Design Specification

Autonomous Truck with Trailer

December 15, 2022

Version 1.0

ADAPT™

Status

Reviewed	Alfred Sundstedt	2022-10-12
Approved	Carl Hynén	2022-10-19

Project Identity

Group E-mail:	alfsu259@liu.se
Homepage:	https://www.control.isy.liu.se/student/tsrt10/
Orderer:	Shamisa Shoja, Reglerteknik, ISY E-mail: shamisa.shoja@liu.se
Customer:	Daniel Axehill, Reglerteknik, ISY E-mail: daniel.axehill@liu.se
Supervisor:	Carl Hynén Ulfsjöö, Reglerteknik, ISY E-mail: carl.hynen@liu.se
Course Responsible:	Daniel Axehill, Reglerteknik, ISY E-mail: daniel.axehill@liu.se

Participants of the group

Name	**Responsibility**	**E-mail**
Martin Axelsson		marax633@student.liu.se
Jesper Barreng	Test Manager	jesba281@student.liu.se
Isak Bokne	Design Manager	isabo438@student.liu.se
Charlie Elf		chael086@student.liu.se
Terese Johansson	Document Manager	terjo233@student.liu.se
Alfred Sundstedt	Project Leader	alfsu259@student.liu.se
Emil Wiman	Software Architect	emiwi425@student.liu.se

CONTENTS

DOCUMENT HISTORY

Version	Date	Changes made	Sign	Reviewer
0.1	2022-09-16	First draft.	J.Barreng, M.Axelsson	A.Sundstedt
0.2	2022-09-21	Second draft.	J.Barreng, M.Axelsson	E.Wiman
0.3	2022-10-05	Third draft.	J.Barreng, M.Axelsson, T.Johansson, C.Elf, I.Bokne, E.Wiman, A.Sundstedt	A.Sundstedt
0.4	2022-10-12	Fourth draft.	J.Barreng, M.Axelsson, T.Johansson, C.Elf, I.Bokne, E.Wiman, A.Sundstedt	A.Sundstedt
1.0	2022-10-17	Final version.	J.Barreng, M.Axelsson, T.Johansson, C.Elf, I.Bokne, E.Wiman, A.Sundstedt	A.Sundstedt

1 INTRODUCTION

This document states the design specification for the project "Autonomous Truck with Trailer" in the automatic control CDIO project course TSRT10 at Linköpings university. This project is conducted during the fall semester of 2022.

The foundation of the project has been built by prior years' project groups. The documentation from the previous group can be found here [1]. The focus of this year's project is to develop a planning algorithm which can take uncertainty into account when planning trajectories in an environment with dynamic obstacles. Moreover, models for predicting pedestrians and other ground vehicles should be developed. This will in turn require a considerable change in the software architecture and in the visualization system of the truck. The existing LEGO truck can be seen in Figure 1.

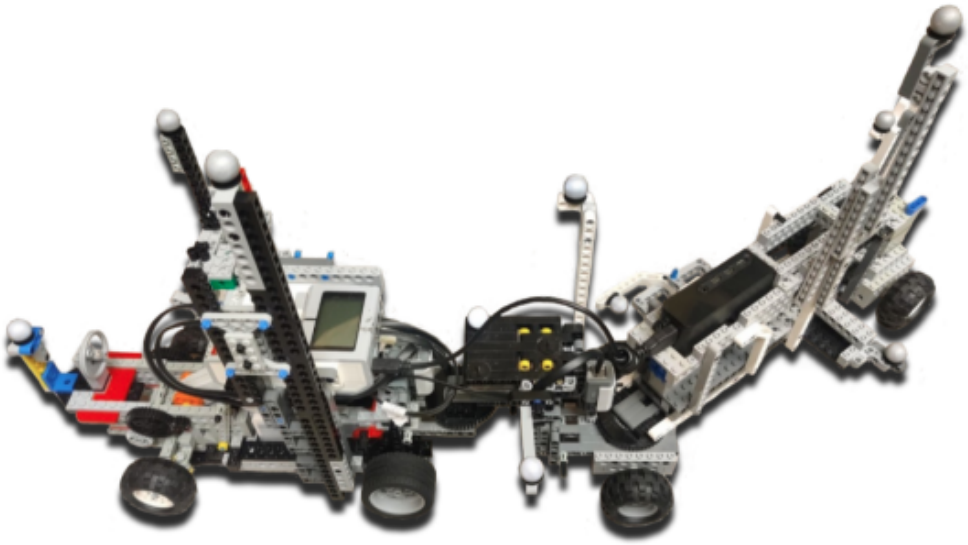

Figure 1: The truck and trailer used in the project

1.1 Background

The autonomous vehicle field are growing fast and the technology is rapidly improving, hence Linköping University created this project to act as a base for future research. Maneuvering a truck with a trailer is a complex task for a truck driver as is. Add dynamic obstacles and the task has to be performed with great care. An autonomous system could be of great use for the truck driver to handle such situations.

1.2 Definition of terms

The terms found in the document are described below.

- **Git** - Software used for version control.

- **MPC** - **M**odel **P**redictive **C**ontrol, a method for process control.
- **Qualisys** - Motion capture and 3D positioning tracking system.
- **ROS** - **R**obot **O**perating **S**ystem, a set of software libraries and tools used for robot apllications.
- **RPi** - Raspberry Pi, a single board computer.
- **Visionen** - An arena for research and education at Linköping University.
- **EV3** - Unit used to control and power the actuators and sensors on the truck.
- **RViz** - A visualization tool used in ROS.
- **POMDP** - **P**artially **O**bservable **M**arkov **D**ecision **P**rocesses. A mathematical framework for decision making with uncertainty. The agent can not observe the full underlying state, hence it is partially observable
- **IMM filter** - **I**nteractive **M**otion **M**odel filter is a filter designed to track several objects that are highly maneuverable.

2 SYSTEM OVERVIEW

The system in this project is a truck with a trailer built in LEGO, with additional components such as a LEGO EV3 and a RPi unit for computing. The EV3 is the control unit for the truck handling the steering and sensor readings. The RPi is the main control unit handling e.g the motion planner, the MPC controller, the predictor and the state observer. Operations and simulations of the truck are handled by a computer equipped with the framework for robot software called ROS. A general illustration of the complete existing system can be seen in Figure 2. The different blocks in Figure 2 represent the subsystems that constitute the whole system. The lines connecting the subsystems represent existing data flows.

The general function of the system is based on a start- and goal node working as input to the system which is given by the system operator, the user. The motion planner takes this as input and plans a path in the environment together with data from the predictor and state observer. This trajectory is then handed to the MPC which sends control signals to the truck for execution. The trucks current position and velocity are fed to the state observer and are used for updating the specific states of the truck. Avoidance of dynamic obstacles are handled in the predictor which affects the control signals given the surroundings of the truck. Given in Figure 2 the state observer also communicates with QualiSys in order to collect data connected to the world.

The visualization part of the project will take place in Visionen. It utilizes a system called QualiSys together with cameras that can detect the truck together with the trailer. A projector can be used to project an obstacle course, the truck will navigate through. A simulation environment is also available to speed up the development and testing process. This allows fast feedback in real time due to the performance can be seen during the simulation.

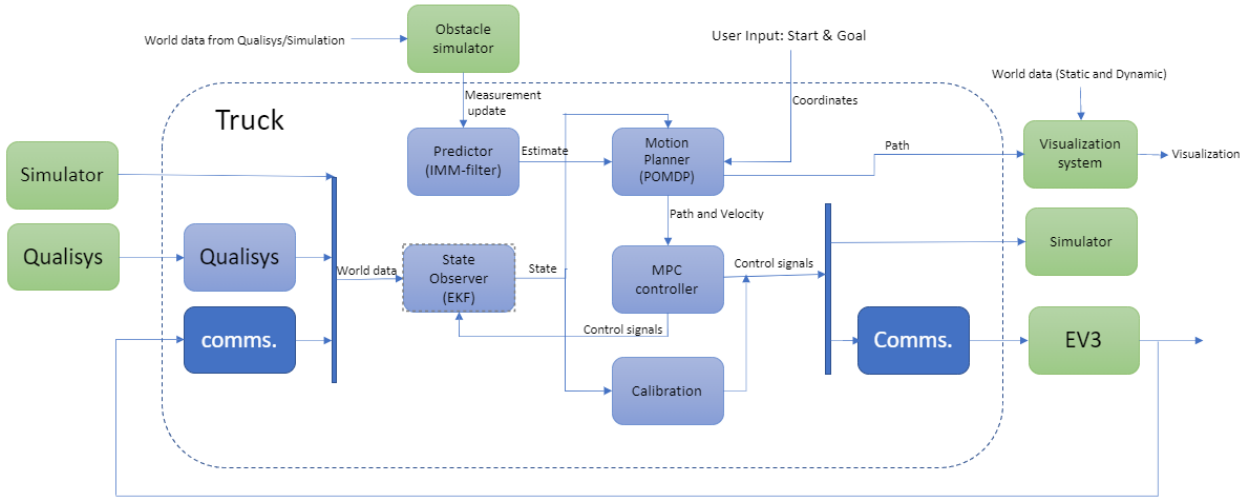

Figure 2: An illustration of the future system

2.1 System description

The system of the autonomous truck can be described generally by the following four subsystems seen in Figure 3

- A control system containing a motion planner and an MPC.
- A LEGO truck that will be controlled by the control signals generated by the MPC.
- A predictor and state observer providing states for the motion planer.
- A visualization system provided in Visionen to track and display the path of the truck and the surrounding obstacles.

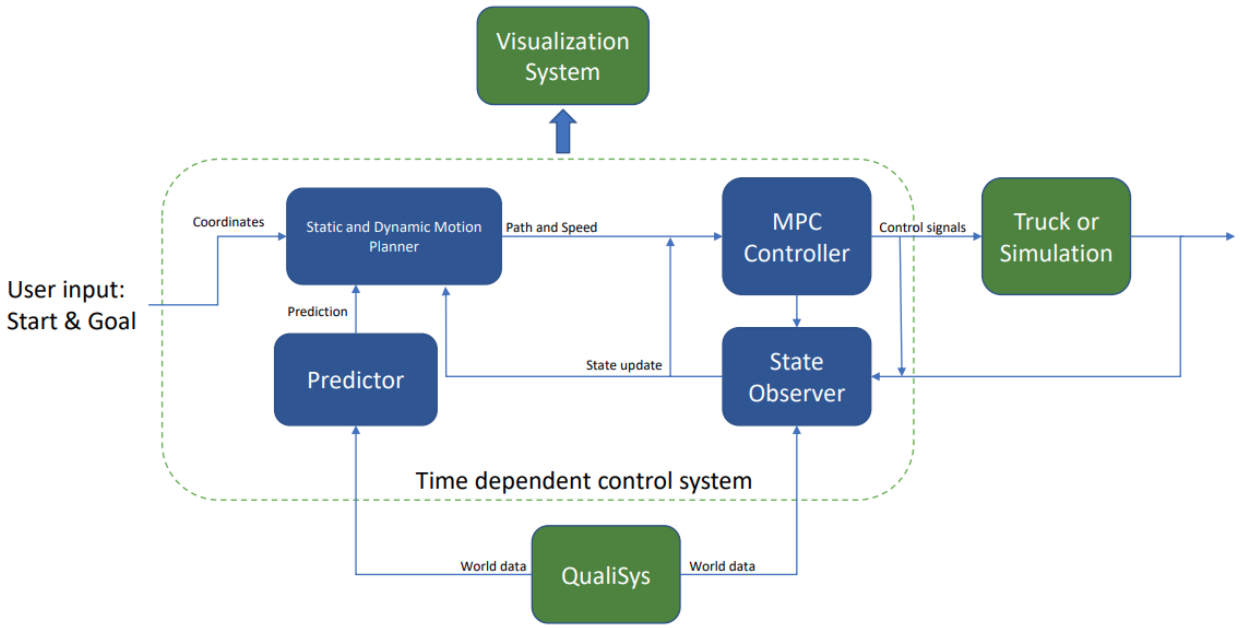

Figure 3: An illustration of the four subsystems and their way of interacting with each other

2.2 ROS

ROS is a set of software libraries and tools used to develop robot applications [INSERT CITE TO ROS]. ROS will in this project be used as a middleware for the software in the project. Each subsystem in the system will be defined as a *node*. This enables the different subsystems to communicate with each other using *topics*.

2.2.1 *Nodes*

In this sections the ROS nodes used in this project are presented, since this is a continuation of last years project, [2] many nodes are the same and can be seen in Table 1.

Table 1: ROS nodes used in the project and their inputs and outputs.

Node	Description	Input(s)	Output(s)
State Observer	Estimates the current state of the vehicle	QualiSys measurements, Actuation commands	Current state of vehicle
Predictor	Predicts the motion of the dynamic obstacle	QualySys measurements	Future states of the dynamic obstacle
Motion Planer	Plans a trajectory for the truck	User inputs, current state of vehicle, future states of dynamic obstacle	Trajectory for the truck
MPC Controller	Follows the trajectory set by the motion planner	Path and velocity	Control signals, updated current state of vehicle
Communications	Sends and receives signals to and from truck	Control signals	Actuation commands
QualiSys	Receives information from QualiSys system in Visionen	QualiSys readings	QualySys measurments
Obstacle Simulator	Simulates the *world* the system is operating in	User input data	Static and dynamic obstacles position
Simulator	Simulates the truck and the surrounding *world*	Same as for the real world scenario	Same as for the real world scenario
Visualization	Projects the *world*, the planed trajectory, static and dynamical obstacles	Static and dynamic obstacles, vehicles current state, trajectory	Visualized data

2.3 Kinematic model of the system

The truck-trailer system is described by Figure 4, obtained from [3]. Figure 4 contains the bodies of the truck, dolly and semitrailer. The three described bodies can be represented by a state vector, $x = \begin{bmatrix} x_3 & y_3 & \theta_3 & \beta_3 & \beta_2 \end{bmatrix}^T$. Here x_3 and y_3 denote the center point of the trailer axle and θ_3 the trailer's heading angle. β_3 represents the joint angle between the trailer and the dolly and β_2 is the angle between the truck and the dolly. The different lengths, L_1, L_2 and L_3 seen in Figure 4, represent the LEGO truck's wheelbase, distance between dolly and LEGO truck, distance between dolly and trailer. α at the front of the truck is the steering angle. The LEGO truck is handed a control signal, $u = \frac{tan\alpha}{L_1}$ which describes the LEGO truck's curvature and the longitudinal acceleration a. The complete kinematic model is given by (1) - (5).

$$\dot{x}_3 = v_3 cos\theta_3 \tag{1}$$
$$\dot{y}_3 = v_3 sin\theta_3 \tag{2}$$
$$\dot{\theta}_3 = v_3 \frac{tan\beta_3}{L_3} \tag{3}$$
$$\dot{\beta}_3 = v_3 \left(\frac{sin\beta_2 - M_1 cos\beta_2 u}{L_2 C_1(\beta_2, \beta_3, u)} - \frac{tan\beta_3}{L_3} \right) \tag{4}$$
$$\dot{\beta}_2 = v_3 \left(\frac{u - \frac{sin\beta_2}{L_2} + \frac{M_1}{L_2} cos\beta_2 u}{Ci(\beta_2, \beta_3, u)} \right) \tag{5}$$

The expression $C_1(\beta_2, \beta_3, u)$ can be replaced by $cos\beta_3(cos\beta_2 + M_1 sin\beta_2 u)$ where M_1 represents the hitching offset, meaning the distance from the LEGO truck's drive axle to the hitch. The expression $C_1(\beta_2, \beta_3, u)$ describes the relation between the trailer's longitudinal velocity, v_3, and the velocity of the LEGO truck's rear axle, v. It is assumed that $C_1(\beta_2, \beta_3, u) > 0$ for control purposes. The equation for the trailer's longitudinal velocity v_3 is given by (6).

$$v_3 = vC_1(\beta_2, \beta_3, u) \tag{6}$$

The three bodies are used in the kinematic motion model and will be implemented in the different systems such as the motion planner, state observer and the MPC controller.

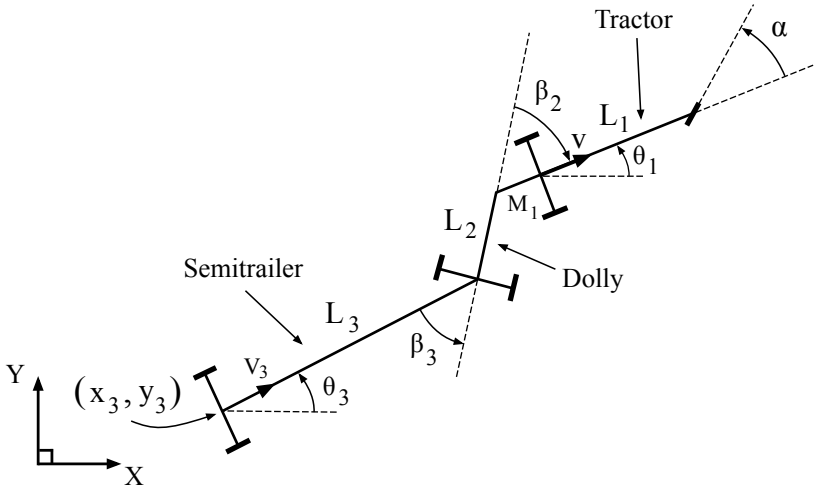

Figure 4: Overview of the kinematic model used for the truck with both a dolly and a trailer.

An overview of the parameters for the kinematic model is presented in Tables 2 - 4.

Table 2: Lengths of parts constituting the truck- trailer system

Name	Description	Value	Unit
L_1	LEGO truck length	0.19	[m]
L_2	Dolly length	0.135	[m]
L_3	Trailer length	0.3	[m]
M_1	Hitch offset length	0.05	[m]

Table 3: Input parameters to the truck-trailer system.

Name	Description	Value	Unit
v	Truck rear axle velocity	-	[m/s]
α	LEGO truck steering angle	-	[rad]
v_3	Trailer rear axle velocity	-	[m/s]
u	Curvature	-	[rad]

Table 4: States of the truck- trailer system.

Name	Description	Value	Unit
x_3	Trailer rear axle x-position	-	[m]
y_3	Trailer rear axle y-position	-	[m]
θ_3	Trailer heading angle	-	[rad]
β_2	Joint angle between dolly and LEGO truck	-	[rad]
β_3	Angle between dolly and trailer	-	[rad]

3 STATE OBSERVER

The state observer is a first order Extended Kalman Filter from here on abbreviated as (EKF). The EKF has a measurement update step and a time update step [4]. The EKF are a discrete-time dynamical system with state x_k and control u_k at time k and can be modeled by (7) - (8).

3.1 Linearization

$$x_{k+1} = f(x_k, u_k) + v_k \tag{7}$$
$$y_k = h(x_k) + e_k \tag{8}$$

where e_k is measurement noise and v_k is process noise. A linear approximation of the system is needed since the uncertainty of a state, the weighting of information from the dynamics and the measurement is based on the linear system corresponding to the non-linear. Hence a linear approximation has to be used. The non-linear system can be

linearized in the previous state estimate by using the Jacobians for the non-linear models according to (9), (10) and (11).

$$F_k = \left.\frac{\partial f}{\partial x}\right|_{x_{k-1|k-1},u_k} \tag{9}$$

$$G_k = \left.\frac{\partial f}{\partial u}\right|_{x_{k-1|k-1},u_k} \tag{10}$$

$$H_k = \left.\frac{\partial h}{\partial x}\right|_{x_{k|k-1}} \tag{11}$$

which results in (7) - (8) can be rewritten as seen in (12) - (13).

$$x_{k+1} = F_k x_k + G_k u_k + v_k \tag{12}$$

$$y_k = H_k x_k + e_k \tag{13}$$

3.2 Prediction step

The prediction step is there to predict the state at the next time instance by using the current state $\hat{x}_{k-1|k-1}$ and the system dynamics. The predicted state $\hat{x}_{k|k-1}$ and its covariance $P_{k|k-1}$ are given by (14) - (15).

$$\hat{x}_{k|k-1} = f(\hat{x}_{k-1|k-1}, u_k) \tag{14}$$

$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k \tag{15}$$

where Q_k is the covariance of the process noise.

3.3 Correction step

In the correction step, information from the measurements and predicted step are merged to create an estimate of the state. The Kalman gain K_k is used to weight the information from the prediction step and measurements and the corrected state estimate $\hat{x}_{k|k}$ and its corresponding covariance $P_{k|k}$ are given by (16) - (18).

$$K_k = P_{k|k-1} H_k^T (R_k + H_k P_{k|k-1} H_k^T) \tag{16}$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k(y_k - h(\hat{x}_{k|k-1})) \tag{17}$$

$$P_{k|k} = (I - K_k H_k) P_{k|k-1} \tag{18}$$

where R_k is the covariance of measurement noise.

3.4 Discretization of dynamics

In order to be able to use the dynamic model presented in (1) - (5) in the EKF, the dynamic model needs to be discretized. This can be done with the Euler forward method using the sample time T_s as seen in (19).

$$\dot{x} \approx \frac{x_{k+1} - x_k}{T_s} \tag{19}$$

which yields the discrete dynamics according to (20).

$$x_{k+1} = x_k + T_s f(x_k, u_k) + v_k \tag{20}$$

4 MOTION PLANNER

The motion planner is tasked with planning a feasible trajectory from the start state to the goal state. It should avoid obstacles along the path, both static and dynamic. The motion planner consists of two main building blocks, a static motion planner and a dynamic motion planner. The static motion planner is used before the mission to plan a path from the start state to the goal state which is free of static obstacles. The static planner from previous years will be used. The dynamic motion planner will be used during the mission to plan a trajectory which is void of dynamic obstacles.

There are two desired main levels of complexity for the motion planner, both expressed in the requirements specification. At the first level (requirement priority 1), the truck will only take longitudinal action to avoid dynamic obstacles, meaning that the trajectory planned by the dynamic planner will follow the path from the static planner. At the second level (requirement priority 2), the truck will take both longitudinal and lateral action to avoid dynamic obstacles, meaning that the dynamically planned trajectory might deviate from the statically planned path.

4.1 Static motion planner

The space in which the truck operates is called *configuration space* and is denoted $\mathcal{C}$. The space contains the states q. In the space are obstacles $\mathcal{C}_{obs} \subset \mathcal{C}$. The free space is $\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obs}$. In order to find a feasible motion plan the search algorithm conducts a search over $\mathcal{C}$. The planner must find a path from start to goal which is possible for the truck to follow while the entire truck is constantly in $\mathcal{C}_{free}$.

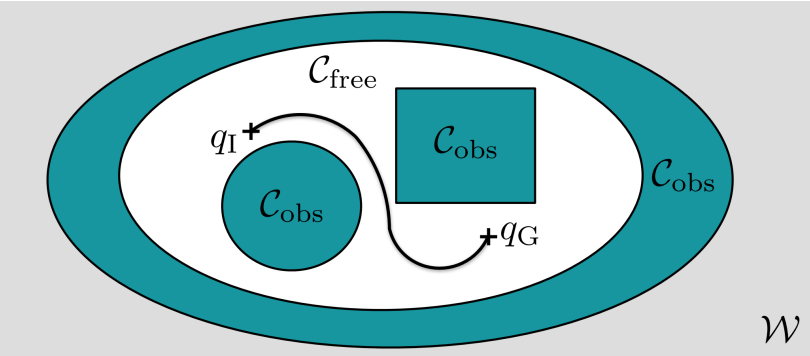

Figure 5: Configuration space together with obstacles

The dynamics of the LEGO truck are limited due to physical aspects and the constraints are described by:

- Control input limitations
- Obstacles
- Constraints on states and state derivatives
  - Velocity
  - Limitations on β_2, β_3 and α

The static planner is a state lattice planner which combines a set of pre-computed motion primitives. The primitives are generated by using the kinematic model described previously. The world is discretized in 1 dm steps in it's lateral and longitudinal directions (x_3^d and y_3^d) and in steps of $\frac{1}{8}\pi$ in θ_3^d. The angles β_2 and β_3 are constrained to zero at the ends of the motion primitives so that all motion primitives can be concatenated with each other to form a continuous path.

4.1.1 *Inputs*

The static motion planner receives inputs from QualiSys about the starting state (the initial state of the truck at static planning) and about the goal state from the user. This is combined to a vector

$$u_{static,planner} = [x_{3,start}, y_{3,start}, \theta_{3,start}, \beta_{2,start}, \beta_{3,start}, x_{3,goal}, y_{3,goal}, \theta_{3,goal}, \beta_{2,goal}, \beta_{3,goal}]$$

which is fed to the planner.

4.1.2 *Outputs*

The static motion planner outputs a feasible path in the form $s_{path} \in S_{path}$ where

$$s_{path} = (x_{3,path}, y_{3,path}, \theta_{3,path}, \beta_{2,path}, \beta_{3,path})$$

These are feasible references for the states of the system. In the case that the dynamical planner is only implemented to take longitudinal action, these will also be the actual references which will be fed to the MPC-controller (along with a time state for each position on the path).

4.2 Dynamic motion planner

In this section the dynamic motion planner is described. The dynamic motion planner will be able to plan suitable longitudinal actions (i.e. slow down, maintain speed, speed up) or lateral actions (deviate from the pre-planned path), depending on the level of complexity as described above. The dynamic motion planner is necessary to navigate in the dynamic environment.

4.2.1 *Inputs*

The dynamic motion planner receives information from the predictor according to Figure 2. The input will come in the form of two vectors for each obstacle motion model. One for the predicted average value which in our case is represented by (x, y) - coordinates, see (21). The other vector is a Gaussian noise, see (22), which represents the system uncertainty where each element is connected to an element pair in (21).

$$\bar{X}_0 = [x_0, y_0, x_1, y_1, ...x_N, y_N] \tag{21}$$

$$\bar{P}_0 = [P_0, P_1, ...P_N] \tag{22}$$

The dynamic planner will also receive the path vector s_{path} from the static motion planner.

4.2.2 *Outputs*

In each time step the dynamic motion planner will compute and output a feasible trajectory in the shape of a list of states:
$s_{trajectory} \in S_{trajectory}$ where $s_{trajectory} = (x_{3,ref}, y_{3,ref}, \theta_{3,ref}, \beta_{2,ref}, \beta_{3,ref}, t)$.

In the first complexity level, only the state t will differ from the static motion planner, the rest will be the same. I.e. t will be updated in each time step such that collisions with dynamic obstacles are avoided. For the second complexity level, all states will be updated dynamically.

4.2.3 *POMDP*

To enable the dynamic motion planner to compute a feasible trajectory with the uncertain and partially observable environment, POMDP will be used. POMDP enables the system to account for the obstacles (e.g. pedestrians) where not all of the necessary states are observable (velocity/intent). The states of the truck itself is seen as fully observable while the uncertainty of dynamic obstacles are only partially observable. The POMDP is described as:

$(S, A, Z, T, O, R, \gamma)$ where:
S is the system's state space ($s \in S$).
A is the system's action space ($a \in A$).
Z is the system's observation space ($z \in Z$).
T is the transition probability function ($T(s', a, s) = P(s'|a, s)$.
O is the observation probability function ($O(s', a, z) = P(z'|a, z)$.
R is the reward ($R(s, a)$).
γ is the discount factor ($\gamma \in [0, 1]$).

The system is in some state $s = (x_3, y_3, \theta_3, \beta_2, \beta_3)$ where action $a = (accelerate, maintain\ speed, slowdown)$ will take the system to state $s' = (x'_3, y'_3, \theta'_3, \beta'_2, \beta'_3)$ with the probability $T(s', a, s)$. At the new state s' the system gets an observation z and a reward R. The goal is to maximize the reward when taking actions on its way to the goal.
The belief (b) over the possible system states over time is key for handling the uncertainty of dynamic obstacles. The states of the truck are seen as fully observable whilst the dynamic obstacles are not. We do not know how the obstacle

Algorithm 1 POMDP

while s_{truck} is not equal to s_{goal} **do**
  $action(a)$
  $s = s'$
  $z = new(z)$
  Update $T(s, a, s')$
  Update $O(s', a, z)$
  $R_{tot} += R(s, a)$
end while

will behave in the next time step. The system will receive an observation (z_t) after taking action (a_t) and then update its beliefs over the system states over time, see (23).

$$b_t(s') = \eta O(s', a, z_t) \sum_s (T(s', a, s) b_{t-1}(s)) \tag{23}$$

The action (a) is based upon the POMDP policy ($\pi(b) = a$). The policy in turn is a mapping of the expected rewards ($V_\pi(b)$). From the possible rewards we choose the policy mapping which gives the highest reward, see (24). Which gives us a sequence of actions ($a_0, a_1, ..., a_t$) to follow for reaching the goal with maximum reward. The actions are executed at each time step.

The rewards ($R(s, a)$) are set according to some preferred behaviours of the truck. It has to drive safe, avoiding running into the dynamic obstacles and it should always reach the goal. Therefore we can set up some rules for the rewards. For safety we choose a safety radius around dynamic obstacles (D_{obs}) and provide it with a low (negative) reward (R_{obs}). If the truck gets within this distance it will receive the reward. For reaching the goal an equal approach is used. The truck needs to get within some distance (D_{goal}) to receive the reward (R_{goal}) which will be a high reward (0 or positive).

$$V_\pi(b) = \sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(b_t)) \tag{24}$$

4.2.4 *DESPOT*

Solving the POMDP planning problem and receiving the policy which will be acted upon is done using DESPOT. We define a DESPOT scenario according to the predictions in (21) and sample elements using the Gaussian noise 22. The computed policy, set of actions (a), and sampling time step Δt will be used to calculate the trajectory of the truck. Going from state $s_0 = (x_0, y_0, \theta_0, t_0)$ following the policy $a_{0..N} = \pi(b)$ will give the state $s_N = (x_N, y_N, \theta_N, t_0 + N * \Delta t)$.

5 PREDICTION OF DYNAMIC OBSTACLES

This section will describe the technical solution for prediction of dynamic obstacles. The goal is to implement a prediction module (predictor) that will use an IMM-filter to predict the future states of dynamic objects.

5.1 Input to the predictor

The input to the predictor comes from an obstacle simulator that simulates measurements from a dynamic object. Ultimately, the predictor collects information from the Qualisys system in Visionen about the dynamic obstacles current state. A dynamic obstacle is represented by the state matrix $\mathbf{Z}$

$$\mathbf{Z} = \begin{bmatrix} x & y & \theta & v_{current} & r & i & \delta & g \end{bmatrix}$$

where x, y, θ are coordinates and heading, $v_{current}$ is the current velocity, r is the radius from the center of the dynamic obstacle, i is an indicator whether it is a pedestrian or a ground vehicle and δ is the current steering angle if it is a ground vehicle. We also have g which is a goal state for the dynamic obstacle. We assume that in each time step, the dynamic obstacle moves in the direction of the goal with its current velocity and some Gaussian noise added. It avoids obstacles using Probabalistic roadmaps, see section 5.2.

5.2 Pedestrian motion model

The dynamics of a pedestrian can be seen in Figure 6.

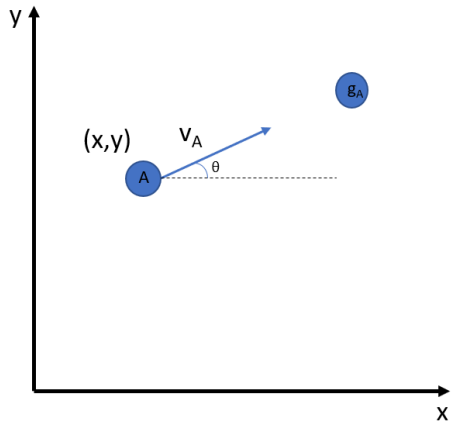

Figure 6: Dynamics for a pedestrian A.

The pedestrian has a position represented by a x,y-coordinate and a heading θ in the global world frame. It also has preferred velocity V_A which it tries to keep as long as the future path remains clear from obstacles. It also has a goal g_A which it is trying to reach. This goal is assumed to be known in advanced by the pedestrian. The pedestrian plans its path towards the goal using Probabalistic roadmap* (PRM*). The algorithm below describes the regular PRM algorithm.

1. Construction phase - Initially we start with an empty roadmap. During this phase the algorithm will try to learn the map by randomly selecting a configuration q in the free configuration space. It will add it to the roadmap if q was previously unreachable (collision-free path) to extend coverage. It may also be added if q adds a connection between two previously unconnected configurations. This process is repeated n times until we have sufficient coverage of the free configuration space.

2. Query phase - In this phase the planner tries to find a path in the previously created roadmap. For this it uses a suitable search algorithm, for example A*. This path connects the goal configuration with the start configuration and we have a motion plan.

PRM* has two important modifications. Firstly the building of the roadmap does not use a fixed radius but a radius that depends on the nodes already placed. Secondly when new nodes are added, their relation to neighboring nodes is not limited by other nodes they have already been added to. Instead, any node that falls within the radius value becomes connected. This algorithm gives the pedestrian a suitable motion plan from a start configuration to a goal configuration. This is implemented into the Obstacle Simulator (see Figure 2).

The motion model used in the IMM-filter for the pedestrian is

$$\mathbf{x}_{k+1} = f_{MODE}(\mathbf{x}_k, \mathbf{u}_k) + g(w), \quad \mathbf{x}_k = \begin{pmatrix} x_k \\ y_k \\ \theta_k \\ v_k \end{pmatrix}, \quad \mathbf{u}_k = \begin{pmatrix} a_k \\ \omega_k \end{pmatrix} \tag{25}$$

$$\mathbf{y}_k = h(\mathbf{x}_k) + \mathbf{e}_k = \begin{cases} y_{1,k} = x_k + e_{1,k} \\ y_{2,k} = y_k + e_{2,k} \\ y_{3,k} = \theta_k + e_{3,k} \end{cases} \tag{26}$$

where the implementation of $f_{MODE}(\mathbf{x}_k, \mathbf{u}_k)$ differs from filter to filter. Here $g(w)$ is some Gaussian noise. The input to the model is acceleration a_k and angular velocity ω_k. Below can the implementation be found for each filter in the IMM-filter:

$$f_{\text{IDLE}}(\mathbf{x}_k, \mathbf{u}_k) \begin{cases} x_{k+1} = x_k \\ y_{k+1} = y_k \\ \theta_{k+1} = \theta_k \\ v_{k+1} = 0 \end{cases} \quad f_{\text{CV}}(\mathbf{x}_k, \mathbf{u}_k) \begin{cases} x_{k+1} = x_k + T_s v_k \cos\theta_k \\ y_{k+1} = y_k + T_s v_k \sin\theta_k \\ \theta_{k+1} = \theta_k \\ v_{k+1} = v_k \end{cases} \quad f_{\text{CVCT}}(\mathbf{x}_k, \mathbf{u}_k) \begin{cases} x_{k+1} = x_k + T_s v_k \cos\theta_k \\ y_{k+1} = y_k + T_s v_k \sin\theta_k \\ \theta_{k+1} = \theta_k + T_s \omega_k \\ v_{k+1} = v_k \end{cases} \tag{27}$$

Each of these implementations aims to model the different movements of a pedestrian.

5.3 Ground vehicle motion model

For ground vehicles, the motion model is described differently then the motion model for the pedestrians. Since vehicles generally are non-holonomic, meaning not having total degrees of freedom, the angular velocity ω_k is substituted to the steering angle δ_k, which is presented in Figure 7. For the car, the weel base l is also relevant, in addition to the position (x, y) and the orientation θ.

The ground vehicles implements a probabilistic roadmap as for the pedestrian motion model. The motion model used in the IMM-filter for the ground vehicles is

$$\mathbf{x}_{k+1} = f_{\text{MODE}}(\mathbf{x}_k, \mathbf{u}_k) + g(w), \quad \mathbf{x}_k = \begin{pmatrix} x_k \\ y_k \\ \theta_k \\ \delta_k \\ v_k \end{pmatrix}, \quad \mathbf{u}_k = \begin{pmatrix} a_k \\ \delta_k \end{pmatrix} \tag{28}$$

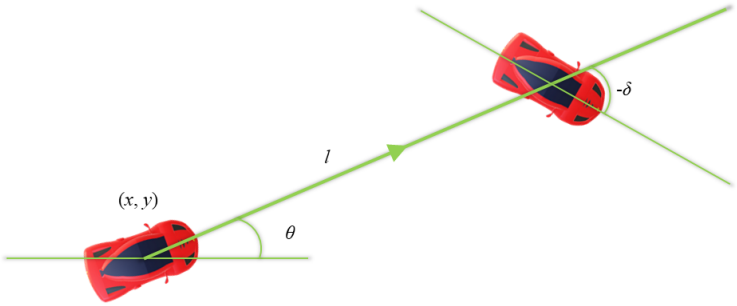

Figure 7: The motion model for the ground vehicles.

$$\mathbf{y}_k = h(\mathbf{x}_k) + \mathbf{e}_k = \begin{cases} y_{1,k} = x_k + e_{1,k} \\ y_{2,k} = y_k + e_{2,k} \\ y_{3,k} = \theta_k + e_{3,k} \end{cases} \tag{29}$$

where the implementation of $f_{MODE}(\mathbf{x}_k, \mathbf{u}_k)$ differs from filter to filter. Here $g(w)$ is some Gaussian noise. The input to the model is acceleration a_k and steering angle δ_k. Below can the implementation be found for each filter in the IMM-filter. Observe that while in IDLE mode, the steering angle δ_k is not constant, meaning that a heading could be set, although the velocity v_k is zero.

$$f_{\text{IDLE}}(\mathbf{x}_k, \mathbf{u}_k) \begin{cases} x_{k+1} = x_k \\ y_{k+1} = y_k \\ \theta_{k+1} = \theta_k \\ \delta_{k+1} = \delta_k \\ v_{k+1} = 0 \\ a_{k+1} = 0 \end{cases} \quad f_{\text{CVCT}}(\mathbf{x}_k, \mathbf{u}_k) \begin{cases} x_{k+1} = x_k + T_s v_k \cos\theta_k \\ y_{k+1} = y_k + T_s v_k \sin\theta_k \\ \theta_{k+1} = \theta_k + T_s v_k \tan(\delta_k / l) \\ \delta_{k+1} = \delta_k \\ v_{k+1} = v_k \\ a_{k+1} = 0 \end{cases} \tag{30}$$

While into CV mode or CA mode, it is presumed that the vehicle is moving forward without turning. This makes the update of the steering angle δ_{k+1} equal to zero.

$$f_{\text{CV}}(\mathbf{x}_k, \mathbf{u}_k) \begin{cases} x_{k+1} = x_k + T_s v_k \cos\theta_k \\ y_{k+1} = y_k + T_s v_k \sin\theta_k \\ \theta_{k+1} = \theta_k \\ \delta_{k+1} = 0 \\ v_{k+1} = v_k \\ a_{k+1} = 0 \end{cases} \quad f_{\text{CA}}(\mathbf{x}_k, \mathbf{u}_k) \begin{cases} x_{k+1} = x_k + T_s v_k \cos\theta_k \\ y_{k+1} = y_k + T_s v_k \sin\theta_k \\ \theta_{k+1} = \theta_k \\ \delta_{k+1} = 0 \\ v_{k+1} = v_k + T_s a_k \\ a_{k+1} = a_k \end{cases} \tag{31}$$

5.4 Tracking

The tracking problem is solved with an IMM-filter. The idea with an IMM filter is to combine several filters with different motion models and use probability theory to determine which filter is the most adequate one at the moment.

This aims to solve the problem of poor prediction quality if one were to use only one motion model. A general overview of the construction of the IMM-filter can be found in Figure 8. This figure shows how the algorithm can predict a future state and also a mode probability. This mode probability is used to determine which filter is the best one at the current time. For the full algorithm consult [5].

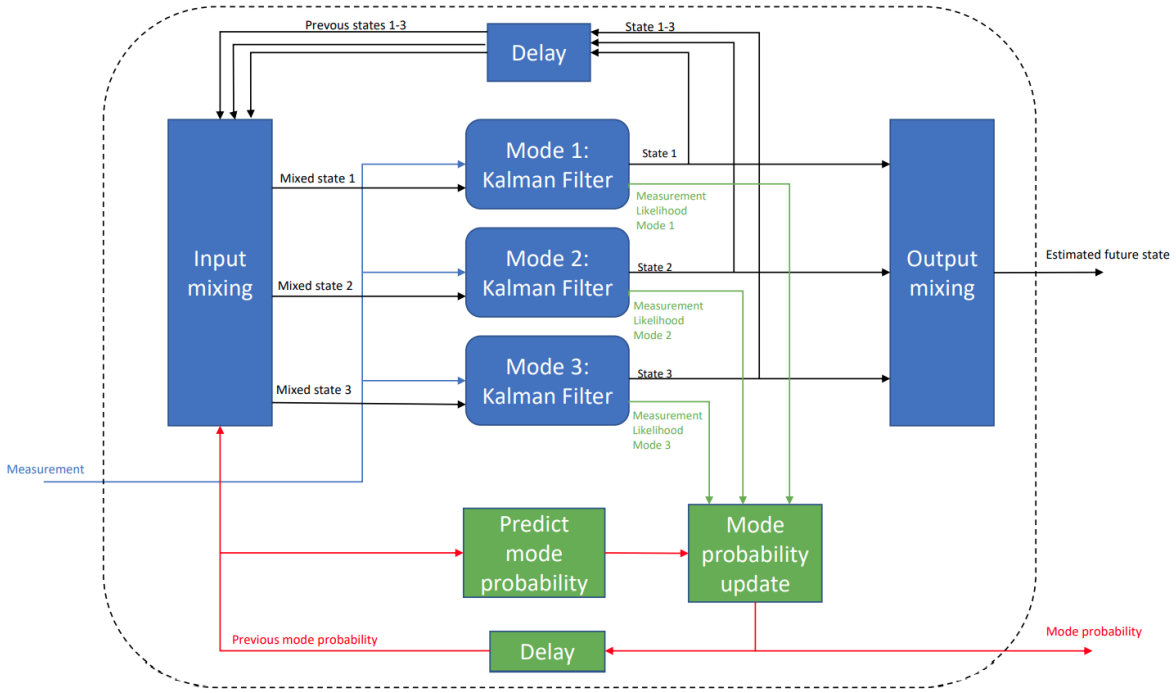

Figure 8: Overview of the IMM algorithm and its components. In this example is the IMM-filter implementing three filters but this can be chosen arbitrary.

5.5 Prediction

The goal of the predictor is to predict a future trajectory for dynamic obstacles. If this trajectory overlaps with the planned path for the truck, the motion planner must take action and re-plan the route to the target to avoid a collision. This results in new control signals to the MPC controller.
As soon as a new measurement arrives, the IMM uses its time update to predict a certain number of states into the future. The prediction algorithm can be seen in Algorithm 2.

5.6 Output from the predictor

The output from the predictor is represented by a number of trajectory lists corresponding to each motion model. A certain list contains means and covariances in each time step until the prediction horizon. These means and covariances can be used to construct Gaussian distributions in each time step. Also a list of mode probabilities is given as output. An illustration of this output is given in (32).

Algorithm 2 Predict Trajectory

while Predictor is running **do**
  $Trajectory \leftarrow Emptylist$
  $x, P \leftarrow IMM(filters, y, u)$
  for all future sample points within horizon **do**
    $x, P \leftarrow IMM.TimeUpdate(x)$
    $Trajectory.insert(x, P)$
  end for
  publish trajectory on topic
  sleep
end while

$$\begin{pmatrix} \text{MM1:} & [x_1, \ldots, x_N], [P_1, \ldots, P_N] \\ \text{MM2:} & [x_1, \ldots, x_N], [P_1, \ldots, P_N] \\ & \vdots \\ \text{Mode Probability:} & [pr(MM1), pr(MM2), \ldots] \end{pmatrix} \tag{32}$$

6 MPC CONTROLLER

This section will describe the MPC controller. The purpose of the MPC controller is to make the truck follow a given trajectory.

6.1 Problem description

The MPC that already exists in the LEGO truck is able to control the truck in accordance to a pre defined path set by the motion planner. When the LEGO truck is given a mission the MPC controller is optimizing the path following in the sense that it is minimizing the error between the path and the truck's position. With dynamic obstacles being introduced in the truck's environment the MPC will follow a nominal trajectory. With information about where and at what time instance the truck will reach a certain point along the path, the MPC controller must also be able to adjust the longitudinal velocity.

6.1.1 *MPC problem formulation*

In this section the problem formulation for the MPC is presented. In (33), x are the states, u are the control input, f is the dynamics of the system, t and T are the time which predicts the horizon, L are the Lagrange integrand and Γ is the Mayer term.

$$\begin{aligned} \text{minimize} \quad & \int_0^T L(x(t),u(t))dt + \Gamma(x(T)) \\ \text{subject to} \quad & x(0) = x_0 \\ & \dot{x}(t) = f(t,x(t),u(t)) \\ & x(t) \in \mathbb{X} \\ & u(t) \in \mathbb{U} \\ & x(T) \in \mathbb{X}(T) \\ & t \in [0,T] \end{aligned} \tag{33}$$

The trajectory following MPC controller is after discretization formulated as:

$$\begin{aligned} \text{minimize} \quad & \sum_{k=0}^{N-1} \left(\tilde{x}_k^T Q \tilde{x}_k + \tilde{u}_k^T R \tilde{u}_k \right) + \tilde{x}_N^T P \tilde{x}_N \\ \text{subject to} \quad & \tilde{x}(0) = x(0) - x_r(0) \\ & \tilde{x}_{k+1} = F_k \tilde{x}_k + G_k \tilde{u}_k \\ & \tilde{x}(k) + x_r(k) \in \mathbb{X} \forall k \in \{0...N-1\} \\ & \tilde{u}(k) + u_r(k) \in \forall k \in \{0...N-1\} \\ & \tilde{x}(N) + x_r(N) \in \mathbb{X}(N) \end{aligned} \tag{34}$$

where the predicition horizon is given by N. The terminal cost is given by $\tilde{x}_N^T P \tilde{x}_N$ and the running cost by $\tilde{x}_k^T Q \tilde{x}_k + \tilde{u}_k^T R \tilde{u}_k$. The cost function seen in (34) is in each iteration solved by the optimization solver acados [6]. For implementation in acados see Section 6.2 and 6.3 .

6.2 Trajectory tracking with MPC

This year a MPC controller will be used to follow a trajectory provided by the motion planer. It is based on the work [7] but with the addition of velocity and acceleration as states in order to handle a trajectory instead of a path. It is assumed that the trajectory provided is feasible and do not contain any obstacles on the trajectory. This is done by using an trajectory following error model. The MPC controller then minimizes the trajectory following error state $\tilde{x} = [\tilde{z_3}, \tilde{\theta_3}, \tilde{\beta_2}, \tilde{\beta_3}, \tilde{v_3}, \tilde{a_3}]^T$.

The variables included in the error state is the following: $\tilde{z_3} = z_3 - z_{3r}$ which is the signed lateral distance from the trailers rear axel to the path. $\tilde{\theta_3} = \theta_3(t) - \theta_{3r}(s(t))$ is the orientation error, $\tilde{\beta_2} = \beta_2 - \beta_{2r}$ which is the joint angle error between the truck and the dolly. $\tilde{\beta_3} = \beta_3 - \beta_{3r}$ is the joint angle error between the dolly and the trailer. $\tilde{v_3} = v_3 - v_{3r}$ is the velocity error and $\tilde{a_3} = a_3 - a_{3r}$ is the acceleration error. For reference see Figure 9. The trailers position on the path is denoted $s(t)$. Included in the MPC controller is the minimization of the curvatures deviation $\tilde{u} = u(t) - u_r(s(t))$ where $u(t)$ is given by $\frac{tan(\alpha(t))}{L1}$ and $u_r(s(t))$ by $\frac{tan(\alpha_r(s(t))}{L1}$.

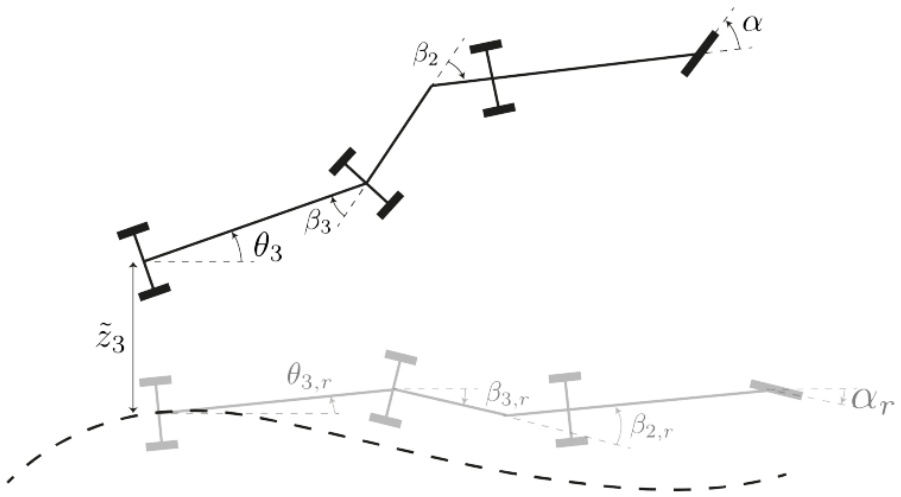

Figure 9: The trailers relation to the trajectory.

The error model for trajectory following can be linearized around $(\tilde{x}, \tilde{u}) = (0, 0)$. Linearizing of the model will then result in (35).

$$\frac{d\tilde{x}}{ds} = A(s)\tilde{x} + B(s)\tilde{u} \tag{35}$$

Where $\frac{d\tilde{x}}{ds} = \frac{d\tilde{x}}{dt}\frac{1}{s}$. Euler-forward is then used to discretize, from which we obtain (36).

$$\tilde{x}_{k+1} = F_k\tilde{x}_k + G_k\tilde{u}_k \tag{36}$$

F_k in (35) can be written as $F_k = I + \Delta_s A_k(s)$ and G_k as $G_k = \Delta_s B_k(s)$. Δ_s is the sampling distance.

6.3 Constraints

In this section the constraints for the system are presented.

6.3.1 *Steering*

The steering angle of the truck will have constraints and will be limited to

$$-\alpha_{max} \leq \alpha \leq \alpha_{max} \iff |\alpha| \leq \alpha_{max}$$

Using the same constraint value for α_{max} as in [2] this will be 0.65 radians, this leads to a curvature constraint:

$$u_{c,max} = \frac{\tan(\alpha_{max})}{L_1} = \frac{\tan(0.65)}{0.19} \approx 4 \Rightarrow |u_c| \leq 4$$

Beside from the steering angle, there will also be limitations on the steering rates.

$$|u_c(k) - u_c(k-1)| \leq \Delta_{u_c,max}$$

The value of $\Delta_{u_c,max}$ is in this case a design parameter and has to be synced with the planner and the system capabilities.

6.3.2 *Velocity and acceleration*

As stated in 2.3 the acceleration will be an input to the system and therefore limitations regarded the maximum velocity, v, and acceleration, a follow. The velocity and accelereration constraints for both forward and reverse motion can be visualized on the form below.

$$v_{max,reverse} \leq v \leq v_{max,forward}$$

$$a_{max,reverse} \leq a \leq a_{max,forward}$$

These restrictions are mainly limited by the system performance and capabilities.

6.3.3 *Jack-knifing*

For the truck to be able to prevent jack-knifing, constraints need to be added to the parameter β_2 and β_3. To these constraints a slack variable ϵ_β is added as well as seen in [2]. This is to make sure that the problem is solvable without compromising the angle constraints. This results in the following constraints:

$$|\beta_2| < \beta_{2,max} + \epsilon_\beta$$

$$|\beta_3| < \beta_{3,max} + \epsilon_\beta$$

The values used for the maximum angles in these constraints are

$$\beta_{2,max} = 0.65$$

$$\beta_{3,max} = 0.75$$

7 SIMULATION SYSTEM

The simulation environment will be the same as for last year, meaning that RViz will be utilized to visualize the truck and trailer as well as both static and dynamic obstacles. In the simulation environment the trajectory from the motion planner can be displayed and one can in real time follow the truck and it's attempt to follow the trajectory as well as it attempting to avoid any dynamical obstacles.

8 VISUALIZATION SYSTEM

This section presents information regarding how *Visionen* will be used in the project.

8.1 Objective

The purpose of the visualization environment is to implement the LEGO truck's mission in a real situation. During a mission, both static and dynamic obstacles will be introduced in the LEGO truck's surroundings. After a finished mission the trajectory which the LEGO truck followed will be displayed in order to evaluate the performance of the implemented systems.

8.2 Communication

The communication will be established between the RPi and a computer that is located in visionen. This will make sure that the RPi communicates the trajectory updates that will occur when a dynamic obstacle is avoided. This communication will be performed via ROS topics.

8.3 Path

The initially planned path for the truck will be displayed before the mission starts. The trucks trajectory and the driven path for the truck will be displayed throughout the mission as well. When the mission is completed the changes in the path from the initial planned path is displayed.

8.4 Visualization system in the ROS-environment

The visualization system will subscribe to the following topics in ROS in order for it to be able to display the wanted information presented above.

- **Trajectory** - The path and trajectory from the motion planer.
- **Xobsstat** - A matrix containing the locations for the static obstacles.
- **Xobsdyn** - A matrix containing the location and time of the dynamical obstacle.

REFERENCES

[1] J. Rosengren, E. Sellén, D. Larsson, D. Similä, G. Ingemarsson, J. Sjöblom, and O. Bergström. (2021) Technical documentation, autonomous truck with a trailer. [Online]. Available: http://www.isy.liu.se/edu/projekt/tsrt10/2021/rev_truck/

[2] D. Larsson, D. Similä, E. Sellén, G. Ingemarson, J. Sjöblom, O. Bergström, and J. Rosengren. (2021) Design specification, autonomous truck with a trailer. [Online]. Available: http://www.isy.liu.se/edu/projekt/tsrt10/2021/rev_truck/

[3] O. Ljungqvist, D. Axehill, H. Pettersson, and J. Löfberg. (2020) Estimation-aware model predictive path-following control for a general 2-trailer with a car-like tractor. [Online]. Available: http://www.researchgate.net/publication/339470933_Estimation-aware_model_predictive_path-following_control_for_a_general_2-trailer_with_a_car-like_tractor

[4] D. Arnström. (2018) State estimation for truck and trailer system using deep learning. [Online]. Available: https://liu.diva-portal.org/smash/get/diva2:1219127/FULLTEXT01.pdf

[5] F. Gustafsson, *Statistical Sensor Fusion*, 3rd ed. Studentlitteratur AB, Lund, 2018.

[6] acados. (2022) acados. [Online]. Available: https://docs.acados.org/

[7] P. Antonsson, E. Bourelius, G. Erbing, J. Gustafsson, A. Holgersson, O. Ismail, F. Jussila, P. Liljeström, K. Rajala, D. Salomonsson, and V. Uvesten. (2020) Design specification autonomous truck with a trailer. [Online]. Available: http://www.isy.liu.se/edu/projekt/tsrt10/2020/rev_truck/images/documents/design_specification.pdf