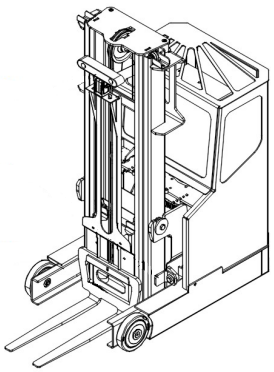
Designspecifikation
Autonom styrning av gaffeltruck

Version 1.0

L.A.M.A.
12 oktober 2016



Status

Granskad	Samtliga projektmedlemmar	2016-10-05
Godkänd	Andreas Bergström	2016-10-12

Projektidentitet

Gruppmail:	jenst280@student.liu.se
Hemsida:	http://www.isy.liu.se/edu/projekt/reglerteknik/2016/forklift/
Beställare:	Andreas Bergström, ISY, Linköpings universitet **Telefon:** +46 (0)10-711 54 54, **Mail:** andreas.bergstrom@liu.se
Kund:	Emil Selse, Toyota Material Handling **Telefon:** +46 (0)702032254 , **Mail:** emil.selse@toyota-industries.eu
Kursansvarig:	Daniel Axehill, ISY, Linköpings universitet **Telefon:** +46 (0)13 284042, **Mail:** daniel@isy.liu.se
Projektledare:	Jenny Stenström
Handledare:	Erik Hedberg, ISY, Linköpings universitet **Telefon:** +46 (0)13 281338 , **Mail:** erik.hedberg@liu.se Samuel Lindgren, Toyota Material Handling **Telefon:** +46 (0)767614024 , **Mail:** samuel.lindgren@toyota-industries.eu

Gruppmedlemmar

Namn	Ansvarsområde	Telefon	Mail (@student.liu.se)
Johan Almgren	Testansvarig	070-206 72 45	johal611
Henrik Andersson	Dokumentansvarig	073-854 77 79	henan562
Gustav Elingsbo	Informationsansvarig	073-685 19 75	gusel411
Mikael Hartman	Integrationsansvarig	076-771 13 38	mikha130
Petter Landerhed	Designansvarig	070-627 82 52	petla189
Andreas Norén	Mjukvaruansvarig	072-394 89 95	andno111
Jenny Stenström	Projektledare	070-329 92 21	jenst280

Dokumenthistorik

Version	Datum	Ändringar	Utförd av	Granskare
1.0	2016-10-12	Första versionen.	L.A.M.A	Samtliga projektmedlemmar
0.1	2016-10-05	Första utkast.	L.A.M.A.	Samtliga projektmedlemmar

Innehåll

1 Inledning

Detta dokument är en designspecifikation för projektet *Planering och Sensorfusion för autonom truck* i kursen *TSRT10 - Reglerteknisk projektkurs, CDIO* som ges vid Linköpings Universitet. Designspecifikationen ger en detaljerad beskrivning över hur varje delsystem ska fungera och även en övergripande bild av systemet.

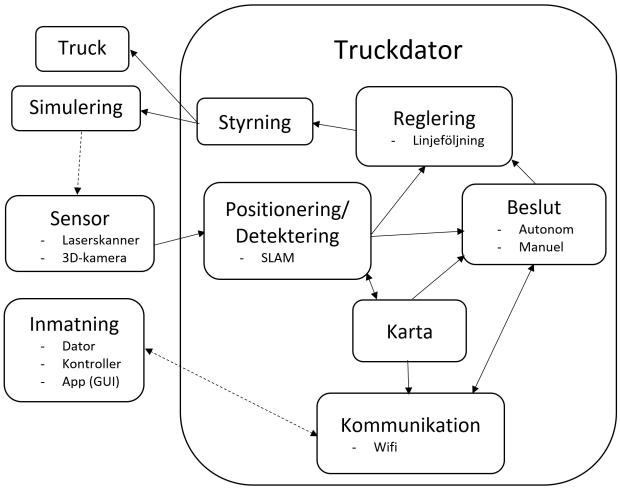
1.1 Definitioner

Nedan definieras vissa kärntermer som är återkommande i dokumentet.

ROS	Robot Operating System, en Open Source programvara som möjliggör och underlättar kommunikation mellan komponenter och moduler vid robotutveckling.
Nod	En process i ROS systemet som självständigt utför beräkningar. Mjukvaran är uppbyggd av ett nät av noder som själva utför uppgifter och kommunicerar via topics, services och parametrar.
Modul	Konceptuell uppdelning av systemets mjukvara baserad på funktionalitet.
Topic	En bus med namn där data delas mellan noder. En nod kan dela data på en topic och/eller lyssna efter ny data på en topic.
Publisher	En funktionalitet som låter en nod dela data på en topic.
Subscriber	En funktionalitet som låter en nod lyssna efter och reagera på ny data på topic.
Service	Ett funktionsanrop mellan noder där en nod kör en funktion i en annan och väntar på svar. För att en service ska kunna användas måste en nod först erbjuda servicen.
IMU	Inertial measurement unit.
Gazebo	Simuleringsmiljö som används för att testa truckplattformens funktioner och programvara i en virtuell miljö.
MiniTruckAppen	Den Androidapplikationen som kan kommunicera med plattformen.

2 Översikt av systemet

Produkten är en mjukvara för autonom körning av MiniReach, samt motsvarande simuleringsmiljö, båda med tillhörande dokumentation. Systemet ska utifrån extern inmatning autonomt kunna utföra viss enklare lagerhantering. Viss funktionalitet finns redan implementerat som kommer att användas som en bas, även om denna i vissa fall kan behöva omarbetas. En översikt av det planerade systemet kan ses i figur 1.



Figur 1: Översikt av systemet.(tillfällig)

Mjukvaran i truckdatorn är uppdelad i olika moduler. Modulerna kan ses som en konceptuell uppdelning av systemets mjukvara baserad på funktionalitet. Nedan listas de olika modulerna.

Beslutsmodul	Huvudprogram som tar in data från de andra modulerna och fattar beslut utifrån detta.
Regleringsmodul	Har i syfte att reglera rörelser enligt beslutsmodulens rutt-/rörelseplanering.
Styrningsmodul	Styr produktens aktuatorer, innehåller regulatorer som reglerar produktens rörelser.
Positioneringsmodul	Modul som håller koll på produktens positionering i rummet samt position relativt andra objekt. Behandlar signaler från linjesensorer samt eventuellt andra sensorer som kan komma att testas under projektets gång. Exempelvis 3D-kameran och en IMU, Inertial Measurement Unit.
Detektionsmodul	Används för att detektera och identifiera laster. Används även för att detektera eventuella hinder.
Kartläggningsmodul	Kartlägger utifrån sensordata området som produkten befinner sig i. Ger användare möjlighet att kartlägga ett nytt, okänt område.
Sensormodul	Tar in och behandlar sensordata.
Kommunikationsmodul	Agerar som interface mellan övriga moduler och MiniTruckAppen.

3 Kommunikation mellan moduler

Kommunikationen mellan moduler sker -topics -publishers - subscribers -... -...

4 Moduler

4.1 Beslutsmodul

Beslutsmodulen kommer vara den övergripande modulen vid användning av trucken i autonomt läge. Det kommer också vara den modulen som behandlar informationen som skickas från MiniTruckAppen och utför dessa instruktioner. Den kommer även skicka information tillbaka till applikationen med information om robotens nuvarande uppgift/status. Den största delen kommer bestå i att utföra uppdrag autonomt.

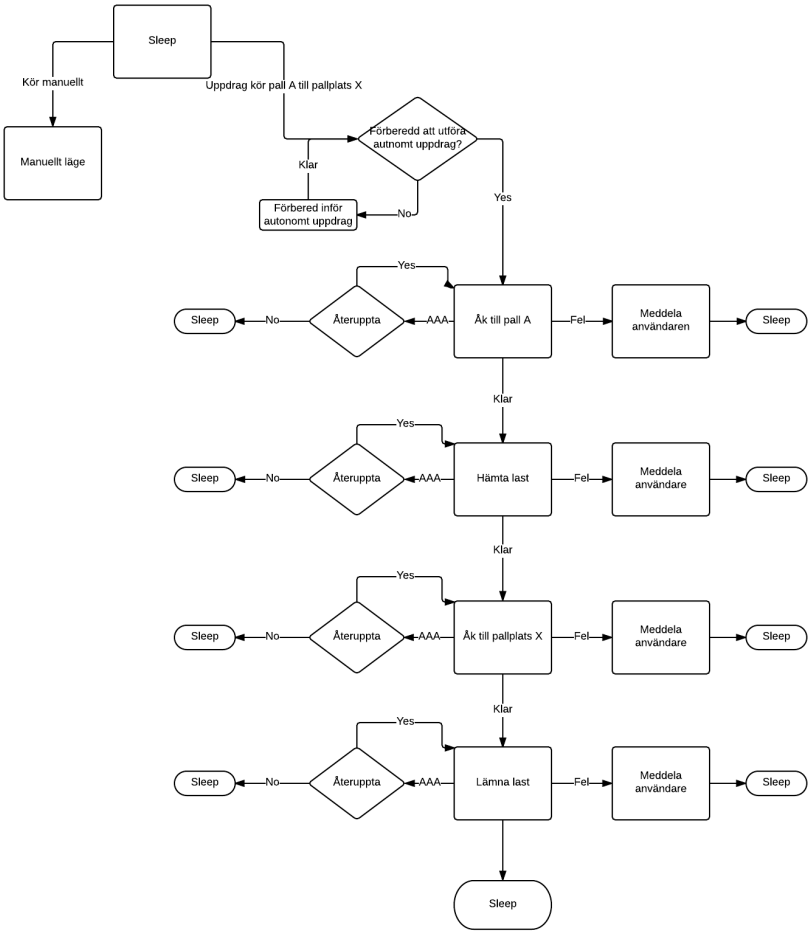
Beslutsmodulen kommer använda sig av ett ROS-paket som heter SMACH. Detta paket gör det enkelt att hantera mer komplexa uppgifter och dela in dessa i olika tillstånd. SMACH har också stöd för att skicka action-handlingar på ett smidigt sätt.

Action-handlingar används för att starta en funktion. Fördelen med action-handlingar är att man kan skicka med ett målvärde. Till exempel om man ska flytta roboten kan man gör en action-handling som flyttar roboten till ett par givna koordinater. Från huvudfunktionen anropar man sedan en action-client tillsammans med målvärdet, i detta fallet position dit man vill. Action-handling är kopplad till en action-server som utför uppdraget. Action-clienten skickar målvärdet till action-servern som sätter igång action-handlingen. En annan fördel är att det under hela förloppet går att få feedback från action-handling, till exempel med nuvarande position. När handlingen är klar returnerar action-servern ett resultat som skickas till action-clienten. Detta värde kan sedan användas av huvudfunktionen.

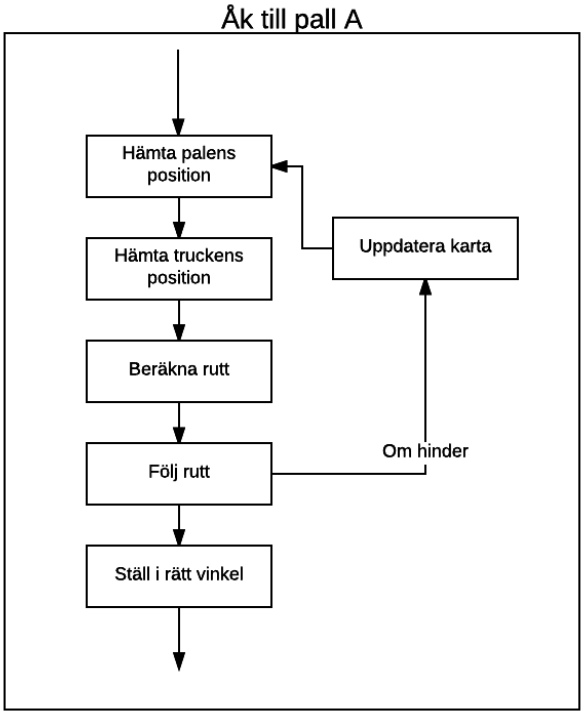
4.1.1 Flytta pall autonomt

I figur 2 ses ett flödesdiagram för utförande av uppgift att flytta en förvald låda till en förvald pallplats. Dessa tillstånd kommer bestå av undersystem som kan ses i figur 3 till 6. Varje tillstånd kommer innehålla action-handlingar för att utföra de specifika handlingarna. Fördelen med tillstånd är att det blir lättare att bygga på om man vill ha mer felhantering. Man kan lägga till olika tillstånd för att klara av fel som kan uppkomma. Till att börja med behandlas nästan alla fel av att användare meddelas att något har gått snett och sedan avbryts uppdraget. I mån av tid kan funktion läggas till för återuppta uppdrag som förhindrats.

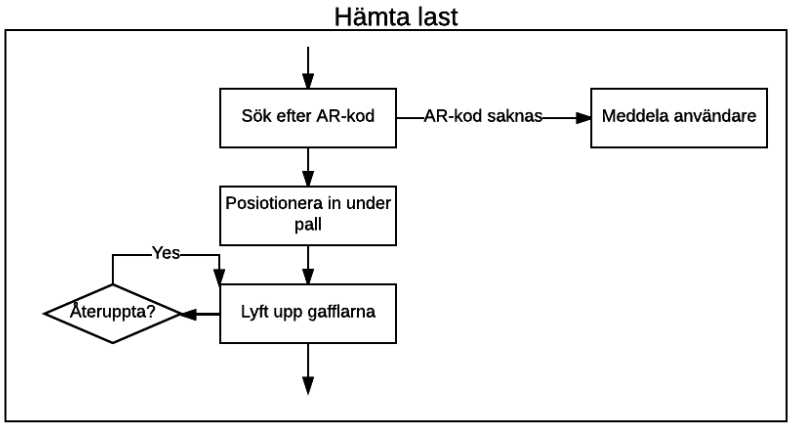
Som flödesdiagramet i figur 2 visar kommer beslutsmodulen ha ett initialt vilotillstånd Sleep. Därifrån kan användaren välja om hen vill starta ett autonomt uppdrag eller köra trucken manuellt. Innan ett autonomt uppdrag startar ska beslutsmodulen säkerställa att trucken är förberedd. Trucken anses vara förberedd om det inte har någon last på gafflarna eller står med gafflarna under en pall. Om pallen inte är förberedd för autonomt läge informeras användaren om detta och uppmanas åtgärda problemen. Som en funktionalitet som senare kan implementeras kan även detta skötas autonomt innan uppdrag startas.



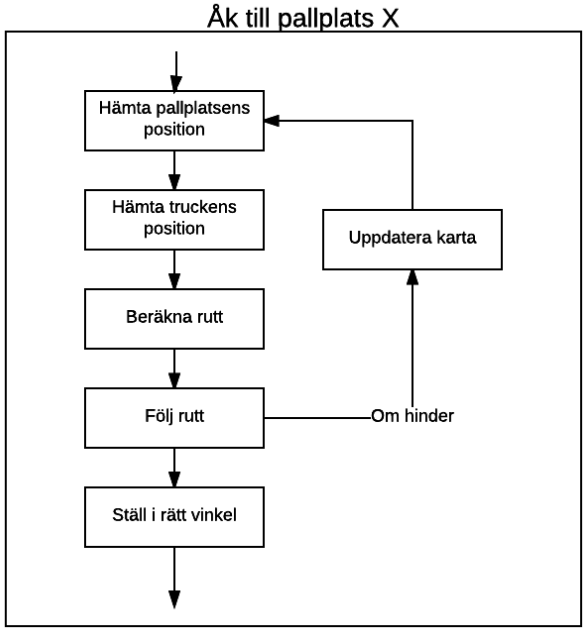
Figur 2: Visar översikt av beslutsmodulen, AAA - Avbrutan av användare.



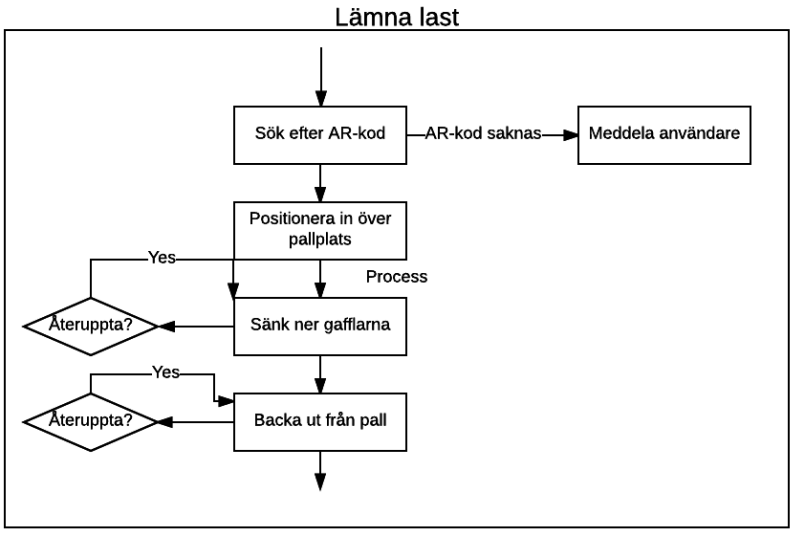
Figur 3: Undersystem till figur 2 - Åk till pall A, AAA - Avbrutan av användare.



Figur 4: Undersystem till figur 2 - Lämna last, AAA - Avbrutan av användare.



Figur 5: Undersystem till figur 2 - Åk till pallplats X, AAA - Avbrutan av användare.



Figur 6: Undersystem till figur 2 - Lämna last, AAA - Avbrutan av användare.

också gör systemet robustare.

I kommande underrubriker kommer varje delsystem mer noggrant analyseras.

Åk till pall A

För att åka till en given pall krävs följande (se figur 3):

- Hämta pallens position:

  Beslutmodulen hämtar in pallens position från kartmodulen. Detta görs vi en service, där koden för den specifika pallen skicka till kartmodul som svara med positionen samt rikting på pallen.

  - Request: map/pallet_position

- Hämta truckens position:

  Hämta in truckens position:

  - Truckens position på kartan beräknas med hjälp av tf och /base_footprint (truckens koordinatsystem projicerad på marken) och /map (kartans koordinatsystem)

- Beräkna rutt:

  Sker genom att använda en funktion som beräknar en rutt från truckens position till pallens position. Här krävs att en punkt framför pallens AR-kod bestäms och sätts som mål samt att trucken ställer sig i rätt vinkel i förhållande till pallen. Detta kommer att ske i en separat funktion i beslutsmodulen.

- Följ rutt:

  Skicka en action till regleringsmodulen att följa den beräknade rutten. Detta görs vi action-handling. Eventuellt kommer allt från att hämta trucken position, beräkna rutt och följa rutt att göras av reglermodulen. Detta för att det redan finns färdig paket för detta. Det som kan användas är ”move_base”.

- Ställ i rätt vinkel:

  Slutligen, skicka action till reglermodulen att vrida upp trucken i rätt vinkel. Detta görs troligen redan i förra steget.

Hämta last

För att positionera in sig under en given pall och lyfta upp lasten (se figur 4):

- Sök efter AR-kod:

  Skicka action till reglermodulen att snurra tills AR-koden upptäcks, detta sker via en service. Läs av från detekteringsmodulen när AR-koden är synlig, detta sker genom att läsa av rätt topic. Sluta avsökning när rätt AR-koden upptäcks.

  Kolla vilka AR-koder som syns:

  - Subscribe: ar_pose_marker

  Skicka till reglermodulen via service:

  - Request: move/spin_around

- Positionera in under pall:

  Skicka en action till reglermodulen att reglera in trucken under pallen med given AR-kod. Detta görs via en /pose_under_pallet action-handling. Goal-värdet blir AR-koden på pallen som man vill positionera in under.

  - Goal: AR-kod på pall att positionera in under
  - Result: Tom

  - Feedback: Tom

- Lyft upp gafflarna: Skicka till reglermodulen att lyfta upp gafflarna till en given höjd. Detta görs via en action-handling: /move_fork
  - Goal: Höjd på gafflarna
  - Result: Tom
  - Feedback: Aktuell gaffelhöjd

Åk till pallplats X

Samma funktionalitet som i kapitel 4.1.1 (se figur 5):

Lämna last

För att positioner in sig över en pallplats krävs likanande funktioner som vid ”Hämta last” se kapitel 4.1.1 (se figur 6). Dock kan avbrottshanteringen skilja sig åt. Om ett uppdrag avbryts kommer trucken fortfarande ha en pall på pallgafflarna. Detta behandlas till och börja med att användaren för köra i manuellt läge och ställa ner pallen.

- Sök efter AR-kod:

  Skicka till reglermodulen att snurra tills AR-kod upptäcks, detta görs på samma sätt som i ”Hämta pall A”.

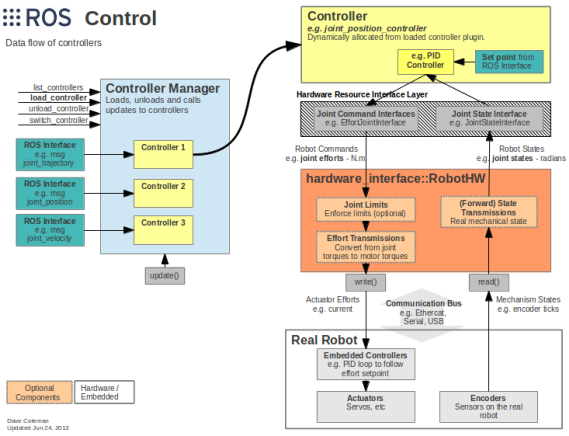
- Positionera in över pallplats:

  Skicka till reglermodulen att reglera in trucken över pallplatsen med given AR-kod. Detta görs med en action-handling, /pose_pallet_place:
  - Goal: AR-kod på pall att positionera in under
  - Result: Tom
  - Feedback: Tom

- Sänk ner gafflarna: Skicka till reglermodulen via topic. Detta görs på samma sätt som vid höjning av pallgafflar.
- Back ut från pall: Skicka till reglermodul att backa ut från pall. Detta görs vi en service:
  - Request: move_reverse

4.2 Regleringsmodul

Reglering sker med hjälp av ros_control som är ett ramverk för reglering i ROS. ros_control innehåller en samling av paket. I figur 7 visas en översiktlig bild över ros_control.



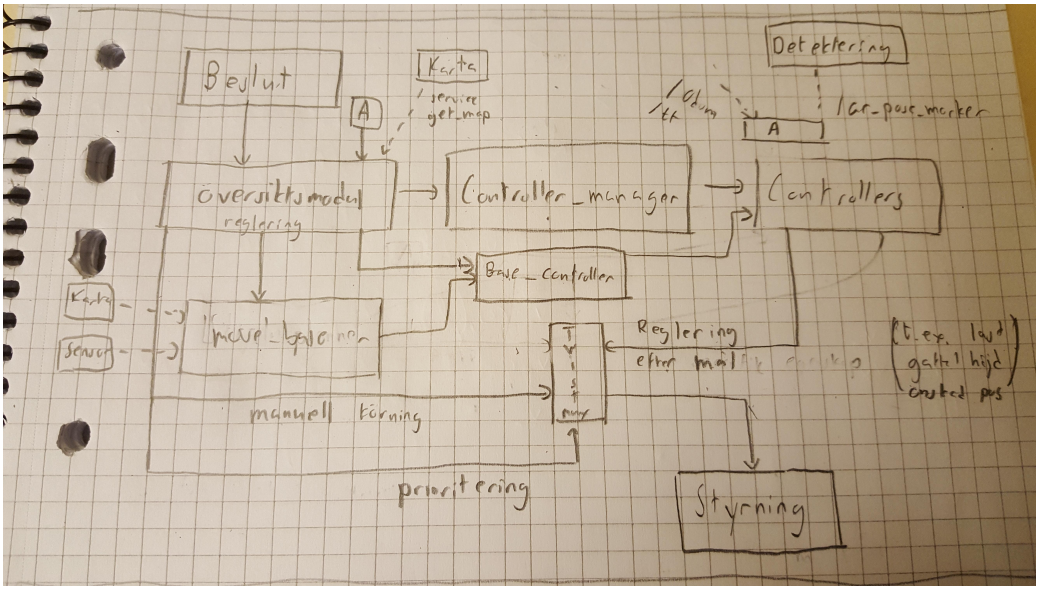
Figur 7: Översiktlig bild av ros_control.

Några av de ingående paketen i ros_control listas nedan:

- **control_toolbox** - Innehåller verktyg som bland annat underlättar vid utveckling av regulatorer.
- **controller_interface** Innehåller basklassen för regulatorer.
- **controller_manager** - Möjliggör att specifika controllers kan laddas in, laddas ut samt stoppas och startas. Detta ger en övergripande kontroll på vilka controllers som ska vara aktiva alternativt avaktiverade.
- **controller_manager_msgs** - Definierar meddelanden och tjänster för controller_manager.
- **hardware_interface** - Innehåller basklassen hardware_interface.
- **joint_limits_interface** - Ett interface som används för att hantera styrsignalsöverskridningar.
- **realtime_tools** - En mängd verktyg som tillåter publicering på en ROS-topic från en realtids-tråd.
- **transmission_interface** - Innehåller metoder för att konvertera intensitet, position och hastighet mellan aktuator och joint states. Det innehåller även datastrukturer för att kunna representera mekaniska transmissioner.

Som indata till controllers skickas data från encoders (joint state data) samt ett börvärde. Controllerna är ofta PID-regulatorer som skickar en styrsignal till aktuatorer. Controllerna i ROS kommunicerar inte direkt med hårdvaran utan meddelanden skickas till *hardware_interface* skapad av *controller_manager*, vilken är en hård realtids kompatibel loop för att kunna reglera en robot. *hardware_interface* innehåller *effort_transmissions* och *Joint_limits*. *effort_transmissions* sköter omvandling av styrsignal i mjukvara till styrsignal i fysisk miljö. *Joint_limits* behandlar fall där styrsignaler överskrider gränsvärden.

Paketet *controller_manager* är ett paket som möjliggör att specifika controllers kan laddas in, laddas ut samt startas och stoppas. Detta medför en möjlighet till en övergripande kontroll på vilka controllers som ska vara aktiva alternativt stoppas. Kommandot spawner



Figur 8: Översikt över reglermodul

ger möjlighet att ladda in och starta flera controllers i ett steg. Kommandot unspawner stoppar flera controllers i ett steg, men de förblir inladdade och därmed snabbt tillgängliga igen vid behov. Switch-kommandot möjliggör att en lista med controllers stoppas och att en annan lista med controllers startas, ett kommando som kan vara lämpligt att använda ifall beslutsmodulen meddelar att förutsättningarna ändrats kraftigt. Samtliga av dessa kommandon körs via ett call-anrop till en service i paketet. Se *controller_manager* i nedanstående schemaläggande bild.

I nuläget regleras Minireach med hjälp av noden *base_controller* som från referenssignaler innehållande hastigheter i x- och y-led och rotationshastighet runt z-axel samt styrhjulets tillstånd beräknar en motsvarande position och hastighet på styrhjulet för att uppfylla detta. *base_controller* subscribar på *base_controller/command* (referenssignaler) och */minireach/joint_states* (styrhjulets tillstånd). Den publicerar på */minireach/wheel_drive_velocity_controller/command* och */minireach/wheel_steer_position_controller/command*, vilka innehåller önskad position och hastighet på styrhjulet. Själva regleringen sker med PID-regulatorer i *joint_position_controller* och *joint_velocity_controller*. Dessa läser av den reglerade storheten från ett interface för aktuell storhet och jämför dessa med ett önskat värde, se setpoint i figur 7. Skillnaden mellan dessa blir reglerfelet vilket används för att bestämma mängden effort som ska skickas in i motsvarande Command_Interface. Detta görs genom att köra kommandot setCommand på en handle till motsvarande interface.

4.3 Styrningsmodul

Styrningsmodulen tar emot körinstruktioner från regleringsmodulen och omvandlar dessa till en signal som leder till önskvärt körbeteende hos minireach alternativt i simulationen. Denna modul tar också hänsyn till begränsningar av styrsignaler. Till exempel kan styrhjulet på trucken bara roteras inom ett intervall från -180 till +180 grader. Införandet av topphastighet kan också implementeras här. Begränsingar av akuatorsignaler sköts av *Joint_Limit_Interface* som har möjlighet att skriva över instruktionen som

skickats från regleringsmodulen ifall denna bryter mot en begränsning. Efter att denna funktionalitet genomförts så publiceras kommandot på ett topic, exempelvis */minireach /wheel_steer_position_controller/command*, vilket gör att det förs vidare för aktuation.

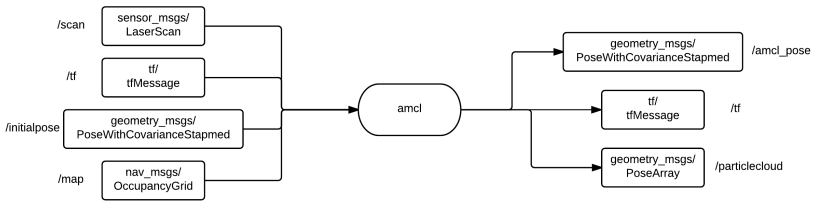
4.4 Positioneringsmodul

4.4.1 Global positionering

Efter det att kartan är ritad kommer trucken vilja placera ut sig själv i denna nu kända omgivning. Detta görs med hjälp av noden *amcl*.

Noden använder sig av en lokaliseringsmetod av Monte Carlo typ som använder ett partikelfilter för att probabilistiskt bestämma truckens position i sin omgivning.

amcl subscribar på data från laserskannern, medelvärden och kovarianser som används till att (om-)initialisera partikelfiltret och även kartan som ritats upp. Noden publicerar topics som bland annat truckens estimerade position, med kovarians, i rummet och uppsättningen av alla estimeringar som partikelfiltret gjort.



Figur 9: amcl med de aktuella topics och dess meddelandetyper.

Följande topics ska användas för den globala positioneringen:

- **/scan** är ett *sensor_msgs/LaserScan* meddelande som innehåller data från laserscannrar. Det har, i normalfall, en uppdateringsintervall på 10 Hz.
- **/tf** är ett *tf/Message* meddelande som ger robotens tillstånd. Är något som noden *amcl* både subscribar och publicerar till.
- **/initialpose** är ett *geometry_msgs/PoseWithCovarianceStamped* meddelande som ger truckens ursprungsposition i rummet.
- **/map** är ett *nav_msgs/OccupancyGrid* meddelande som ger en beskrivande karta av rummet.
- **/amcl_pose** är ett *geometry_msgs/PoseWithCovarianceStamped* meddelande från amcl noden som beskriver truckens position i rummet.

4.4.2 Lokal positionering

Den lokala positioneringen avser positionering av trucken i förhållande till pall och/eller pallplats. Detta kommer att ske genom att man utnyttjar de AR-koder som pallar och pallplatser är utrustade med. Till detta utnyttjas noden *ar_track_alvar* som detekterar, identifierar och ger AR-koders position i rummet. Se rubrik 4.5 Detektering för mer information.

4.5 Detekteringsmodul

Detekteringsmodulen nyttjar datan från sensorer för att ta fram positioner på AR-koder respektive hinder. Modulen består av två separata noder **ar_track_alvar** och **/mini-reach/collision_detection** som hanterar detektion av AR-koder respektive hinder.

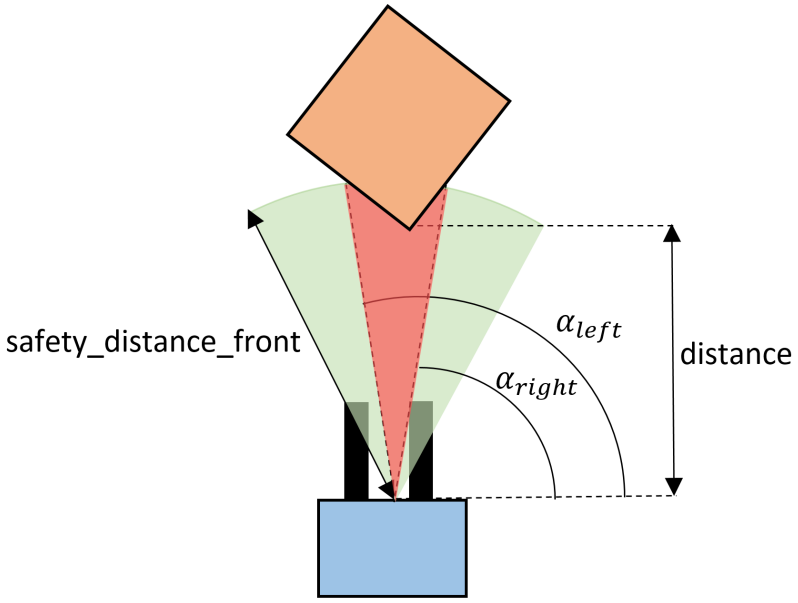
4.5.1 AR-koder

För att identifiera pallar och pallplatser används AR-koder utplacerade på dessa objekt. För att detektera dessa krävs kontinuerlig avsökning av bilderna från truckens 3D-kamera. Detta sköts av den existerande noden **ar_track_alvar** som ger ut information om upptäckta AR-koder på topicen **ar_pose_marker**. Den information som inkluderas där är AR-kodens position i rummet, dess orientation samt dess ID.

(Eventuellt att hinderdetektion i detekteringsmodulen inte behövs då detta redan ingår i kartläggning, detta skulle bara agera extra säkerhetsåtgärd.)

4.5.2 Hinder

Detektering av hinder sker genom kontinuerlig avsökning av avståndsvärden från laserscannrar. Det intervall av avståndmätningarna som är i körriktning avsöks efter avstånd mindre än en kostant *safety_distance_front* och dessa flaggas. Om ett delintervall innehåller flaggade mätningar identifieras ett hinder ligga i det intervallet på med avstånd motsvarande det minsta av mätningarna.



Figur 10: Skiss av kollisionsdetektion vid körning framåt.

Alla för tillfället upptäckta hinder delas på en topic där de beskrivs med α_{left}, α_{right} samt *distance* se Figure 10. Eftersom laserscannern fram ej är centrerad kommer mätvärdena behöva transformeras innan de behandlas. Vid behov kommer mätningarna även filtreras

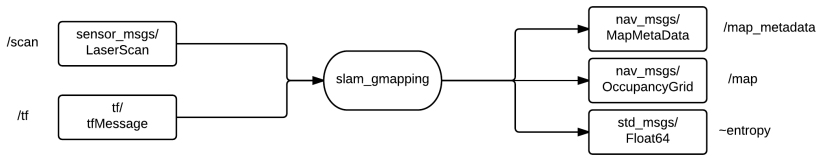
innan objekt annonseras för att undvika falska detekteringar som följd av tillfälligt felaktiga mätvärden från sensorn. Samma princip görs även i andra körriktningar då trucken svänger eller backar. Om noden ej kan hitta mätvärden i den aktiva körriktningen meddelas detta på samma topic som upptäckta hinder. Den nod som hanterar kollisions detektion är

/minireach/collision_detection: Hämtar in avstånd sensor_msgs/LaserScan meddelanden, transformerar till truckens bas och avgör om det finns hinder som blockerar truckens aktiva körriktning. Information om dessa hinder delas på topic.
Subscribar på: /scan, /odom
Publiserar på: /minireach/obstacle

4.6 Kartläggningsmodul

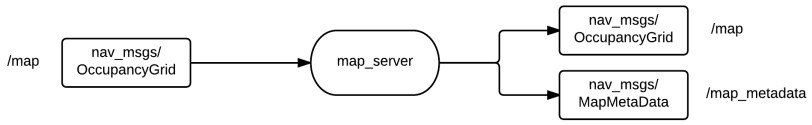
För att kunna skapa en uppfattning om omgivningen kan sensorsignalerna användas för att bygga upp en karta. Kartan byggs upp med hjälp av noden *slam_gmapping* och hanteras sedan av noden *map_server*.

slam_gmapping tar in data från laserskannern och truckens tillstånd och publicerar data på bland annat topicen **/map** genom meddelandet *nav_msgs/OccupancyGrid*. Det ger en beskrivning av omgivningen genom att dela in området i pixlar och ge varje pixel ett värde mellan [0, 100] beroende på hur sannolikt det är att det står ett hinder i pixeln. Ju högre värde desto större sannolikhet. Pixlar som ej är avsökta har värdet -1.



Figur 11: slam_gmapping med de aktuella topicsen och dess meddelandetyper.

map_server subscribar på topicet **/map** som ger meddelandetypen *nav_msgs/OccupancyGrid*. Den lagrar, sparar och laddar kartor.



Figur 12: Map_server med de aktuella topicsen och dess meddelandetyper.

Kartläggningsmodulen kommer även placera ut och spara positionen för eventuella hinder på kartan som detekteringsmodulen detekterar, tills dessa flyttar på sig. Den ska även innehålla en databas som sparar detekterade AR-koder samt dess position i rummet. Till detta används noden *ar_track_alvar* och topicen **ar_pose_marker** som ger taggens position i rummet. I topicen **visualization_marker** skickas meddelandet *geometry_msgs/Point* som beskriver en positionen i rummet.

Kartläggningsmodulen erbjuder servicen */GetMap* som andra noder intresserade av att få uppdaterade kartor kan kalla på.

4.7 Sensormodul

Sensorer inkluderade i robotmodellen till ROS publiserar direkt mätvärden med tillhörande sensorinformation på topics. Trucken är utrustad med två laserskannrar (en fram och en bak), en 3D-kamera monterad framåt, en inertial measurement unit (IMU) samt en gaffelhöjdsensor.

4.7.1 Laserskanner

Truckens laserskanners är av modell Hokuyo URG-04LX-UG01. Skannern mäter avstånd till närmaste hinder längs en horisonell linje. Avstånden tillsammans med sensorns position publiseras på topicen *scan* med en frekvens på 10 Hz. Då trucken har två av dessa sensorer publiseras mätningar på *scan* med en frekvens av 20 Hz och mätvärdena separeras med tillhörande publiserade sensorpositioner.

Modellinformation:

- **Modell:** Hokuyo URG-04LX-UG01
- **Räckvidd:** 5.6 meter
- **Synfält:** 240°
- **Noggrannhet:** ± 30mm för avstånd kortare än 1000 mm, för längre avstånd är noggrannheten ±3% av mätningen
- **Uppdateringsfrekvens:** 10 Hz
- **Optisk upplösning:** 0.36°

4.7.2 3D-kamera

I nuläget kan Minireach utrustas med två olika 3D-kameror. Den ena är en ZED stereokamera och den andra kameran är en Orbecc Astra. Båda kamerorna ger ifrån sig liknande information och publicerar flertalet topics. Nedan listas en teknisk specifikation för Orbecc Astra.

- **Modell:** Astra
- **Räckvidd:** 0.4-8 meter
- **Synfält:** 60° horisontellt, 49.5° vertikalt
- **Upplösning på djup-bild:** 640x480 px (VGA)
- **Upplösning på RGB-bild:** 1280x960 px

Nedan listas några av de topics som Orbecc Astra publicerar.

- **/camera/depth_registered/points** är ett *sensor_ msgs/PointCloud2*-meddelande vilken innehåller 3D- och färgdata. Det publiceras med VGA-upplösning (640x480) vid 30 Hz.
- **/camera/depth_downsampled/points** är ett *sensor_ msgs/PointCloud2*-meddelande som bara innehåller 3D-data. Det publiceras med QQVGA-upplösning (160x120) vid 30Hz och används främst för undvikande av hinder.

- **/camera/depth/image_raw** är ett *sensor_msgs/Image*-meddelande. Detta innehåller en 16-bitars djup-bild i 2D. Det publiceras vid 30 Hz med VGA-upplösning.
- **/camera/depth/image** är ett *sensor_msgs/Image*-meddelande. Detta är en float djup-bild i 2D. Det publiceras vid 30 Hz med VGA-upplösning.
- **/camera/rgb/image_raw** är ett *sensor_msgs/Image*-meddelande. Detta meddelande innehåller färgdata i 2D. Det publiceras vid 10 Hz med VGA-upplösning.

Nedan följer en teknisk specifikation över ZED stereo-kameran. Kameran har olika videolägen vilka ger olika antal bilder per sekund och upplösning. De olika lägena är *2.2K*, *1080p*, *720p*, och *WVGA*.

- **Modell:** ZED stereo
- **Räckvidd:** 0.7-20 m
- **Synfält:** 110°diagonalt
- **Bilder per sekund:** *2.2K*: 15 fps *1080p*: 30 fps *720p*: 60 fps *WVGA*: 100 fps
- **Upplösning:** *2.2K*: 4416x1242 *1080p*: 3840x1080 *720p*: 2560x720 *WVGA*: 1344x376

Nedan listas de topics som ZED stereo-kameran publicerar.

- **/camera/point_cloud/cloud** Här publiceras ett 3D-strukturerat punktmoln från kameran.
- **/camera/depth/camera_info** är ett *sensor_msgs/CameraInfo*-meddelande. Här publiceras metadata och kamerakalibrering relaterat till djupseende.
- **/camera/depth/image_rect_color** innehåller en korrigerad djup-bild från kameran.
- **/camera/left/camera_info** innehåller information om den vänstra kameran.
- **/camera/left/image_rect_color** innehåller en korrigerad bild från den vänstra kameran.
- **/camera/rgb/camera_info** är ett *sensor_msgs/CameraInfo*-meddelande. Här publiceras metadata och kamerakalibrering relaterat till RGB.
- **/camera/rgb/image_rect_color** innehåller en korrigerad RGB-bild från kameran.
- **/camera/odom** innehåller information om odometrin som ges av ZED trackingen.

4.7.3 3D-kamera styrning

Det är möjligt att vinkla 3D-kameran vertikalt genom att publisera kommandon på /minireach/camera_tilt_controller/command. Den kan riktas in på ett intervall av 90°från att kameran är ställd rakt framåt till rakt ner i marken.

Kameran kommer att ha ett standard läge inriktat för att se AR-koder på avstånd. Vid pallyft kan det dock behövas att kameran riktas in lägre för att kunna se AR-koden under en större del av lyftet. En nod **ar_track_control** som tar in 3D-kamera läget och /ar_track_pose och ger ut /minireach/camera_tilt_controller/command används för att

få kameran att följa AR-koden. Kamerans position jämförs med AR-koden för att bestämma vilken vinkel kameran bör vara i för att centrera sin bild på AR-koden. Noden är en action server som tar in kommando om ID på AR-koden den ska följa där uppgiften avbryts vid kommando från action klienten eller då koden ej längre kan spåras. Noden återställer kameran till standardläge efter uppgiften.

ar_track_control: Centrerar truckens 3D-kamera vertikalt mot känd AR-kod.
Subscribar: /ar_track_pose, (kamera_position)
Publiserar: /minireach/camera_tilt_controller/command

4.7.4 IMU

En IMU av modell MPU-9250 installeras för att ge bättre uppdatering vid truckens rörelsemodell. Den publiserar ett *sensor_msgs/Imu*-meddelande på topicen *imu* som innehåller information om linjära accelerationer och rotationshastigheter.

odometry: Uppskattar och publiserar truckens position baserat på rörelsemodell och värden från IMU. Uppdaterar även /tf som används i amcl.
Subscribar: /minireach/joint_states, /imu
Publiserar: /odom, /tf

4.8 Kommunikationsmodul

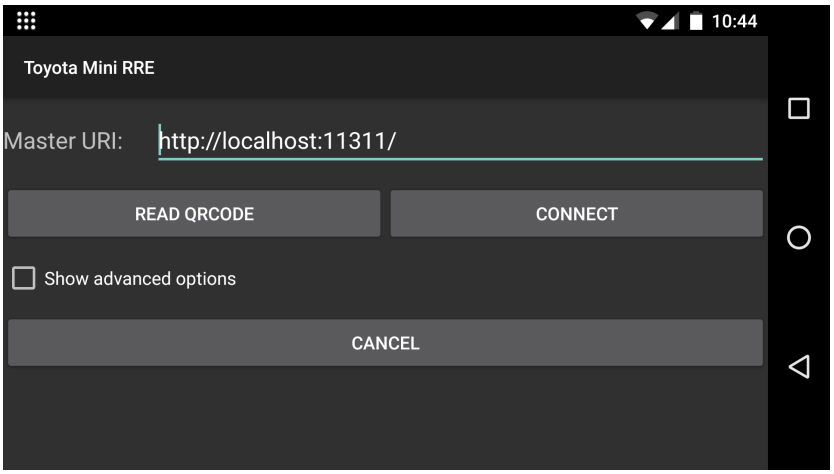
Kommunikationsenheten har i uppgift att agera interface mellan trucken och externa wifi uppkopplade enheter. Detta hanteras dock redan av ROS på det sättet att trucken lyssnar på uppkopplingar på PORT 11311. De externa enheterna kan koppla upp mot denna och själva agera noder i ROS nätverket utan att man behöver ta hänsyn till vilken enhet noden körs på. Detta gör att kommunikationsenheten ej har någon funktionalitet och kommer ej implementeras i nuläget.

4.9 MiniTruckApp

Den till trucken tillhörande kontrollplattformen MiniTruckApp kommer att vara en applikation körbar på Android-plattformer av version 4.0.3 (API 15) eller senare. Koden till applikationen skrivs i Java med paket från Android SDK samt Android ROS. Applikationen inriktar sig och testas primärt på plattformar med större skärm (≥ 8") och använder engelska som operativt språk. I mån av tid och vid behov kommer även stöd för mindre skärmar och mer språkstöd läggas till.

Detta projekt kommer att bygga vidare på en existerande applikation utvecklad av Toyota vid namn ”Toyota Mini RRE”. Denna har vid start redan en stor del av den sökta funktionaliteten. En beskrivning av applikationens design utifrån dess grafiska upplägg ges nedan.

4.9.1 Uppkoppling till truck



Figur 13: GUI vid initial körning och uppkoppling mot truck.

Det första som krävs då applikationen körs är att upprätta en uppkoppling till trucken som är uppkopplad mot internet eller ett lokalt nätverk. Trucken antas ha en känd IP-adress samt lyssna på uppkopplingar på port 11311.

Applikationen öppnar upp ett enkelt fönster för inmatning av IP-adress och port som kan ses i Figure 13. Detta kommer från klassen MasterChooser som är en del av Android ROS paketet. Denna körs och hanteras från RosActivity som är en förlängning av klassen Activity från Android SDK. RosActivity med ett antal andra klasser från Android ROS ger all funktionalitet som krävs för att hantera uppkopplingen mot trucken. Ett urval av relevanta Android ROS klasser som agerar inteface mot trucken är

- **Publisher:** Ger möjlighet att publicera på en topic i trucken.
- **Subscriber:** Ger möjlighet att lyssna på en topic i trucken efter meddelande av viss typ.
- **ServiceServer:** Ger möjlighet att skicka iväg en service till en nod.
- **ServiceClient:** Ger möjlighet att ta emot en service från en nod.

Det finns även möjlighet att skapa ROS noder i Android applikationen genom att utöka klassen NodeMain från ROS Android paketet. Dessa beter sig som övriga noder i trucken och kommer sett från andra noders perspektiv vara en nod i trucken som alla andra utan att behöva ta hänsyn till att noden körs på annan plattform.

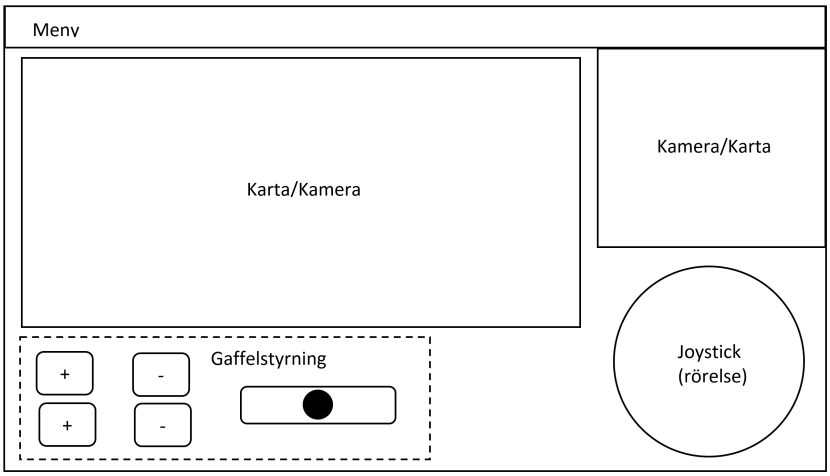
4.9.2 Noder

De noder som kommer att användas med beskrivningar av dess funktion, publiseringar och subscriptions av topics, samt service användning ges nedan.

- **android/camera_view:** Hämtar in och visar kamera bild i GUI.
  - **Publiserar på:** -
  - **Subscribar på:** camera/rgb/image_raw/compressed

- **android/virtual_joystick:** Hanterar joysticken i appens GUI vid manuellt läge.
  - **Publiserar på:** nav_vel
  - **Subscribar på:** odom
- **android/map_view:** Hämtar in och visar karta samt pallar/pallplatser.
  - **Publiserar på:** -
  - **Subscribar på:** /map, /move_base/TrajectoryPlannerROS/global_plan, /scan , (pallinfo), (pallplatsinfo)
- **android/fork_controller:** Skickar instruktioner om gaffelposition till trucken vid manuellt läge.
  - **Publiserar på:** minireach/fork_position_controller/command, minireach/reach_position_controller/command
  - **Subscribar på:** -
- **android/instruction_publisher:** Skickar ut instruktioner till truckens beslutmodul samt tar emot status meddelande från samma modul vid autonom körning.
  - **Publiserar på:** (service instruction)
  - **Subscribar på:** (beslut_status_msgs)

4.9.3 Manuell körning



Figur 14: GUI vid manuell körning.

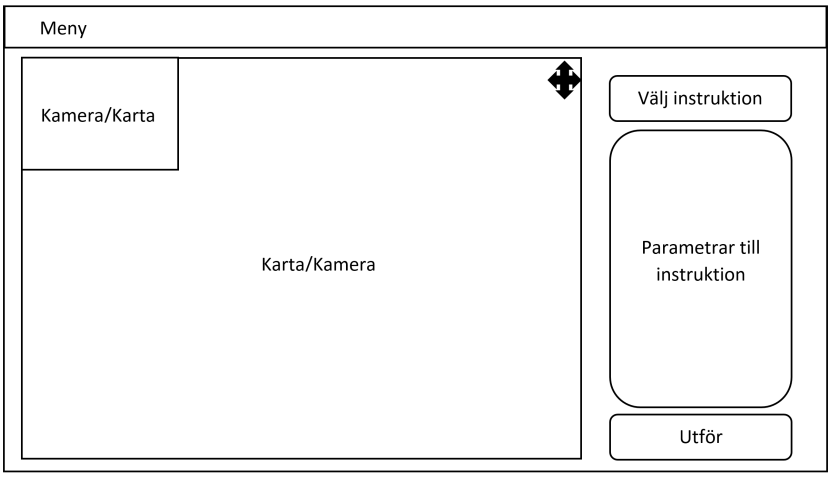
Vid manuell körning ser användaren det GUI som ses i Figure 14. Övre meny fält ger möjlighet att gå tillbaka och ändra uppkopplingsinställningar, uppdatera kartbild, byta till autonomt läge samt visar batteristatus för truck.

Karta/Kamera hanteras av noderna map_view och camera_view respektive och de visar lagerkarta och bild från kamera på trucken. Vid klick på den mindre av bilderna växlar karta och kamera plats. En toggleknapp låter användaren växla mellan att kartan roterar efter truckens position eller är i fast läge. Kartan visas alla kända paller och pallplatser utmarkerade och identifierade med användarvänlig benämning, exempelvis A, B, C,...

Joystick hanteras av virtual_joystick noden och tar inmatning från användaren för rörelse av truck. Denna har möjlighet att vrida trucken på sin position vid inmatning rakt åt vänster eller höger.

Gaffelstyrning hanteras med undantag för det rent grafiska av noden fork_controller. Gafflarna höjs, sänks, skjuts in eller ut stegvis med knapparna +/- för gaffelhöjd och inskjutning respektive. Steglängden för dessa ändras via en slider som visar steglängd i cm. Den aktuella höjden/inskjutningen visas även vid +/- knapparna.

4.9.4 Autonom körning



Figur 15: GUI vid autonom körning.

Vid autonom körning ser användaren det GUI som ses i Figure 15. Övre meny har samma funktionalitet och utseende som vid manuell körning med undantag för möjlighet att byta till manuellt läge istället för autonom.

Karta/Kamera fungerar på samma sätt som vid manuell körning med annorlunda positionering samt möjlighet att dölja den mindre bilden med fullskärms ikon.

Högra del av GUI ger användaren möjlighet att välja instruktion med tillhörande parametrar för att sedan skicka dessa till trucken då ”Utför” trycks. Publiseringen av instruktionerna hanteras av noden instruction_publisher medan inmatningslogik hanteras separat. De instuktioner som finns och vilka övriga parametrar dessa har listas nedan.

4.9.5 Instruktioner

Move palett:
Den instruktion som ska användas för att flytta en pall från en plats till en pallplats. Första parametern som behövs är val av pall utifrån en lista av alla kända pallar identifierade med samma id som ges i karta. Andra parametern är val av pallplats att ställa pallen vid. Även denna väljes utifrån en lista av kända pallplatser.

Instruktionen går även att använda i steg genom att endast ange pall att hämta, utan att ange destination. Då skickas instruktion till trucken att hämta upp pallen för att sedan vänta på ytterligare instruktioner. Om trucken redan bär på en låda ger denna instruktion inte möjlighet att välja en ny låda att flytta utan endast pallplats att ställa ifrån sig den låda trucken redan har.

Go to:
Denna instuktion används för att få trucken att köra till önskad position i lagret. De parametrar som instuktionen kräver är koordinater till den önskade positonen. Detta kan matas in på flera sätt där det första är direkt inmatning av värden.

Det är även möjligt att med klick på position i karta överföra dessa koordinater till instruktionen. Till sist kan position även väljas utifrån lista av kända pallar och pallplatser. De koordinater som då skickas till instruktionen är position för på-början av pallyft.

Map area:
Instruktion för att påbörja en kartläggning av lagerlokalen. De parametrar som skickas med instruktionen är om trucken ska glömma existerande karta vid påbörjan av kartläggning eller ej.

4.9.6 Instruktionformat

Instruktion meddelandena som skickas till beslutmodulen formateras i ACII kod för enkel felsökning. Dessa skickas som service meddelanden med instruktionen som första argumentet och övriga pararmetrar som en array för övriga argument. Appen väntar då på ett status liknande svar från trucken.

5 Säkerhet

Säkerheten som kommer att prioriteras i detta projekt är att trucken inte ska köra in i några föremål. Hänsyn till om trucken ställer ner pallen på någon/något kommer inte tas. För att inte köra in i något kommer det finnas en nod som kollar på sensordatan för de två laserscanarna som finns. En riktad framåt och en riktad bakåt. Noden kommer också kolla på vilken riktning trucken kör i. Detta genom att kolla på på topic för vilket håll vi står åt och om vi åker framåt och bakåt. Noden kommer sedan skicka ut på en specifik topic om roboten har tillåtelse att köra eller inte. Det kommer också finnas in topic där man kan stänga av om säkerhetsfunktionen ska vara aktiverad eller inte. Den behöver bland annat kunna stängas av om man ska köra in till en pall eller liknande.

- Subscribe:
  - **/minireach/wheel_steerposition_controller/command**: Kollar åt vilket håll vi svänger.
  - **/minireach/wheel_drive_velocity_controller/command**: Kollar om vi kör framåt eller bakåt och med vilken hastighet.
  - **/scan**: Kollar på sensordata för laserscanner (hur blir det med den nya laserscannern?)
  - **/minireach/saftey/check_saftey**: Innehåller information om säkerhetsfunktion ska var på/av. Är en boolean: True - kollar efter hinder, False - kollar inte efter hinder.

- Publish:
  - **/minireach/safety/ok_to_drive**: Innehåller information om trucken håller på att köra på något. Är en boolean: True - OK att köra, False - inte OK att köra.

6 Sammanställning av topics

Topic	Subscribe/Publish	Modul
ar_pose_marker	Publish	Detektionsmodul
ar_pose_marker	Subscribe	Beslutsmodul
ar_pose_marker	Subscribe	Reglermodul
/amcl_pose	Publish	Positioneringsmodul
/initialpose	Subscribe	Positioneringsmodul
/particlecloud	Publish	Positioneringsmodul
/entropy	Publish	Kartläggningsmodul
/tf	Subscribe	Reglermodul
/tf	Subscribe	Kartläggningsmodul
/tf	Subscribe	Positioneringsmodul
/tf	Publish	Positioneringsmodul
/tf	Publish	Sensor
/map	Subscribe	Positioneringsmodul
/map	Subscribe	Kartläggningsmodul
/map	Publish	Kartläggningsmodul
/map_metadata	Publish	Kartläggningsmodul
/minireach/wheel_drive_velocity_controller/command	Publish	Styrningsmodul
/minireach/wheel_steerposition_controller/command	Publish	Styrningsmodul
/minireach/wheel_steerposition_controller/command	Subscribe	Säkerhet
/minireach/wheel_drive_velocity_controller/command	Subscribe	Säkerhet
/minireach/fork_position_controller/command	Publish	App
/minireach/reach_position_controller/command	Publish	App
/minireach/safety/check_safety	Subscribe	Säkerhet
/minireach/safety/ok_to_drive	Publish	Säkerhet
/minireach/joint_states	Subscribe	Regleringsmodul
/minireach/joint_states	Publish	Sensor
/base_controller/command	Subscribe	Regleringsmodul
/scan	Publish	Sensor
/scan	Subscribe	Säkerhet
/scan	Subscribe	App
/scan	Subscribe	Positioneringsmodul
/scan	Subscribe	Kartläggningsmodul
/camera/depth/points	Publish	Sensor
/camera/rgb/image_raw	Publish	Sensor
/camera/rgb/image_raw	Subscribe	Detektering
/camera/rgb/image_raw/compressed	Publish	Sensor
/camera/rgb/camera_info	Publish	Sensor
/camera/rgb/camera_info	Subscribe	Detektering
/camera/depth_registered/points	Publish	Sensor
/camera/depth_downsampled/points	Publish	Sensor
/camera/depth/image_raw	Publish	Sensor

/camera/depth/image	Publish	Sensor
/camera/point_cloud/cloud	Publish	Sensor
/camera/depth/camera_info	Publish	Sensor
/camera/depth/image_rect_color	Publish	Sensor
/camera/left/camera_info	Publish	Sensor
/camera/left/image_rect_color	Publish	Sensor
/camera/rgb/image_rect_color	Publish	Sensor
/camera/odom	Publish	Sensor
/imu	Publish	Sensor
/imu	Subscribe	Sensor
/odom	Publish	Sensor
/odom	Subscribe	App
/nav_vel	Publish	App
/move_base_simple/goal	Publish	App
/move_base/goal	Publish	App

7 Sammanställning av services

Service	Request/Reply	Modul
map/pallet_position	Request	Beslutsmodul
map/pallet_position	Reply	Kartläggningsmodul
move/reverse	Request	Beslutsmodul
move/reverse	Reply	Reglermodul
move/spin_around	Request	Beslutsmodul
move/spin_around	Reply	Reglermodul
nav_msgs/GetMap	Reply	Kartläggningsmodulen
nav_msgs/GetMap	Request	Reglermodul Positioneringsmodulen GUI
instruction	Request	App
instruction	Reply	Beslutsmodul
std_srvs/Empty	Reply	Positioneringsmodulen

8 Sammanställning av actions

Action	Modul	Goal	Result	Feedback
move_base	Beslutsmodul	Position	-	Aktuell position
pos_under_pallet	Beslutsmodul	AR-kod	-	-
move_fork	Beslutsmodul	Önskad gaffelhöjd	-	Aktuell gaffelhöjd
move_reach	Beslutsmodul	Önskad gaffelinskjutning	-	Aktuell gaffelinskjutning
pos_pallet_place	Beslutsmodul	AR-kod	-	-