Designspecifikation

Hampus Westerberg
Joel Wiklund
Tomasz Mazurek
Carl Liljeberg
Mohammad Rajabi
Anton Lund
Simon Svahn

12 december 2024

Version 1.0

Autonom Truck

Status

Granskad	ALLA	2024-10-13
Godkänd	Gustav Zetterqvist	2024-10-16

Projektidentitet

Grupp E-post: hamwe392@student.liu.se

Beställare: Gustav Zetterqvist, Linköpings universitet
Tfn: 013-28 19 94
E-post: gustav.zetterqvist@liu.se

Kund: Johan Lindell, Toyota Material Handling
E-post: johan.lindell@toyota-industries.eu

Handledare 1: Sebastian Karlsson (ISY)
E-post: sebastian.karlsson@liu.se

Handledare 2: Oskar Bergkvist (Toyota Material Handling)
E-post: oskar.bergkvist@toyota-industries.eu

Handledare 3: Andreas Bergström (Toyota Material Handling)
E-post: Andreas.Bergstrom.ext@toyota-industries.eu

Kursansvarig: Daniel Axehill, Linköpings universitet
Tfn: +4613284042
E-post: daniel.axehill@liu.se

Projektdeltagare

Namn	**Ansvar**	**E-post**
Hampus Westerberg	Projektledare (PL)	hamwe392@student.liu.se
Joel Wiklund	Testansvarig (TA)	joewi329@student.liu.se
Carl Liljeberg	Informationsansvarig (IA)	carli426@student.liu.se
Mohammad Rajabi	Dokumentansvarig (DOK)	mohra735@student.liu.se
Anton Lund	Mjukvaruansvarig (MA)	antlu106@student.liu.se
Simon Svahn	Designansvarig (DES)	simsv926@student.liu.se
Tomasz Mazurek	Sekreterare (SEK)	tomma870@student.liu.se

INNEHÅLL

1 Inledning 1
2 Systembeskrivning 1
3 Truckens specifikationer 3
4 Kartläggning 4
4.1 LIDAR . . . 4
4.2 Kartläggning . . . 5
4.3 Bresenhams linjealgoritm . . . 5
4.4 Bayesianskt filter . . . 7
4.5 Hinderdetektion . . . 7
4.6 Ruttplaneringskarta . . . 7
5 Ruttplanering 9
5.1 Tillståndsmiljön . . . 9
5.2 Ruttplaneringsalgoritm . . . 9
5.3 Kinematisk modell . . . 14
6 Simuleringsmiljö 15
6.1 Simuleringsverktyg . . . 15
6.2 Miljö och hinder . . . 16
6.3 Koordinatsystem och Transformationer . . . 16
6.4 Sensorsimulering . . . 16
6.5 Truckmodell . . . 16
7 Gränssnitt 17
7.1 ROS2 . . . 17
7.2 Noder och *topics* . . . 17
7.3 Användargränssnitt . . . 18
8 Referenser 20

DOKUMENTHISTORIK

Version	Datum	Utförda förändringar	Utförda av	Granskad
0.1	2024-10-07	Första utkastet	Alla	Alla
0.2	2024-10-13	Andra utkastet	Alla	Alla
1.0	2024-10-20	Första versionen	Alla	Alla

1 INLEDNING

Detta dokument är en designspecifikation för ett projekt i kursen TSRT10. Projektet är framtaget i samarbete med Toyota Material Handling, härefter benämnt Toyota, och syftar till att undersöka möjligheterna för hinderundvikande eller *obstacle avoidance*.

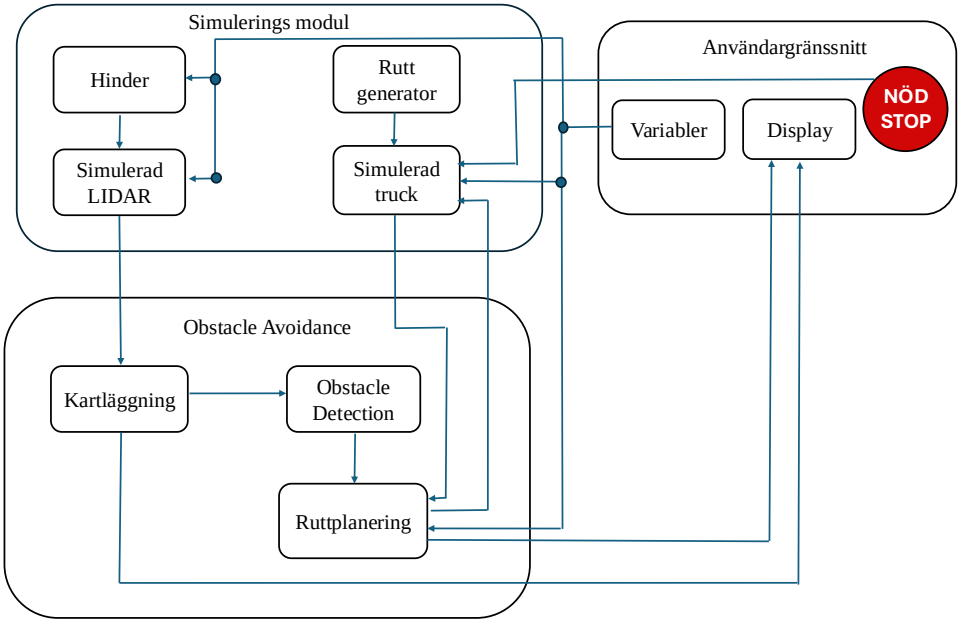
Toyota utvecklar och säljer i dagsläget självkörande gaffeltruckar för arbete i lager och liknande miljöer. I nuläget kan dessa endast köra längs förutbestämda banor i lagret och stannar upp när de möter ett hinder och kräver då hjälp från en anställd. Projektets syfte är att utveckla ett system som låter truckarna själva hitta alternativa rutter runt hindret. Systemet ska primärt utvecklas och testas i en simuleringsmiljö. Vid goda resultat kan systemet eventuellt komma att testas på en verklig gaffeltruck tillsammans med Toyota Material Handling.

2 SYSTEMBESKRIVNING

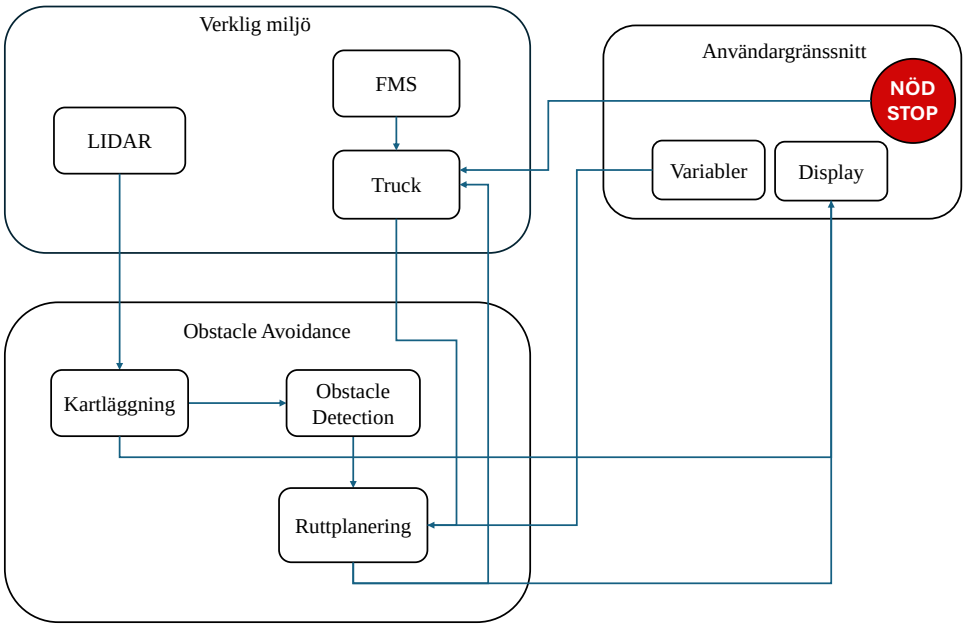
Det simulerade systemet består av tre huvudmoduler, där varje modul innehåller flera undermoduler. För översikt se Figur 1. Kommunikationen mellan modulerna samt undermodulerna sker via *topics* i *Robotic Operating System* (ROS) som är ett flexibelt ramverk för att hantera meddelandeutbyte i robotiksystem. Se Avsnitt 6 för mer information om ROS. Nedan ges en kort beskrivning av huvudmodulerna och deras syften:

- **Simuleringsmodulen** fungerar som en simuleringsvärld där hinder, LIDAR-data, gaffeltrucken och dess ursprungliga rutt modelleras och simuleras.
- **Användargränssnitts** syfte är att möjliggöra konfigurering av truckens dimensioner och kinematik, visa en karta över den kartlagda omgivningen samt konfigurera miljö- och hinderparametrar.
- **Obstacle avoidance-modulen** utför kartläggning av truckens omgivning med hjälp av LIDAR-data. Kartan används sedan till detektion av hinder och planering av ny rutt runt hindret.

Om projektet blir framgångsrikt kommer den hinderavvikelsealgoritmen som utvecklats av projektgruppen att implementeras i en verklig truck hos Toyota. En övergripande bild av det faktiska systemet presenteras i Figur 2.



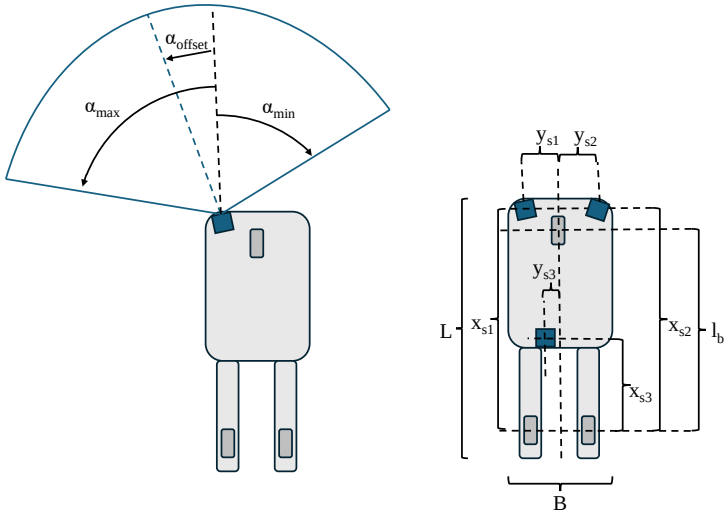
Figur 1: Generell översikt över det simulerade systemet och kommunikationen mellan modulerna (indikerat med blåa pilar).



Figur 2: Generell översikt över det faktiska systemet. Till skillnad från simuleringen antas den verkliga trucken vara försedd med en nödstoppsknapp, vilket inte framgår av denna figur.

3 TRUCKENS SPECIFIKATIONER

Den verkliga trucken tillverkas av Toyota och är försedd med LIDAR-enheter för att uppfatta omgivningen samt detektera hinder. Truckarna kommer i olika storlekar, vilket lägger krav på parametrisering i mjukvaran. Truckens övergripande mått går att se i Figur 3. Parametrarnas betydelse går att se i Tabell 1.



Figur 3: En bild över relevanta mått och storheter på trucken

Tabell 1: Visar truckens relevanta mått

Parameter	Beskrivning
α_{max}	Sensorfältets maximala vinkel
α_{min}	Sensorfältets minimala vinkel
α_{offset}	Vinkel mellan fordonets framåtriktning och sensorns centrumlinje
y_s	Sensorns förskjutning från truckens centrumlinje
L	Truckens totala längd
B	Truckens totala bredd
l_b	Avståndet mellan truckens fram- och bak-hjul
$x_{s1,s2,s3}$	Alla tre sensorers förskjutning i x-led
$y_{s1,s2,s3}$	Alla tre sensorers förskjutning i y-led

4 KARTLÄGGNING

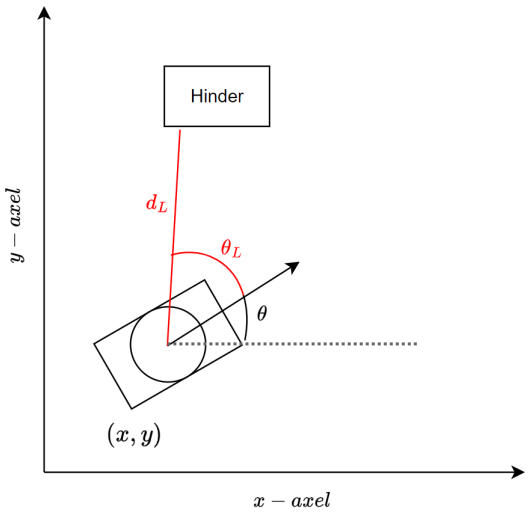
I det här avsnittet beskrivs hur sensordata från LIDAR används för att kartlägga gaffeltruckens omgivande miljö.

4.1 LIDAR

En LIDAR-enhet med 360 graders synvinkel används i *Gazebo*, en simuleringsmjukvara, för att simulera punktmoln som kartläggningsmodulen använder för att kartlägga truckens omgivning. Varje mätning från LIDAR-enheten representeras som (d_L, θ_L), där d_L är avståndet och θ_L är vinkeln till mätpunkten, se Figur 4. Denna data ska först transformeras till kartesiska koordinater innan den används för kartläggningen. Eftersom truckens position och orientering, betecknad som (x, y, θ), i den kartan antas vara känd har denna transformation följande form:

$$x_L = x + d_L \cdot \cos(\theta + \theta_L)$$
$$y_L = y + d_L \cdot \sin(\theta + \theta_L)$$

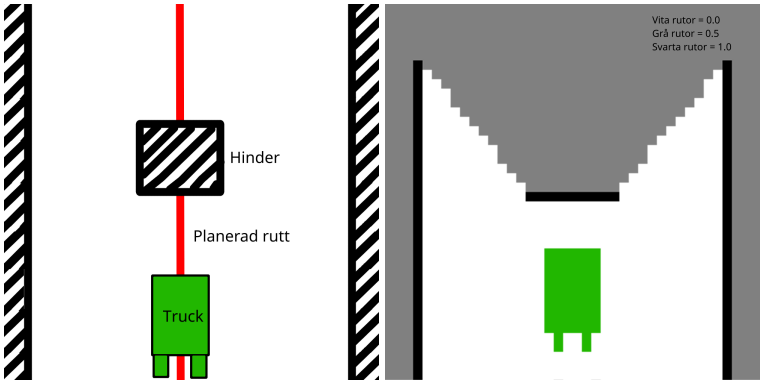
där x_L och y_L är LIDAR-mätningen omvandlad till kartesiska koordinater.



Figur 4: Truckens position och orientering i globala koordinatsystemet, tillsammans med en LIDAR-mätning (markerad i rött).

4.2 Kartläggning

Den kartesiska LIDAR-datan, tillsammans med Bresenhams linjealgoritm (se 4.3), används för att bygga en diskret rutnätskarta. Varje ruta i kartan har storleken 10 cm x 10 cm och tilldelas ett värde mellan 0 och 1, som representerar sannolikheten för att rutan är upptagen av ett objekt. Storleken på rutorna kommer att kunna ändras med hjälp av en variabel i användargränssnittet. I användargränssnittet visualiseras denna sannolikhet med hjälp av färgskalor, där svart indikerar en hög sannolikhet och vit en låg sannolikhet för att rutan är upptagen. Sannolikheten för varje ruta beräknas och uppdateras efter varje LIDAR-mätning med ett Bayesianskt filter (se 4.4). Figur 5 visar ett konceptuellt exempel på hur den diskreta kartan kan se ut i gränssnittet.



Figur 5: Skiss av kartan som kommer att användas i simuleringsmiljön (vänster) och vad som är förväntat att den diskreta kartan kommer visa (höger).

4.3 Bresenhams linjealgoritm

För att uppdatera ockupationssannolikheten för rutorna i den rasterbaserade kartan krävs ett samband mellan rutorna och LIDAR-mätningarna. När en laserstråle färdas från sensorn till en träffpunkt indikerar detta att linjen mellan sensorpositionen och hindret har låg ockupationssannolikhet, medan rutan där LIDAR registrerar en mätning har hög sannolikhet för ockupation. Eftersom kartan är uppbyggd av ett diskret rutnät, representeras denna linje av en sekvens av rutor som på bästa sätt approximerar linjen mellan sensorpositionen och mätpunkten. Algoritmen som används för att göra denna linjeapproximation är Bresenhams linjealgoritm.

Algoritmen utgår från två punkter, i detta fall truckens position (x, y) och LIDAR-mätningens position (x_L, y_L), och beräknar den diskreta linjen enligt Algoritm 1. Figur 6 visar hur den diskreta linjen kan se ut [4]. Som visas i figuren finns det rutor som inte inkluderas i den diskreta linjen, trots att den kontinuerliga linjen passerar genom dem. Detta beror på hur Bresenhams linjealgoritm är utformad.

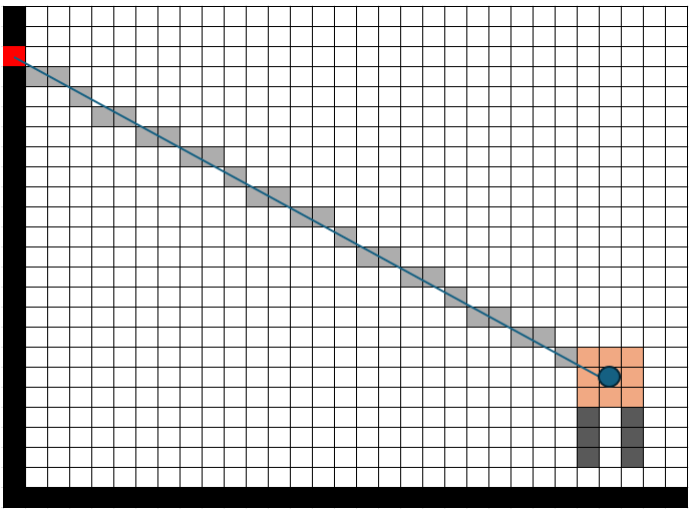
Algorithm 1 Beräkna Bresenhams Linje

procedure BRESENHAMS_LINJE((x,y), (x_L,y_L))
    # Beräkna differenser
    $\Delta x \leftarrow x_L - x$
    $\Delta y \leftarrow y_L - y$

    # Initiera variabler och beslutskriteriet
    $var_x, var_y \leftarrow x, y$
    $p \leftarrow 2\Delta y - \Delta x$

    # Skapa en lista och lägg till den första rutan på Bresenhamslinjen
    bresenhams_rutor_list $\leftarrow (var_x, var_y)$

    # Rita linjen
    while $(var_x, var_y) =!\ (x_L, y_L)$ **do**
        if $p < 0$ **then**
            # öka var_x med 1 och lägg till i listan och uppdatera beslutskriteriet
            $var_x \leftarrow var_x + 1$
            bresenhams_rutor_list $\leftarrow (var_x, var_y)$
            $p \leftarrow p + 2\Delta y$
        else if $p >= 0$ **then**
            # öka var_x och var_y med 1 och lägg till i listan och uppdatera beslutskriteriet
            $var_x \leftarrow var_x + 1$
            $var_y \leftarrow var_y + 1$
            bresenhams_rutor_list $\leftarrow (var_x, var_y)$
            $p \leftarrow p + 2\Delta y - 2\Delta x$
        end if
    end while
    return
    bresenhams_rutor_list
end procedure



Figur 6: En Bresenhamslinje (gråa celler i figuren) given truckens position och en LIDAR-mätning (röda cellen i figuren).

4.4 Bayesianskt filter

Ett Bayesianskt filter ska implementeras enligt [1]. Alla rutor får ett värde $0 \leq p(m_i|z_{1:t}) \leq 1$ där $p(m_i|z_{1:t})$ är sannolikheten att rutan m_i är ockuperad givet LIDAR-mätningen z_t. Dessa värden kommer översättas till ettor och nollor för att kunna användas i ruttplanering. Detta kommer att ske enligt:

$$p(m_{1,0}) = \begin{cases} 0 & \text{om } p(m_i|z_{1:t}) < \alpha \\ 1 & \text{om } p(m_i|z_{1:t}) \geq \alpha \end{cases}$$

där α är avvägd parameter.

4.5 Hinderdetektion

Syftet med programmet är att behandla LIDAR-data från kartläggningen för att identifiera vilka delar av området som innehåller hinder samt ta hänsyn till den planerade rutten och truckens storlek för att avgöra om hindret är tillräckligt stort för att blockera den planerade rutten. Om rutten ej är genomförbar, ska hinderdetektionen informera ruttplaneringsalgoritmen om hindret och avstånd till hindret.

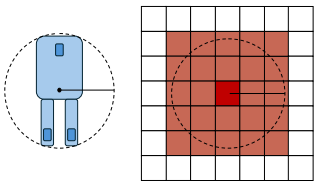
4.6 Ruttplaneringskarta

För att förhindra att trucken kolliderar med föremål skapas en alternativ karta med ett utökat otillåtet område, vilket används av ruttplaneringsalgoritmen. Algoritmen tar den ursprungliga kartan och expanderar de otillåtna områdena för att säkerställa att den planerade rutten undviker kollisioner. För att skapa dessa utökade områden och kontrollera truckens placering genomförs flera steg.

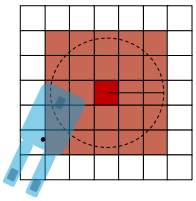
Till att börja med placeras ett antal cirklar runt trucken. Dessa cirklar, med radier som motsvarar det utökade otillåtna området, placeras längs truckens centrumlinje. I Figur 7a och Figur 7c visas exempel där trucken omsluts av en respek-

tive tre cirklar, vilket resulterar i olika stora otillåtna områden. Vid ruttplaneringen kontrollerar algoritmen att ingen av cirklarnas centrumpunkter hamnar innanför det otillåtna området, vilket illustreras i Figur 7b och Figur 7d. Detta säkerställer att trucken aldrig kommer i kontakt med det riktiga hindret.

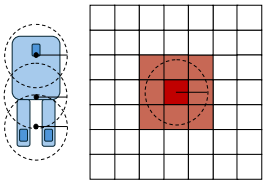
Fler cirklar gör att det otillåtna området på kartan blir mer precist. För att skapa en extra säkerhetsmarginal kan radien för de otillåtna områdena ökas, medan storleken på cirklarna kring trucken förblir densamma. Detta säkerställer att trucken inte kommer för nära hindret. Ruttplanerings-kartan används enbart för planering och visas inte i användargränssnittet.



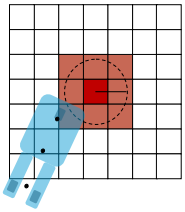
(a) Expansion av det otillåtna området när en cirkel används för att omsluta trucken.



(b) Visar när ruttplanerings-algoritmen inte längre försöker planera in på det stora otillåtna området.



(c) Expansion av det otillåtna området när tre cirklar används för att omsluta trucken.



(d) Fjärde steget i RRT* där noder inom radien γ kopplas om via den nya noden, om det blir billigare.

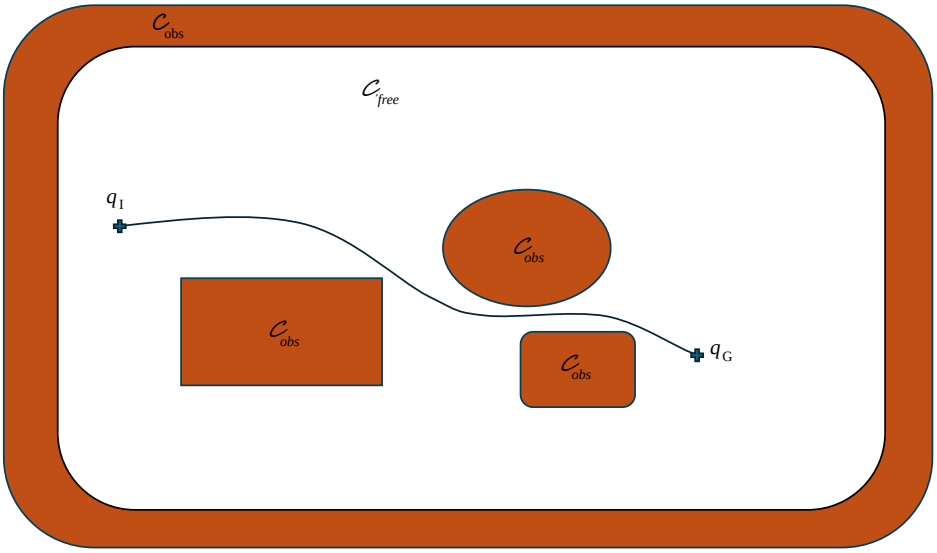
Figur 7: Visar när ruttplanerings-algoritmen inte längre försöker planera in på det lilla otillåtna området.

5 RUTTPLANERING

Detta avsnitt fokuserar på komponenterna i ruttplaneringen som ska implementeras i truckarna. Truckarna kommer att följa en standardrutt i normalläget. Om ett hinder upptäckts av det hinderundvikande systemet så kommer ruttplaneringen att skifta från den ursprungliga rutten till en omdirigerad rutt som leder runt hindret. Sedan återgår den till den ursprungliga rutten så snart det är möjligt. Ruttplaneringsalgoritmen använder sig av kartan som presenteras i avsnitt 4.6. Nedan presenteras en översikt av ruttplaneringsteori och hur ruttplaneringen kommer att genomföras.

5.1 Tillståndsmiljön

Området som trucken kommer att verka i kallas *configuration space*, och betecknas $\mathcal{C}$. Det innehåller alla tillstånd som betecknas q, i vårt fall x koordinat, y koordinat och vinkeln θ. $\mathcal{C}$ har delområden som innehåller hinder, $\mathcal{C}_{obs} \subseteq \mathcal{C}$. Det som blir kvar är det fria området $\mathcal{C}_{free} = \mathcal{C} \backslash \mathcal{C}_{obs}$. Målet för ruttplaneringen är att hitta en rutt från startnoden q_I till slutnoden q_G. Under rutten måste hela trucken befinna sig i $\mathcal{C}_{free}$. Ett bildexempel för tillståndsmiljön går att se i Figur 8.



Figur 8: Tillståndsmiljön för ruttplanering

5.2 Ruttplaneringsalgoritm

Det finns flera ruttplaneringsalgoritmer som beräknar en rutt från en startnod till en slutnod. Nedan följer en förklaring av syftet med algoritmen i det här projektet och vilken algoritm som ska användas.

5.2.1 *Syftet med algoritmen*

Syftet med ruttplaneringsalgoritmen är att hitta en rutt som avviker från den förutbestämda originalrutten runt hindret och sedan återansluter till den förutbestämda rutten. Ruttplaneringsalgoritmen kommer vara av typen *motion planning with differential motion constraints*. Denna typ av algoritm tar hänsyn till truckens kinematik, såsom maximal styrvinkel och längd mellan hjulen på trucken, vilket avgör hur trucken kan manövrera. Algoritmen kommer med detta som begränsning att kontinuerligt försöka hitta en så kort sträcka som möjligt runt hindret givet övriga krav på systemet såsom säkerhetsmarginaler.

5.2.2 *RRT*

RRT står för *Rapidly exploring Random Trees*, och är en algoritm som med hjälp av slumpmässigt placerade punkter successivt bygger ett träd som täcker tillståndsmiljön utifrån startnoden. Ett träd är en graf där två godtyckligt valda noder endast kopplas samman med en väg. Algoritmen slumpar ett tillstånd i det fria utrymmet $\mathcal{C}_{free}$ som benämns q_{rand}. Sedan kollar algoritmen vilket tillstånd i trädet som befinner sig närmast som benämns $q_{nearest}$. Sedan försöker algoritmen koppla samman det närmaste tillståndet med det slumpade tillståndet. Algoritmen har dock en begränsad steglängd, så tillståndet kommer att placeras ut innan om den maximala steglängden är uppnådd. Detta tillstånd benämns q_{new}. Det nya tillståndet kopplas till det närmaste $q_{nearest}$ förutsatt att vägen mellan tillståndet inte går genom ett objekt, d.v.s. $\mathcal{C}_{obs}$. Ett exempel på detta går att se i Figur 9a och Figur 9b. Detta fortsätter tills q_{new} placeras inom en given radie från måltillståndet q_g. Då har en väg från starttillståndet q_i till q_g hittats.

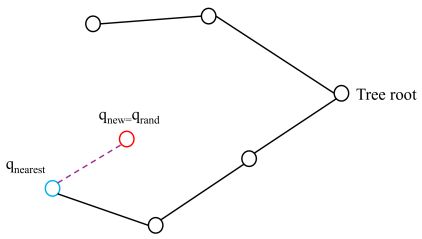
(a) RRT algoritmen utan hinder i vägen

(b) RRT algoritmen med hinder i vägen

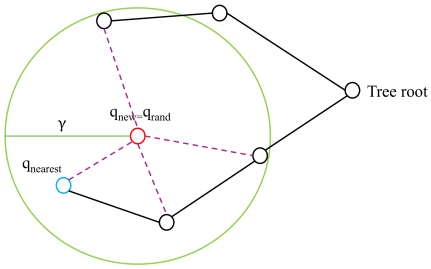
Figur 9: RRT

5.2.3 *RRT**

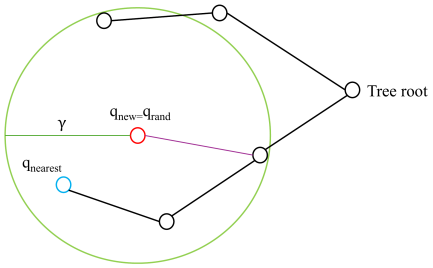
RRT* är en förbättring av RRT som försöker närma sig en optimal ruttplanering. Algoritmen fungerar som RRT fram tills det nya tillståndet placerats ut vilket går att se i Figur 10a. Istället för att koppla till det närmaste tillståndet undersöks vilket tillstånd inom en radie γ som ger lägst kostnad, i det här fallet kortast väg till q_{new} från q_i, och kopplas till den. Detta går att se i Figur 10b och Figur 10c Vidare kopplas övriga tillstånd inom radien om till det nya tillståndet om även de får en lägre kostnad. Detta går att se i Figur 10d. Allteftersom antalet iterationerna går mot oändligheten närmar sig algoritmen en optimal lösning.



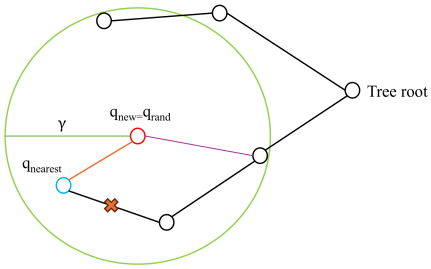
(a) Första steget i RRT* som sker på samma sätt som RRT.



(b) Andra steget i RRT* där den nya noden undersöker vilka noder som den kan kopplas till inom en radie γ.



(c) Tredje steget i RRT* där den nya noden kopplas till det tillstånd som ger den kortast väg till initial tillståndet.

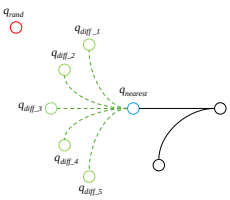


(d) Fjärde steget i RRT* där tillstånd inom radien γ kopplas om via det nya tillståndet, om dess sträcka till initialtillståndet blir kortare.

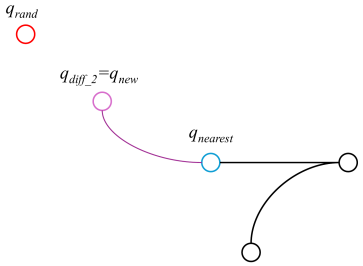
Figur 10: RRT* översikt

5.2.4 *RRT* med kinematik*

RRT och RRT* tar endast fram möjliga vägar enligt beskrivningen ovan, men måste även ta hänsyns till truckens kinematik. Det gör algoritmerna genom att använda rörelseekvationer som tas fram i sektionen nedan. RRT-delen av algoritmen fungerar nästan på samma sätt. Nu genereras punkterna med koordinater men även med en vinkel. Så istället för att generera q_{new} så nära q_{rand} som den begränsade steglängden tillåter så genereras ett antal punkter utifrån ett diskret antal insignaler, i det här fallet styrvinklar. Detta genererar ett antal noder med vinklar, q_{diff} som går att se i Figur 11a. Först görs en kontroll att ingen väg till dessa rutter är blockerade av hinder. Sedan kommer den nod som är närmast q_{rand} att väljas till den nya noden q_{new} vilket går att se i Figur 11b.



(a) Visar hur de noder som är möjliga att nå tas fram med hänsyn till kinematik.



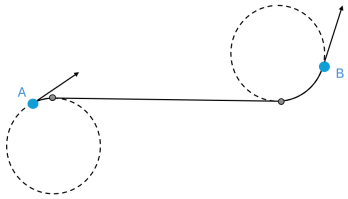
(b) Visar hur en av noderna väljs till q_{new} baserat på kortast avstånd till q_{rand}.

Figur 11: Noder

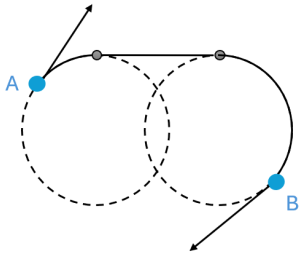
Det stora problemet med kinematik jämfört med tidigare RRT är att tillstånden nu innehåller vinklar som algoritmen måste ta hänsyn till. In- och ut-vinkel är samma för ett tillstånd, något som inte var fallet i avsnitt 5.2.2 eller 5.2.3. Nu kommer vinkeln ut från q_{new} inte vara samma som vinkeln in i det tillstånd som ska kopplas om. För att göra omkopplingen krävs då en ny metod *Dubins path*. Det är rutter som kan visas vara optimala mellan två kända punkter där man har begränsningar på styrvinkel samt start- och slutvinkel. En optimal rutt kommer då att bestå av räta linjer och cirkelsegment med minimal radie R_{min} som bestäms av en maximal styrvinkel. Det finns sex typer av lösningar till dubins path som med notationen R = Right, L = Left och S = Straight är följande: RSL, RSR, LRL, LSR, LSL och RLR. Exempel på detta visas med RSL i Figur 12a, RSR i Figur 12b och RLR i Figur 12c.

5.2.5 *RRT-utvecklingar*

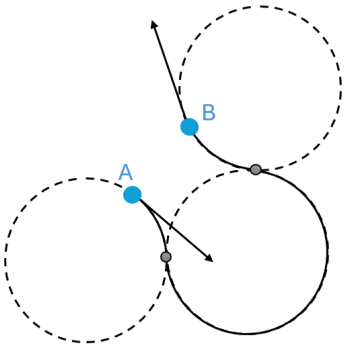
Det finns även ytterligare förbättringar av RRT* som exempelvis *Informed RRT** som försöker att slumpa noder på ett smartare sätt. Detta är dock något som kanske undersöks i mån av tid.



(a) Visar RSL-operationen



(b) Visar RSR-operationen



(c) Visar RLR-operationen

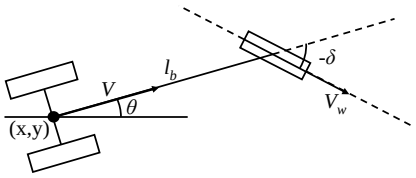
5.3 Kinematisk modell

Trucken approximeras med en trehjuls-modell som går att se i Figur 13. I bilden är δ styrvinkeln, V_w framhjulets hastighet, V bakaxelns hastighet, θ är vinkeln mellan den globala x-axeln och bakaxelns hastighet och slutligen är (x, y) positionen på trucken som ska styras. Utifrån den kinematiska modellen tas rörelseekvationerna fram som går att se i sektionen nedan. Dessa ekvationer används i ruttplaneringsalgoritmen för att hitta rimliga lösningar.

Truckens position bestämd med följande tillståndsvektor:

$$s = [x \quad y \quad \theta] \tag{1}$$

där x och y är koordinater på den bakre axeln och θ är truckens riktningsvinkel. Styrvinkel δ och hjulhastighet V_w skulle kunna vara insignaler, men för att göra problemet mindre komplex och beräkningstungt så kommer hastigheten att vara konstant vid omkörning av hinder. Däremot kommer den konstanta hastigheten att anpassas baserat på den inställda säkerhetsmarginalen för att uppfylla standarder för exempelvis CE-märkning. Insignalen till systemet blir således δ.



Figur 13: En bild över en kinematisk trehjuls-modell.

5.3.1 *Rörelseekvationer/"Differential motion constraints"*

Den kinematiska modellen i globala koordinater tas fram med ekvation (2) till ekvation (5). Sedan tas tillståndsekvationerna fram i ekvation (6).

$$V = V_w cos(\delta) \tag{2}$$

$$\dot{x} = v cos(\theta) \tag{3}$$

$$\dot{y} = v sin(\theta) \tag{4}$$

$$\dot{\theta} = \frac{V_w sin(\delta)}{l_b} \tag{5}$$

$$\dot{s} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} V_w cos(\delta) cos(\theta) \\ V_w cos(\delta) sin(\theta) \\ \frac{V_w sin(\delta)}{l_b} \end{pmatrix} \tag{6}$$

6 SIMULERINGSMILJÖ

Miljön och dess hinder samt truckmodellen simuleras i Gazebo [2] som är en öppen källkodsplattform för robotsimulering. Gazebo erbjuder en fysikmotor och en simulerad miljö som kan läsas av med en simulerad LIDAR. Nav2 [5] är ett ramverk för autonom navigering av robotar och fungerar med ROS2. Vi använder Nav2 för lokal planering, lokalisering och hastighetsreglering.

6.1 Simuleringsverktyg

För att möjliggöra realistisk simulering och visualisering av systemet kommer programmen Gazebo och RViz2 att användas. Dessa verktyg integreras med ROS 2 och används för att simulera miljön och sensorinformation samt för att visualisera truckens beteende i realtid.

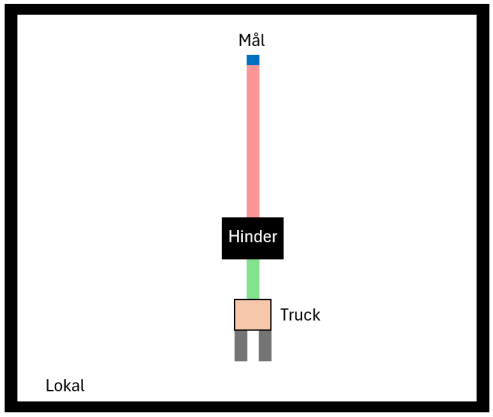
Gazebo är ett fysikbaserat simuleringsverktyg som används för att modellera den fysiska miljön där trucken befinner sig. I vår simuleringsmiljö används Gazebo för att modellera truckens kinematik och omgivning. Gazebo kommer dessutom användas för att generera realistisk sensordata från LIDAR som ska användas för kartläggning. Gazebo integreras med ROS2 genom att publicera och prenumerera på ROS2-topics, vilket gör det möjligt att simulera robotens sensorer och styrning i realtid.

Syftet med RViz2 är att visualisera och övervaka truckens tillstånd och sensordata. RViz2 integreras med ROS2 via ROS2-topics som innehåller truckens position, sensordata, karta och planerad rutt.

6.2 Miljö och hinder

Simuleringsmiljön är utformad för att vara enkel och fokusera på grundläggande funktioner som hinderdetektion och ruttplanering. Miljön består av ett enkelt rum med fyra väggar och ett eller flera lådliknande hinder (se Figur 14). Antalet hinder, deras dimensioner och positioner ska kunna konfigureras så att miljöns komplexitet kan justeras. I mån av tid kommer dessutom andra typer av hinder och rum att implementeras.

Miljön och hinder modelleras i Gazebo och definieras i en SDF (Simulation Description Format) [3] som sedan används i simuleringen av miljön. Miljön och hinder ska även kunna konfigureras genom användargränssnittet.



Figur 14: Trucken i en enkel miljö.

6.3 Koordinatsystem och Transformationer

Systemet kommer att innehålla tre olika koordinatsystem, ett som är fäst vid truckens kropp, ett kopplat till LIDAR-sensorn som sitter på trucken samt det globala referenssystemet som representerar omgivningen. För att kunna spåra truckens position och orientering kommer transformationer att göras med TF2-biblioteket i ROS2.

6.4 Sensorsimulering

I vår simulering används en LIDAR-sensor som monteras på truckens framsida för att samla in data om omgivningen. Denna data används för att upptäcka hinder, skapa en karta över miljön och möjliggöra ruttplanering. Skanningsområdet kan konfigureras genom att välja vilka vinklar den kan täcka. LIDAR-sensorns utdata kommer vara representerad som avstånd och vinklar vilket innebär att datan kommer att behöva bearbetas för att få ett 2D-punktmoln.

6.5 Truckmodell

Trucken modelleras som en enkel rektangulär box med två bakhjul och ett framhjul med den kinematiska beskrivningen som diskuteras i Avsnitt 5.3.

7 GRÄNSSNITT

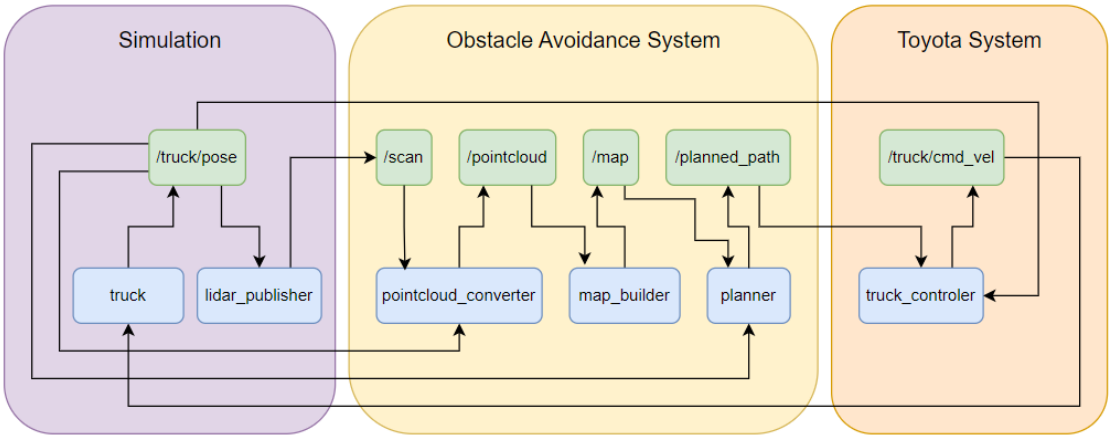
Detta avsnitt handlar om de gränssnitt som kommer användas och utvecklas under projektet.

7.1 ROS2

ROS2 är ett gränssnitt som möjliggör kommunikation mellan olika programvarukomponenter kallade noder. Genom sin ”Publish/Subscribe”-arkitektur kan utvecklingen enkelt göras modulär. ROS2 har även omfattande funktionalitet för att möjliggöra simuleringar, integration av sensorer, felsökning och rörelseplanering. I utvecklingen av Obstacle avoidance av Toyotas lagertruckar kommer ROS2 användas som gränssnitt för ruttplanering, sensormodul, hinder detektion och simulering.

7.2 Noder och *topics*

I Tabell 2 visas alla noder listade med en beskrivning av vad de har för funktion samt hur informationsflödet ser ut mellan dem. Noderna och topics illustreras i Figur 15.



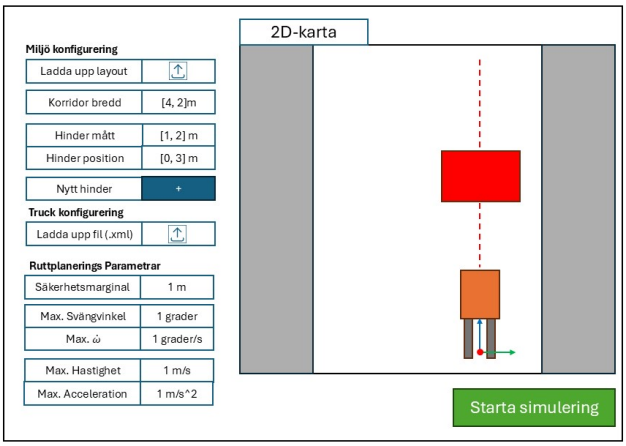
Figur 15: Noder (blå) och *topics* (gröna).

Nodnamn	Topics	Beskrivning av noden
`truck_controller`	`/truck/pose` (input) `/truck/cmd_vel` (output)	Hanterar truckens rörelse genom att styra dess hastighet och publicera dess position och orientering i simuleringen.
`lidar_publisher`	`/scan` (output)	Simulerar LIDAR-sensorn genom att publicera 2D-skanningsdata för att detektera hinder i omgivningen.
`pointcloud_converter`	`/scan` (input) `/truck/pose` (input) `/pointcloud` (output)	Omvandlar LIDAR-skanningsdata till ett 2D-punktmoln, som kan användas för kartläggning och hinderdetektion.
`map_builder`	`/pointcloud` (input) `/map` (output)	Skapar en 2D-karta från det bearbetade punktmolnet som representerar miljön för trucken att navigera i.
`obstacle_detector`	`/map` (input) `/obstacles` (output)	Detekterar hinder i truckens väg baserat på den genererade kartan och publicerar hinderpositioner.
`planner`	`/map` (input) `/obstacles` (input) `/truck/pose` (input) `/planned_path` (output)	Planerar en kollisionsfri väg för trucken baserat på miljökartan och detekterade hinder.

Tabell 2: Översikt över noder, deras *topics* och funktioner

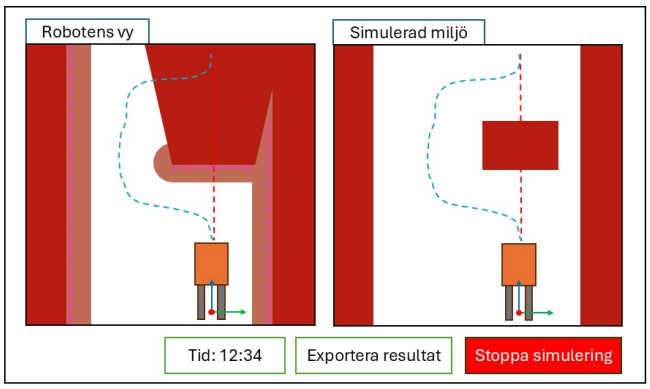
7.3 Användargränssnitt

Användargränssnittet ska möjliggöra konfiguration av truckens dimensioner och kinematik via förprogrammerade modeller. Det kommer finnas parametrar som truckens säkerhetsmarginal till andra objekt, maximal svängningsvinkel, maximal hastighet/acceleration och LIDAR-sensorernas antal och position. Man ska dessutom se en karta av den kartlagda delen av omgivningen, truckens planerade rutt samt simulerade körning. Det ska även vara möjligt att konfigurera miljön och placera ut hinder. En illustration av användargränssnittet finns i Figur 16.



Figur 16: Skiss av användargränssnittet, vy 1

När simuleringen genomförs så syns robotens vy i en del av användargränssnittet och bredvid så syns den simulerade miljön. Sedan kan resultatet av simuleringen exporteras. Detta användargränssnitt syns i Figur 17.



Figur 17: Sketch av användargränssnittet vy 2

8 REFERENSER

REFERENSER

[1] Drew Bagnell. *Occupancy Maps*. https://www.cs.cmu.edu/~16831-f14/notes/F14/16831_lecture06_agiri_dmcconac_kumarsha_nbhakta.pdf.

[2] Open Robotics. *Gazebo Simulation Platform*. https://gazebosim.org/home. 2024.

[3] Open Source Robotics Foundation. *Simulation Description Format (SDF)*. http://sdformat.org/. 2024.

[4] Shivan Pradhan. *Bresenham's Line Generation Algorithm*. https://www.geeksforgeeks.org/bresenhams-line-generation-algorithm/. 2024.

[5] ROS Navigation Working Group. *Nav2*. https://nav2.org/. 2024.