotics*, årg. 9, s. 66, 25. aug. 2020. DOI: 10.3390/robotics9030066.

[19] (). "Epoch converter," adresse: https://www.epochconverter.com (sjekket 28. mai 2021).

[20] (). "GPS Visualizer," adresse: https://www.gpsvisualizer.com/ (sjekket 12. mai 2021).

[21] Intel. (). "Depth camera d435," Intel® RealSense™ Depth and Tracking Cameras, adresse: https://www.intelrealsense.com/depth-camera-d435/ (sjekket 19. feb. 2021).

[22] Cognite. (). "API Reference v1 - Cognite documentation," adresse: https://docs.cognite.com/api/v1/ (sjekket 2. mar. 2021).

[23] C. Cognite. (). "SDK documentation - Getting started," adresse: https://docs.cognite.com/dev/guides/sdk/python/ (sjekket 2. mar. 2021).

[24] (). "API platform: API tools & solutions," Postman, adresse: https://www.postman.com/api-platform/ (sjekket 31. mai 2021).

[25] OpenCV. (). "OpenCV: Camera Calibration," adresse: https://docs.opencv.org/master/dc/dbb/tutorial_py_calibration.html (sjekket 12. feb. 2021).

[26] (). "Intel® RealSense™ self-calibration for d400 series depth cameras," Intel® RealSense™ Developer Documentation, adresse: https://dev.intelrealsense.com/docs/self-calibration-for-depth-cameras (sjekket 29. mai 2021).

[27] (). "About the ubuntu project," Ubuntu, adresse: https://ubuntu.com/about (sjekket 14. mai 2021).

[28] S. Pokhrel. (11. mar. 2020). "Image data labelling and annotation — everything you need to know," Medium, adresse: https://towardsdatascience.com/image-data-labelling-and-annotation-everything-you-need-to-know-86ede6c684b1 (sjekket 25. feb. 2021).

[29] H. Dwivedi, *Implementing SSH ®: Strategies for Optimizing the Secure Shell*. Hoboken, UNITED STATES: John Wiley & Sons, Incorporated, 2003, ISBN: 978-0-7645-5725-5. adresse: http://ebookcentral.proquest.com/lib/hogskbergen-ebooks/detail.action?docID=215119 (sjekket 9. feb. 2021).

[30] E. Olson, "AprilTag: A robust and flexible visual fiducial system," s. 8,

[31] (). "Jackal UGV - small weatherproof robot - clearpath," Clearpath Robotics, adresse: https://clearpathrobotics.com/jackal-small-unmanned-ground-vehicle/ (sjekket 5. feb. 2021).

Appendiks

A1 Aktivitetsplan

Aktivitetsplan

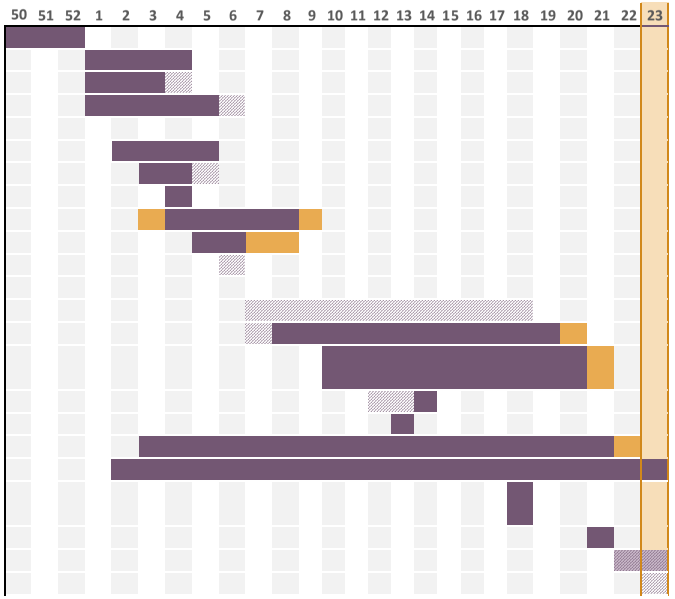
Rød = Milepæl

Bacheloroppgave Cognite BO21E-38

Periodeutheving 23

Faktisk start | % fullført | Faktisk (utover planen) | % fullført (uto

AKTIVITET	PLANLAGT START	PLANLAGT VARIGHET	FAKTISK START	FAKTISK VARIGHET	PROSENT FULLFØRT	Ansvar
Skaffe prosjekt	45	12	45	12	100 %	F&I
Definere prosjekt & Forrarbeid	1	4	1	4	100 %	F&I
Apriltags (fuducial marks), Vslam	1	4	1	3	100 %	F
Kamerakalibrering / OpenCV	1	6	1	5	100 %	I
Suksessful kalibreringstest	TBD	1		0	0 %	I
Forstudie	2	4	2	4	100 %	F&I
Cognite kurs	3	3	3	2	100 %	F&I
Forstudie innlevering (31.1)	4	1	4	1	100 %	F&I
Algoritme for kalibrering	4	5	3	7	100 %	I
Lese opp på ROS	5	2	5	4	100 %	F
Tur til Aker (opplæring i robot)	6	1		0	0 %	F&I
Suksessful datainsamling	TBD	1		0	0 %	F&I
Turer til Aker (datinnhenting)	7	12		0	0 %	F&I
Databehandling	7	13	8	13	100 %	F&I
Opplasting til CDF (Cognite data fusion)	10	11	10	12	100 %	F&I
Midtveis presentasjon (22.3)	12	3	14	1	100 %	F&I
Påskeferie	13	1	13	1	100 %	F&I
Rapport	3	19	3	20	100 %	F&I
Møter med bedrift og veileder	2	22	2	22	100 %	F&I
Seminar (frivillig)(endret til obligatorisk)	18	1	18	1	100 %	F&I
Innlevering (28.5)	21	1	21	1	100 %	F&I
Presentasjon (3-9.6)	22	2	22	2	0 %	F&I
EXPO (10.6) - (utgår)	23	1			0 %	F&I



A2 ROS / Ubuntu kommandoer

SIMULATE JACKAL WITH REALSENSE:

```
export JACKAL_URDF_EXTRAS=\$HOME/Desktop/realsense.urdf.xacro
roslaunch jackal_gazebo jackal_world.launch platform:=jackal
roslaunch jackal_viz view_robot.launch
```

ROSBAG:

```
rosbag record -a #records all available data
rosbag record -O subset /turtle1/cmd_vel /turtle1/pose
```

ROSBAG VISUALIZE:

```
rosbag info <your bagfile> #shows available info in terminal
roscore
rqt_bag <your bagfile> #opens bagfile in gui
```

REALSENSE:

```
roslaunch realsense2_camera rs_camera.launch
rostopic list
```

A3 Installasjon av Ubuntu / ROS

Guide for installasjon av Ubuntu 16.04 er hentet fra www.aksaswiss.com (januar 2021)

https://www.askaswiss.com/2017/01/how-to-install-ubuntu-16-04-alongside-windows-10-dual-boot.html

Prosjekt PC: Asus GL502VM Laptop

```
CPU: Intel Core i5 @ 2.30GHz (Skylake)
RAM: 8GB @ 1064MHz
GPU: Nvidia GeForce RTX 1060 2GB
SSD: Kingston 250GB
```

Partisjonstørrelser:

```
Root (/): 25GB
Swap: 16GB
Home (/home): 200GB
```

Installer ROS Kinetic med å følge ROS tutorial:

http://wiki.ros.org/kinetic/Installation/Ubuntu

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc)
    main" > /etc/apt/sources.list.d/ros-latest.list'

sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80'
    --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654

sudo apt-get update

sudo apt-get install ros-kinetic-desktop-full

echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc source ~/.bashrc

sudo apt install python-rosdep python-rosinstall python-rosinstall-generator
    python-wstool build-essential

sudo apt install python-rosdep

sudo rosdep init

rosdep update
```

Sett opp ROS environment:

http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment

```
source /opt/ros/kinetic/setup.bash

mkdir -p ~/catkin_ws/src

cd ~/catkin_ws/

catkin_make

source devel/setup.bash

echo $ROS_PACKAGE_PATH /home/<youruser>/catkin_ws/src:/opt/ros/kinetic/share
```

A4 Python: Bag Extract Server

```python
#!/usr/bin/env python

import csv
import os
import string

import cv2 as cv
import numpy as np
import rosbag
import rospy
import yaml
from cv_bridge import CvBridge, CvBridgeError
from jackal_edge.srv import BagExtract, BagExtractRequest, BagExtractResponse

ROS_SERVICE_NAME = "bag_extract"

def handle_bag_extract(bagPath):
    extractFolder = export_bag(bagPath)
    rospy.loginfo("bag contents extracted to \"%s\"\n" % extractFolder)
    return BagExtractResponse(extractFolder)

def bag_extract_server():
    rospy.init_node(bag_extract_server.__name__)
    service = rospy.Service(ROS_SERVICE_NAME, BagExtract, handle_bag_extract)
    rospy.loginfo("ready to recieve .bag path\n")
    rospy.spin()

def load_yaml_params():
    basepath = os.path.dirname(__file__)
    filepath = os.path.abspath(os.path.join(basepath, "..", "params", "extract.yaml"))
    try:
        yaml_config_file = open(filepath)
        parsed_yaml_file = yaml.load(yaml_config_file)

        # known topic namese for RGB and DEPTH images
        listOfImages = (parsed_yaml_file["image_topics"])
        listOfDepth = (parsed_yaml_file["depth_topics"])
        # known list of unnecessary/large topics
        filteredTopics = (parsed_yaml_file["large_topics"])

        return listOfImages, listOfDepth, filteredTopics
    except:
        return -1

def read_write_image_topics(folder, bag, listOfImages, listOfDepth):
    # rgb & depth image saver
    bagContents = bag.read_messages(topics=(listOfDepth + listOfImages))
    bridge = CvBridge()

    for topic, msg, t in bagContents:
        head, tail = os.path.split(folder)  # head is path, tail is filename
        bagName = string.replace(tail, '/', '-')
        newTopic = string.replace(topic, '/', '-')
        imFolder = folder + '/' + newTopic
        metaFile = folder + '/' + (bagName + '_' + newTopic + '_list.csv')

        try:  # make folder if none exists
            os.makedirs(os.path.join(folder, newTopic))
            rospy.loginfo("saving {} in {}".format(topic, imFolder))
        except:
            pass

        if listOfImages.__contains__(topic): # if topic is RGB image
            file = bagName + '_' + newTopic + '_' + str(msg.header.stamp) + '.jpeg'
        elif listOfDepth.__contains__(topic): # else if topic is depth image
            file = bagName + '_' + newTopic + '_' + str(msg.header.stamp) + '.npy'

        # write list of created image files to csv
        openMetaFile = open(metaFile, 'ab')
```

```
70          with openMetaFile as f:
71              thewriter = csv.writer(f)
72              if os.path.getsize(metaFile) == 0: # if beginning of file
73                  thewriter.writerow(['topic', 'seqno', 'timeStamp', 'fileName'])
74              thewriter.writerow([str(topic), str(msg.header.seq), str(msg.header.stamp), file])
75
76          # print images
77          try:
78              if listOfImages.__contains__(topic):
79                  cv_img = bridge.imgmsg_to_cv2(msg, msg.encoding)
80                  cv.imwrite(os.path.join(imFolder, file), cv_img)
81
82              # FROM
                https://github.com/nihalsoans91/Bag_to_Depth/blob/master/src/bag2rgbdepth/scripts/
                grabdepth.py
83              elif listOfDepth.__contains__(topic):
84                  cv_image = bridge.imgmsg_to_cv2(msg, msg.encoding)
85                  numpy_image = np.array(cv_image, dtype=np.uint8)
86                  np.save(os.path.join(imFolder, file), numpy_image)
87
88          except CvBridgeError:
89              rospy.loginfo(CvBridgeError)
90
91  def read_write_csv_topics(folder, bag, filteredTopics):
92      # get list of topics from the bag
93      listOfTopics = []
94      rospy.loginfo("topics to save as csv:")
95
96      for topic in bag.get_type_and_topic_info().topics:
97          if str(topic) not in filteredTopics:
98              listOfTopics.append(topic)
99              rospy.loginfo (topic)
100
101     # write topic info to csv file
102     for topicName in listOfTopics:
103         # Create a new csv for each topic
104         filename = folder + '/' + string.replace(topicName, '/', '-') + '.csv'
105         rospy.loginfo("writing {}".format(filename))
106
107         with open(filename, 'a') as csvfile:
108             filewriter = csv.writer(csvfile, delimiter=',')
109             firstIteration = True  # allows header row
110
111             # for each instant in time that has data for topicName
112             for subtopic, msg, t in bag.read_messages(topicName):
113                 # parse data from this instant, which is of the form of multiple lines of
                    "Name: value\n"
114                 #    - put it in the form of a list of 2-element lists
115                 msgString = str(msg)
116                 msgList = string.split(msgString, '\n')
117                 instantaneousListOfData = []
118
119                 for nameValuePair in msgList:
120                     splitPair = string.split(nameValuePair, ':')
121
122                     for i in range(len(splitPair)):  # should be 0 to 1
123                         splitPair[i] = string.strip(splitPair[i])
124
125                     instantaneousListOfData.append(splitPair)
126
127                 # write the first row from the first element of each pair
128                 if firstIteration:  # header
129                     headers = ["rosbagTimestamp"]  # first column header
130
131                     for pair in instantaneousListOfData:
132                         headers.append(pair[0])
133
134                     filewriter.writerow(headers)
135                     firstIteration = False
```

```
136
137                 # write the value from each pair to the file, first column will have rosbag
                    timestamp
138                 values = [str(t)]
139                 for pair in instantaneousListOfData:
140                     if len(pair) > 1:
141                         values.append(pair[1])
142
143                 filewriter.writerow(values)
144
145
146 def export_bag(bagExtractInput):
147     bagSource = bagExtractInput.bagPath
148     rospy.loginfo("reading bagfile: " + str(bagSource))
149
150     # access bag
151     try:
152         bag = rosbag.Bag(bagSource)
153
154         # check bag compression, unpacking of comressed bag is EXTREMELY slow
155         if not (bag.get_compression_info().compressed  == bag.get_compression_info().
            uncompressed):
156             return "ERROR: the bag you have entered is compressed, please decompress before
                continuing"
157
158     except rosbag.ROSBagException as e:
159         return "ERROR: faulty bag file: %s" % e
160
161     # read config file
162     try:
163         listOfImages, listOfDepth, filteredTopics = load_yaml_params()
164     except:
165         return "ERROR: cannot load config file ../params/extract.yaml."
166
167     bagDir = bag.filename
168     # create a new directory
169     newFolder = string.rstrip(bagDir, ".bag")
170     # make folder else already exists
171     try:
172         os.makedirs(newFolder)
173     except OSError as e:
174         rospy.loginfo(e)
175
176     read_write_csv_topics(newFolder, bag, filteredTopics)
177     read_write_image_topics(newFolder, bag, listOfImages, listOfDepth)
178
179     bag.close()
180
181     return newFolder
182
183 if __name__ == "__main__":
184     try:
185         bag_extract_server()
186     except rospy.ROSInterruptException as e:
187         rospy.loginfo(e)
188
```

A5 Python: Bag Extract Client

```
#!/usr/bin/env python

from __future__ import print_function
import sys
import rospy
from jackal_edge.srv import BagExtract, BagExtractRequest, BagExtractResponse

ROS_SERVICE_NAME = "bag_extract"

def bag_extract_client(bagPath):
    rospy.wait_for_service(ROS_SERVICE_NAME)
    try:
        bag_extract = rospy.ServiceProxy(ROS_SERVICE_NAME, BagExtract)
        return bag_extract(bagPath)
    except rospy.ServiceException as e:
        rospy.loginfo("service call failed: %s" % e)

def usage_message():
        # prints intended command format to user ( $python bag_extract_client.py
        "/home/user/../bagName.bag" )
        print("%s \"/home/user/../bagName.bag\"" % (ROS_SERVICE_NAME + "_client.py"))

if __name__ == "__main__":
    # get input arguments
    if len(sys.argv) == 2:
        bagPath = str(sys.argv[1])
    else:
        usage_message()
        sys.exit(1)
    rospy.loginfo("requesting extract of %s" % (bagPath))
    rospy.loginfo("extraction of %s saved in %s" % (bagPath, bag_extract_client(bagPath)))

```

A6 Python: Bag Extract - yaml

```yaml
# For configuration of ros topics for BagExtract rosservice
# BagExtract service need list of image and depth topics to save them as jpg and npy

# Current known RBG and Depth topics
# Add more if new cameras are installed
# NOTE: topics added to this list should allso be added to large_topics, to prevent writing them to csv
image_topics:
  - /realsense/color/image_raw
  - /realsense_right/color/image_raw

depth_topics:
  - /realsense/depth/image_rect_raw
  - /realsense_right/depth/image_rect_raw

# "large_topics" is used to prevent certain topics from beeing writen to csv
# realsense topics are in this list because image and depth topics become 10x ++ larger when writen to csv vs jpg
#
# If topics ARE in this list, and NOT in image_ or depth_topics, they won't be extracted from the rosbag
#
# Leave as is, unless you spesifically want to extract the topic to csv
large_topics:
  - /realsense/aligned_depth_to_color/image_raw
  - /realsense_right/aligned_depth_to_color/image_raw
  - /realsense/color/image_raw
  - /realsense_right/color/image_raw
  - /realsense/depth/image_rect_raw
  - /realsense_right/depth/image_rect_raw
  - /realsense/color/image_raw/compressed
  - /realsense/color/image_raw/theora
  - /realsense/depth/color/points
  - /realsense_right/color/image_raw/compressed
  - /realsense_right/color/image_raw/theora
  - /realsense_right/depth/color/points

```

A7 Python: Bag to Waypoint Server

```python
#!/usr/bin/env python

from jackal_edge.srv import BagToWaypoints, BagToWaypointsRequest, BagToWaypointsResponse

import rosbag
import string
import os
import rospy
import json
import geopy.distance
import yaml

ROS_SERVICE_NAME = "bag_to_waypoints"

def handle_bag_to_waypoints(bagPath):
    getWaypointsFolder = main_bag_to_waypoints(bagPath)
    rospy.loginfo("returning waypoints in \"%s\"\n" % getWaypointsFolder)
    return BagToWaypointsResponse(getWaypointsFolder)

def bag_to_waypoints_server():
    # start server node
    rospy.init_node(bag_to_waypoints_server.__name__)
    service = rospy.Service(ROS_SERVICE_NAME, BagToWaypoints, handle_bag_to_waypoints)
    rospy.loginfo("Ready to receive .bag path\n")
    rospy.spin()

def get_data_table_from_bag(bagSource):
    # open bag if possible
    try:
        bag = rosbag.Bag(bagSource)
    except rosbag.ROSBagException as e:
        rospy.loginfo("No such .bag: %s" % e)

    # get relevant bag topic and write to list
    bagContents = bag.read_messages(topics="/navsat/fix")
    topicTable = []

    # separates message into individual components in a list
    for subtopic, message, time in bagContents:
        if subtopic == "/navsat/fix":
            messageString = str(message)
            # split message for each newline
            messageList = string.split(messageString, '\n')

            for nameValuePair in messageList:
                # split header and message value. {"example" : "1.93"} to ["example", "1.93"]
                splitPair = string.split(nameValuePair, ':')

                for i in range(len(splitPair)):
                    splitPair[i] = string.strip(splitPair[i])
                topicTable.append(splitPair)
    bag.close()

    return topicTable

def load_yaml_params(jsonDict):
    def waypoints_default_structure():
        jsonDict["items"].append({"_id": "exampleID",
                                  "name": "jackalWaypoints",
                                  "inspectionPoints": [],
                                  "projectId": "exampleProjectID",
                                  "locationId": "exampleLocationID",
                                  "waypoints": [] })

    basepath = os.path.dirname(__file__)
    filepath = os.path.abspath(os.path.join(basepath, "..", "params", "waypoints.yaml"))

    jsonDict["items"] = []

```

```
70          try:
71              yaml_config_file = open(filepath)
72              parsed_yaml_file = yaml.load(yaml_config_file)
73
74              jsonDict["items"].append({"_id": parsed_yaml_file["items"]["_id"],
75                                        "name": parsed_yaml_file["items"]["name"],
76                                        "inspectionPoints": parsed_yaml_file["items"]["inspectionPoints"],
77                                        "projectId": parsed_yaml_file["items"]["projectId"],
78                                        "locationId": parsed_yaml_file["items"]["locationId"],
79                                        "waypoints": [] })
80
81              yaml_config_file.close()
82
83          except UnboundLocalError:
84              rospy.loginfo("ERROR: No waypoints.yaml params file found, using default values")
85              waypoints_default_structure()
86
87          except IOError:
88              rospy.loginfo("ERROR: No waypoints.yaml params file found, using default values")
89              waypoints_default_structure()
90
91          return jsonDict
92
93
94      def create_waypoints_files(bagSource):
95          newFolder = string.rstrip(bagSource, (".bag")) + "-waypoints"
96
97          # make folder else already exists
98          try:
99              os.makedirs(newFolder)
100         except OSError as e:
101             rospy.loginfo(e)
102
103         jsonFile = os.path.join((newFolder), "waypoints.json")
104         txtFile = os.path.join((newFolder), "waypointsAsList.txt")
105
106         # delete old files if exists
107         try:
108             os.remove(jsonFile)
109             os.remove(txtFile)
110         except OSError as e:
111             rospy.loginfo(e)
112         except IOError as e:
113             rospy.loginfo(e)
114
115         jsonDict = {}
116         jsonDict = load_yaml_params(jsonDict)
117
118         # creates waypoints.json, dump setup
119         dump_to_json_file(jsonDict, jsonFile)
120
121         #create text fil
122         txtWriter=open(txtFile,'w')
123         txtWriter.write("name,latitude,longitude\n")
124         txtWriter.close()
125
126         return jsonFile, txtFile, newFolder
127
128     def main_bag_to_waypoints(BagToWaypointsInput):
129         # get input variables from service call
130         bagSource = BagToWaypointsInput.bagPath
131         metersBetweenWaypoints = BagToWaypointsInput.metersBetweenPoints
132
133         topicTable = get_data_table_from_bag(bagSource)
134
135         # check if topic exists
136         if len(topicTable) > 0:
137             # create folder and files for waypoints
138             jsonFile, txtFile, newFolder = create_waypoints_files(bagSource)
```

```
139
140             # filewriter
141             txtWriter=open(txtFile,'a+')
142             txtFile_NameIndex = 0
143
144             # get latitude and longitude from list
145             latitude = []
146             longitude = []
147             for x in topicTable:
148                 if x[0] == "latitude":
149                     latitude.append(x[1])
150                 if x[0] == "longitude":
151                     longitude.append(x[1])
152
153             # write first GPS point
154             waypointsList = [{"lat": latitude[0], "lon": longitude[0]}]
155             txtWriter.write(str(txtFile_NameIndex) + "," + str(latitude[0]) + "," + str(longitude
                [0]) + "\n")
156             txtFile_NameIndex += 1
157
158             # Add intermediate coordinates, filtered by metres
159             distanceBetweenPoints = 0.0
160             for i in range(1, len(longitude)):
161                 if not metersBetweenWaypoints == 0:
162                     coordinateFrom = (latitude[i-1], longitude[i-1])
163                     coordinateTo = (latitude[i], longitude[i])
164                     # calculate direct distance between two points, add to total distance
165                     distanceBetweenPoints += geopy.distance.distance(coordinateFrom, coordinateTo
                        ).meters
166
167                     # if direct distance is at choosen distanse, add waypoint, reset distance
168                     if distanceBetweenPoints >= metersBetweenWaypoints:
169                         waypointsList.append({"lat": latitude[i-1], "lon": longitude[i-1]})
170                         txtWriter.write(str(txtFile_NameIndex) + "," + str(latitude[i-1]) + "," +
                             str(longitude[i-1]) + "\n")
171                         txtFile_NameIndex += 1
172                         distanceBetweenPoints = 0.0
173
174                 else: # add all points if accuracy is 0
175                     waypointsList.append({"lat": latitude[i-1], "lon": longitude[i-1]})
176                     txtWriter.write(str(txtFile_NameIndex) + "," + str(latitude[i-1]) + "," + str
                        (longitude[i-1]) + "\n")
177                     txtFile_NameIndex += 1
178
179             # always adds last coordinate
180             waypointsList.append({"lat": latitude[len(latitude) - 1], "lon": longitude[len(
                longitude)-1]})
181             txtWriter.write(str(txtFile_NameIndex) + "," + str(latitude[len(latitude)-1]) + "," +
                 str(longitude[len(longitude)-1]) + "\n")
182             txtWriter.close()
183
184             # append new waypoints data to json file
185             with open(jsonFile) as json_file:
186                 jsonLoader = json.load(json_file)
187                 itemsWaypoints = jsonLoader["items"]
188                 waypointsData = itemsWaypoints[0]["waypoints"]
189                 waypointsData += waypointsList
190
191             dump_to_json_file(jsonLoader, jsonFile)
192
193             rospy.loginfo("succesfully read GPS coordinates from \"{}\" with {} metres between
                points".format(bagSource, metersBetweenWaypoints))
194             return(newFolder)
195         else:
196             return("No waypoints created, missing /navsat/fix")
197
198     def dump_to_json_file(data, filename):
199         with open(filename, "w") as f:
200             json.dump(data, f, indent=4)
```

```
201
202 if __name__ == "__main__":
203     try:
204         bag_to_waypoints_server()
205     except rospy.ROSInterruptException as e:
206         rospy.loginfo(e)
207
208
```

A8 Python: Bag to Waypoint Client

```python
#!/usr/bin/env python

from __future__ import print_function
import sys
import rospy
from jackal_edge.srv import BagToWaypoints, BagTowaypointsRequest, BagTowaypointsResponse

ROS_SERVICE_NAME = "bag_to_waypoints"

def bag_to_waypoints_client(bagPath, metersBetweenPoints):
    rospy.wait_for_service(ROS_SERVICE_NAME)
    try:
        #Declare service name and inputs
        SERVICE = rospy.ServiceProxy(ROS_SERVICE_NAME, BagToWaypoints)
        return SERVICE(bagPath, metersBetweenPoints)
    except rospy.ServiceException as e:
        rospy.loginfo("service call failed: %s" % e)

def error_message():
        print("%s \"/home/user/../bagName.bag\" metersBetweenPoints" % (ROS_SERVICE_NAME +
        "_client.py"))

if __name__ == "__main__":
    # get number of input arguments
    # 3 args means this was the inut: $python myCode.py arg2 arg3
    if len(sys.argv) == 3:
        bagPath = str(sys.argv[1])
        try:
            metersBetweenPoints = int(sys.argv[2])
        except ValueError:
            print("Error: enter int value for metersBetweenPoints")
            error_message()
            sys.exit(1)
        except IndexError:
            print("Error: enter int value for metersBetweenPoints")
            error_message()
            sys.exit(1)
    elif len(sys.argv) == 2:
        bagPath = str(sys.argv[1])
        metersBetweenPoints = 1
        print("No value given for metersBetweenPoints, using 1 meter")
    else:
        print("wrong input argument, use:")
        error_message()
        sys.exit(1)

    rospy.loginfo("requesting waypoints from %s" % (bagPath))
    rospy.loginfo("waypoints from %s created in %s" %(bagPath, bag_to_waypoints_client(
    bagPath, metersBetweenPoints)))


```

A9 Python: Bag to Waypoint - yaml

```
# For configuration of output waypoints.json file
# Set desired values for all parameters

items:
    "_id": "exampleID007"
    "name": "Plaza Lap"
    "inspectionPoints": []
    "projectId": "robotics-playground"
    "locationId": "cognite-head-office-plaza"


```

A10 Python: Upload Server

```
#!/usr/bin/env python

import os
from datetime import datetime

import pandas as pd
import rospy
from cognite.client import CogniteClient

from jackal_upload.srv import (BagFolderUpload, BagFolderUploadRequest,
BagFolderUploadResponse)

## This program are uploading csvfiles, picture(RGB and depth) and timeseries to robotics
playground

#The assetID for upload to timeseries for Imudata is shown below:
#The x/x.1/x.2 is how python reads the csv files
# Orientation(x, y, z) angular_velocity(x.1, y.1, z.1) linear_acceleration(x.2, y.2, z.2)
ID_TIMESERIES_IMU = dict([("x", 3075430807978820), ("y", 4304526979076438), ("z",
6544591070376320),
                      ("x.1", 4970236671420955), ("y.1", 6057580012294914), ("z.1",
                      3820907703163096),
                      ("x.2", 275935312892255), ("y.2", 5286559232844084), ("z.2",
                      7126410338691453)])

#The assetID for upload to timeseries for GPS data is shown below:
ID_TIMESERIES_GPS = dict([("latitude", 1944609220734914), ("longitude", 6495144992165154), (
"altitude", 3142047466296944)])

#The asset ID for uploading picture(RGB and depth) to CDF for both camera
ID_ASSET_CAMERAS = dict([("-realsense-color-image_raw", 648613141372344), (
"-realsense_right-color-image_raw", 3295790077107133),
                      ("-realsense-depth-image_rect_raw", 5916255947188486), (
                      "-realsense_right-depth-image_rect_raw", 3340633696030551)])

#The asset ID for jackal, this is where the csv files wil be uploaded to
ID_ASSET_JACKAL = 4458655741152877

ROS_SERVICE_NAME = "bag_folder_upload"
PROJECT_ID = "robotics-playground"
API_KEY_ENVIRONMENT = "COGNITE_API_KEY"
CLIENT = "Jackal_Upload"

def handle_bag_folder_upload(bagPath):
    folderPath = bagPath.extractedBagFolder
    uploadRequest = main_upload_folder(folderPath)
    rospy.loginfo("bag contents uploaded: \"%s\"\n" % uploadRequest)
    return BagFolderUploadResponse(uploadRequest)

def bag_folder_upload_server():
    rospy.init_node(bag_folder_upload_server.__name__)
    service = rospy.Service(ROS_SERVICE_NAME, BagFolderUpload, handle_bag_folder_upload)
    rospy.loginfo("ready to recieve .bag path\n")
    rospy.spin()

def main_upload_folder(bagPath):
    if c.login.status().logged_in == True:
        if (os.path.isdir(bagPath)):
            head, bagName = os.path.split(bagPath)  # tail is bag name
            ## upload sequence ##
            rospy.loginfo("\n----\n")
            upload_timeseries(bagPath)
            rospy.loginfo("\n----\n")
            upload_image(bagPath, bagName)
            rospy.loginfo("\n----\n")
            upload_csv(bagPath, bagName)
            rospy.loginfo("\n----\n")
            return("Successful upload")
        else:
```

```
62              return("Failed upload: No such path: {}".format(bagPath))
63
64      else:
65          return("Failed upload: Can't access CDF")
66
67  # updates known TimeSeries with GPS & IMU data from bag
68  def upload_timeseries(bagPath):
69      direct = bagPath
70
71      #Topics we are intrested in for upload to CDF
72      listOfTopics = ['-imu-data.csv', '-navsat-fix.csv']
73
74      def df_upload(tab, l):  # make DataFrame from dataFile, update relevant TimeSeries
75          t = tab["rosbagTimestamp"]
76          tArray = []
77          for value, idC in l.items():
78              for Y in t:  # epoch2date (YYYY-mm-dd HH:MM:SS.ffffff)
79                  tArray.append(datetime.utcfromtimestamp(
80                      Y/1000000000).strftime('%Y-%m-%d %H:%M:%S.%f'))
81
82              data = tab[value]
83              df = pd.DataFrame({idC: data.values}, index=tArray)
84
85              # c.datapoints.insert_dataframe(df)
86
87              rospy.loginfo(df)
88              tArray.clear()
89
90      if os.path.exists(os.path.join(direct, listOfTopics[0])):
91          tab = pd.read_csv(os.path.join(direct, listOfTopics[0]))
92
93          #Starts the class df_upload with input -imu-data.csv and the IMU headernames and
            assets IDs
94          df_upload(tab, ID_TIMESERIES_IMU)
95          rospy.loginfo("upload of IMU data to TimeSeries complete\n")
96
97      if os.path.exists(os.path.join(direct, listOfTopics[1])):
98          tab = pd.read_csv(os.path.join(direct, listOfTopics[1]))
99
100         #Starts the class df_upload with input -navsat-fix.csv and the GPS headernames
            assets IDs
101         df_upload(tab, ID_TIMESERIES_GPS)
102         rospy.loginfo("upload of GPS data to TimeSeries complete\n")
103
104 # uploads RGB images and Depth array from bag, asset is Jackal/Camera/location/imagetype
105 def upload_image(bagPath, bagName):
106     direct = bagPath
107
108     for topic, idC in ID_ASSET_CAMERAS.items():
109         d = (os.path.join(direct, topic))
110         if os.path.isdir(d):
111             if topic.__contains__("depth"):
112                 mim = "application/npy"
113             else:
114                 mim = "image/jpeg"
115
116             # res = c.files.upload(d, asset_ids=[idC], mime_type=mim, source=bagName)
117
118             rospy.loginfo("%s, %s, %s, %s" %(d, idC, mim, bagName))
119             rospy.loginfo('upload of {} complete\n'.format(topic))
120
121 # uploads all files in base folder as csv files, with Jackal as asset
122 def upload_csv(bagPath, bagName):
123     direct = bagPath
124     mim = "text/csv"
125
126     # res = c.files.upload(direct, asset_ids=[ID_ASSET_JACKAL], mime_type=mim, source=
        bagName)
127
```

```
128         rospy.loginfo("%s, %s, %s, %s" %(direct, ID_ASSET_JACKAL, mim, bagName))
129         rospy.loginfo('upload of csv files in {} complete\n'.format(direct))
130 
131     if __name__ == "__main__":
132         c = CogniteClient(api_key=os.getenv(API_KEY_ENVIRONMENT), project=PROJECT_ID,
133                           client_name=CLIENT, disable_pypi_version_check=True)
134 
135         try:
136             bag_folder_upload_server()
137         except rospy.ROSInterruptException as e:
138             rospy.loginfo(e)
139 
140     # def load_yaml_params():
141     #     basepath = os.path.dirname(__file__)
142     #     filepath = os.path.abspath(os.path.join(basepath, "..", "params", "extract.yaml"))
143     #     try:
144     #         yaml_config_file = open(filepath)
145     #         parsed_yaml_file = yaml.load(yaml_config_file)
146 
147     #         # known topic namese for RGB and DEPTH images
148     #         listOfImages = (parsed_yaml_file["image_topics"])
149     #         listOfDepth = (parsed_yaml_file["depth_topics"])
150     #         # known list of unnecessary/large topics
151     #         filteredTopics = (parsed_yaml_file["large_topics"])
152 
153     #         return listOfImages, listOfDepth, filteredTopics
154     #     except:
155     #         return -1
156 
157 
```

A11 Python: Upload Client

```
1  #!/usr/bin/env python
2
3  from __future__ import print_function
4  import sys
5  import rospy
6  from jackal_upload.srv import BagFolderUpload, BagFolderUploadRequest, BagFolderUploadResponse
7
8  ROS_SERVICE_NAME = "bag_folder_upload"
9
10 def bag_folder_upload_client(bagPath):
11     rospy.wait_for_service(ROS_SERVICE_NAME)
12     try:
13         bag_upload = rospy.ServiceProxy(ROS_SERVICE_NAME, BagFolderUpload)
14         return bag_upload(bagPath)
15     except rospy.ServiceException as e:
16         rospy.loginfo("service call failed: %s" % e)
17
18 def usage_message():
19         # prints intended command format to user ( $python bag_extract_client.py
           "/home/user/../bagName.bag" )
20         print("%s \"/home/user/../bagName.bag\"" % (ROS_SERVICE_NAME + "_client.py"))
21
22 if __name__ == "__main__":
23     # get input arguments
24     if len(sys.argv) == 2:
25         bagPath = str(sys.argv[1])
26     else:
27         usage_message()
28         sys.exit(1)
29     rospy.loginfo("requesting upload of %s" % (bagPath))
30     rospy.loginfo("upload of %s complete as %s" % (bagPath, bag_folder_upload_client(bagPath
       )))
31
```

A12 Python: Upload - yaml

```yaml
## This program are uploading csvfiles, picture(RGB and depth) and timeseries to robotics playground

#The assetID for upload to timeseries for Imudata is shown below:
#The x/x.1/x.2 is how python reads the csv files
# Orientation(x, y, z) angular_velocity(x.1, y.1, z.1) linear_acceleration(x.2, y.2, z.2)
ID_TIMESERIES_IMU :
  "x": xxxxxxxxxxxx
  "y": xxxxxxxxxxxx
  "z": xxxxxxxxxxxx
  "x.1": xxxxxxxxxxxx
  "y.1": xxxxxxxxxxxx
  "z.1": xxxxxxxxxxxx
  "x.2": xxxxxxxxxxxx
  "y.2": xxxxxxxxxxxx
  "z.2": xxxxxxxxxxxx

#The assetID for upload to timeseries for GPS data is shown below:
ID_TIMESERIES_GPS:
  "latitude": xxxxxxxxxxxx
  "longitude": xxxxxxxxxxxx
  "altitude": xxxxxxxxxxxx

#The asset ID for uploading picture(RGB and depth) to CDF for both camera
ID_ASSET_CAMERA:
  "-realsense-color-image_raw": xxxxxxxxxxxx
  "-realsense_right-color-image_raw": xxxxxxxxxxxx
  "-realsense-depth-image_rect_raw": xxxxxxxxxxxx
  "-realsense_right-depth-image_rect_raw": xxxxxxxxxxxx


#The asset ID for jackal, this is where the csv files wil be uploaded to
ID_ASSET_JACKAL:
  - xxxxxxxxxxxx
```

A13 Python: Kamerakalibrering

```python
#This code i made for camera calibration. You have to input
#muliple picture with square pattern(like a chessboard).
#The code has been made with help from the tutorial Camera Calibration on open CV
#https://docs.opencv.org/master/dc/dbb/tutorial_py_calibration.html

import numpy as np
import cv2 as cv
import glob
import time


#board size
length = 9  #squares
width = 6
square_size = 0.0253 # meters

#termination criteria
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)

# prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
objp = np.zeros((width*length,3), np.float32)
objp[:,:2] = np.mgrid[0:length,0:width].T.reshape(-1,2)

# Arrays to store object points and image points from all the images.
objpoints = [] # 3d point in real world space
imgpoints = [] # 2d points in image plane.

image = glob.glob('calib_*.png')


for fname in image:
    img = cv.imread(fname)
    gray = cv.cvtColor(img,cv.COLOR_BGR2GRAY)


    # Find the chess board corners
    ret, corners = cv.findChessboardCorners(gray, (length,width),None)

    # If found, add object points, image points (after refining them)
    if ret == True:
        objpoints.append(objp)

        corners2 = cv.cornerSubPix(gray,corners,(11,11),(-1,-1),criteria)
        imgpoints.append(corners2)

        # Draw and display the corners
        img = cv.drawChessboardCorners(img, (length,width), corners2,ret)
        cv.imshow('Images for calibration',img)
        cv.waitKey(500)

ret, mtx, dist, rvecs, tvecs = cv.calibrateCamera(objpoints, imgpoints, gray.shape[::-1],None
,None)

print(' ')
print('Kameramatise: ')
print(mtx)
print(' ')
print('Distortionkoeffisient (k1 k2 p1 p2 k3) :')
print(dist)     #distortioncoeffisient
print(' ')
print('Rotasjonsmatisen:')
print(rvecs)    #rotation matrix from the camera to the calibraton picture(chess board)
print(' ')
print('Transformasjonsmatisen:')
print(tvecs)     #transformation to the calibration picture(chessboard)
print(' ')


scaling = 0      #If the scaling parameter alpha=0, it returns undistorted image with
```

```
69                  #minimum unwanted pixels. So it may even remove some pixels at image
                    corners.
70                  #If alpha=1, all pixels are retained with some extra black images.
71  img = cv.imread('calib_2.png')
72  h,  w = img.shape[:2]
73  newcameramtx, roi=cv.getOptimalNewCameraMatrix(mtx,dist,(w,h),scaling,(w,h))
74
75  # undistort
76  mapx,mapy = cv.initUndistortRectifyMap(mtx,dist,None,newcameramtx,(w,h),5)
77  dst = cv.remap(img,mapx,mapy,cv.INTER_LINEAR)
78
79  # crop the image
80  x,y,w,h = roi
81  dst = dst[y:y+h, x:x+w]
82  cv.imwrite('calibresult.png',dst)
83
84  #cv.destroyAllWindows()
85
86  # re-projection error
87  mean_error = 0          #gives a estimation on how good the parameters are (closer to zero
    then more accurate)  less than 0.2 is good
88  for i in range(len(objpoints)):
89      imgpoints2, _ = cv.projectPoints(objpoints[i], rvecs[i], tvecs[i], mtx, dist)
90      error = cv.norm(imgpoints[i], imgpoints2, cv.NORM_L2)/len(imgpoints2)
91      mean_error += error
92  print( "Total error: {}".format(mean_error/len(objpoints)) )
93  print(' ')
94
95  #lagre til fil:
96  Filnavn='Kalibreringsmatrise.txt'
97  Savefile=open(Filnavn,'w')
98
99  Savefile.write('Kameramatrise: \n'+str(mtx))
100 Savefile.write('\n\nDistortionkoeffisient (k1 k2 p1 p2 k3) :\n'+str(dist))
101 Savefile.write('\n\nRotasjonsmatisen: '+str(rvecs))
102 Savefile.write('\n\nTransformasjonsmatrisen: '+str(tvecs))
103 Savefile.write('\n\nTotal error: '+ str(mean_error/len(objpoints)))
104 Savefile.close()
105
106 print('Fil med parameter er lagret som: '+Filnavn )
```

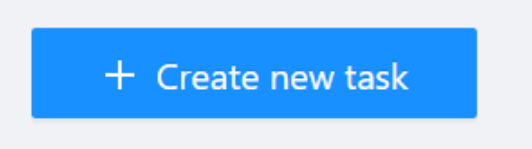
A14 Prosedyre for data annotasjon

Formål:

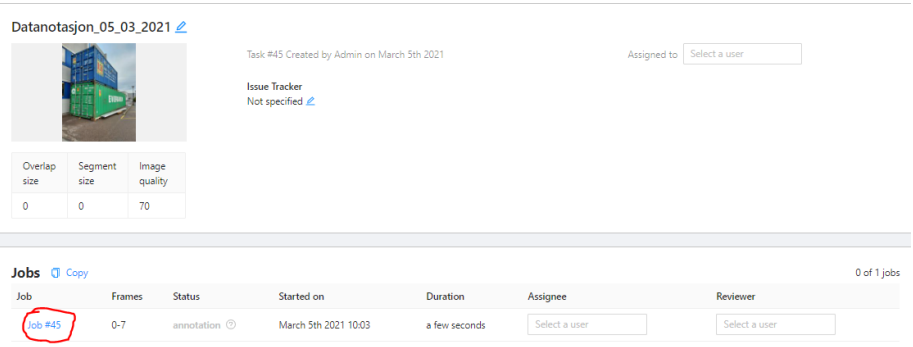
Formålet med prosedyren er å sørge for at data annotasjonen blir gjennomført riktig. Dette skal brukes i en deteksjonsalgoritme som skal detektere containere. Programvaren som blir benyttet heter CVAT og den kjører i samarbeid med Docker desktop. Dette må være på plass før en kan stegvis gjennomføre prosedyren.

Fremgangsmåte

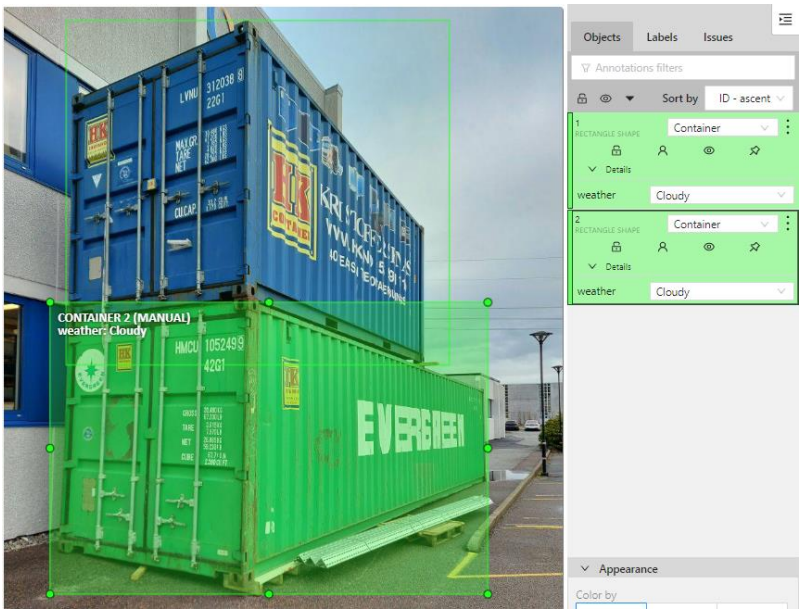
1. Lagre bildene i Cognites google drive.
2. Starte opp Docker desktop og Cvat med følgende link: http://localhost:8080/tasks
3. Gå inn i det ønskede prosjektet (Yard robotics 2021)
4. Tykk på "Crate new task"



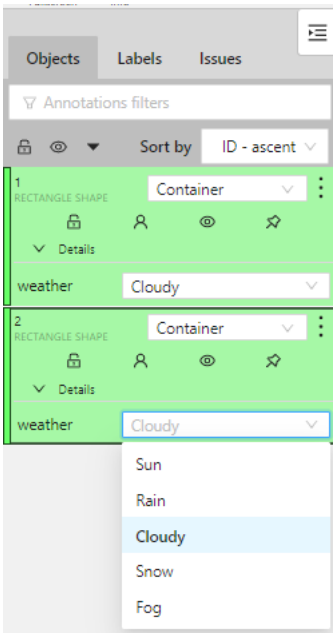
5. Skriv inn ønsket navn (gjerne med datoen datanotasjonen gjennomføres) eksempel: Datanotasjon 17_02_2021
   Velg ønskede filer som en vil benytte i "tasken". Her må en markere de bildene en skal benytte samtidig. (bruk "CTRL" når en velger eventuell dra over et vindu med datamusen)
   Trykk "submit"
6. En vil så få beskjeden "The task has been created" i høyre hjørne. Her kan en velge "open task", eventuelt kan en finne tasken i prosjektet.
7. Åpne Tasken og det vil se slik ut:



8. Trykk så på "Jobb#*" for å starte data annotasjonen
9. Marker hele container med verktøyet "**Draw new rectangle**" sørg for å få hele containeren innenfor rektangelet (Se bilde). Dersom det er flere containere, må hver enkel få sin egen rektangel rundt seg med "labelen": Container

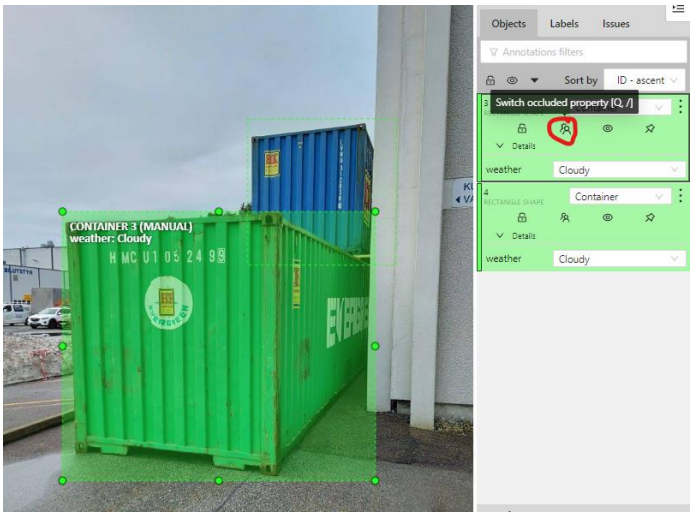


10. En vil så få opp følgende vindu dersom en trykker på “Details” her må en markere for hvordan været er.

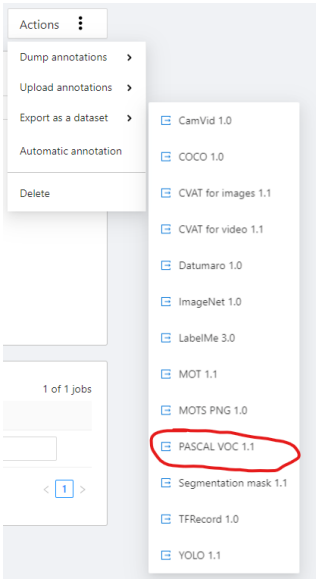


11. En annen ting som er viktig for data annotasjonen er å krysse av for om en ser hele containeren eller om den er tildekket. Som en ser på bilde under ser en ikke hele containerne. Da må vi markere de som “occluded property”(tildekket). Dette gjør en ved å

trykke på symbolet som er merket med rød strek rundt.



12. Gjør så denne rutinen på alle containerne i bilde. Gjenta step 9-11 for hvert bilde.
13. Når dette er gjort, gå inn i “menu” som ligger oppe til venstre og velg “Finish the job”.
14. For å dele data annotasjonen videre trykk på “Actions” velg “Export as a dataset” og trykk på “PASCAL VOC 1.1”



15. En vil da få en Zip fil med en xml og jpg fil for hvert bilde. Dette blir så brukt til opplæring av en algoritme for gjenkjenning av containere.

Eksempel på informasjonen som xml filen inneholder er vist under:

Dette er kode som er generert fra de første eksempelbilde i prosedyren.

```
<?xml version="1.0"?>
<annotation>
    <folder/>
    <filename>20210302.jpg</filename>
    <source>
        <database>Unknown</database>
        <annotation>Unknown</annotation>
        <image>Unknown</image>
    </source>
    <size>
        <width>867</width>
        <height>1156</height>
        <depth/>
    </size>
    <segmented>0</segmented>
    <object>
        <name>Container</name>
        <occluded>0</occluded>
        <bndbox>
            <xmin>96.6</xmin>
            <ymin>21.65</ymin>
            <xmax>691.39</xmax>
            <ymax>556.5</ymax>
        </bndbox>
        <attributes>
            <attribute>
                <name>weather</name>
                <value>Cloudy</value>
            </attribute>
        </attributes>
    </object>
    <object>
        <name>Container</name>
        <occluded>0</occluded>
        <bndbox>
            <xmin>71.4</xmin>
            <ymin>459.3</ymin>
            <xmax>751.3</xmax>
            <ymax>912.6</ymax>
        </bndbox>
        <attributes>
            <attribute>
                <name>weather</name>
                <value>Cloudy</value>
            </attribute>
        </attributes>
    </object>
</annotation>
```