Technical Documentation

Autonomous Truck with Trailer

December 15, 2022

Version 0.1

ADAPT™

Status

Reviewed	Alfred Sundstedt	2022-12-12
Approved	Shamisa Shoja	2022-12-12

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	2022-12-07	First Draft	Project Group	Project Group

1 INTRODUCTION

This document states the technical documentation for the project "Autonomous truck with trailer" in the automatic control CDIO project course TSRT10 at Linköpings Universitet during the fall of 2022. This year the focus was on handling dynamic obstacles that were present in the test environment. This resulted in a new predictor, a new planner and a new controller. The predictor consists of an IMM-filter which handles multiple motion models. The planner is a POMDP planner able to adjust the movement according to predicted dynamic obstacle movement. The controller is an MPC controller able to steer according to the plan received from the planner.

1.1 Background

The autonomous vehicle field is 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. 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 Project goals

The purpose of this project is to investigate and implement algorithms for autonomous maneuvering of a truck in a complex dynamic environment. The main focus of this year can be summarized as:

- Investigation and implementation of planning algorithms that can handle dynamic obstacles with uncertainty
- Investigation of models used for prediction of pedestrians and other dynamic obstacles
- Development of the systems architecture to implement the new functionality and to enable easier development for future groups
- Development of the truck's visualization system to illustrate its surroundings and current path

The long-term aim of the project is to develop a robust system that can be used for research and education at the Department of Electrical Engineering.

1.3 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 applications.
- **RPi** - **R**aspberry **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 consists of a LEGO truck with a trailer. It is mounted with a LEGO EV3 system as well as an RPi as a computing unit. The EV3 handles steering commands and sensor readings, while the RPi runs the underlying system. The LEGO truck is shown in Figure 1.

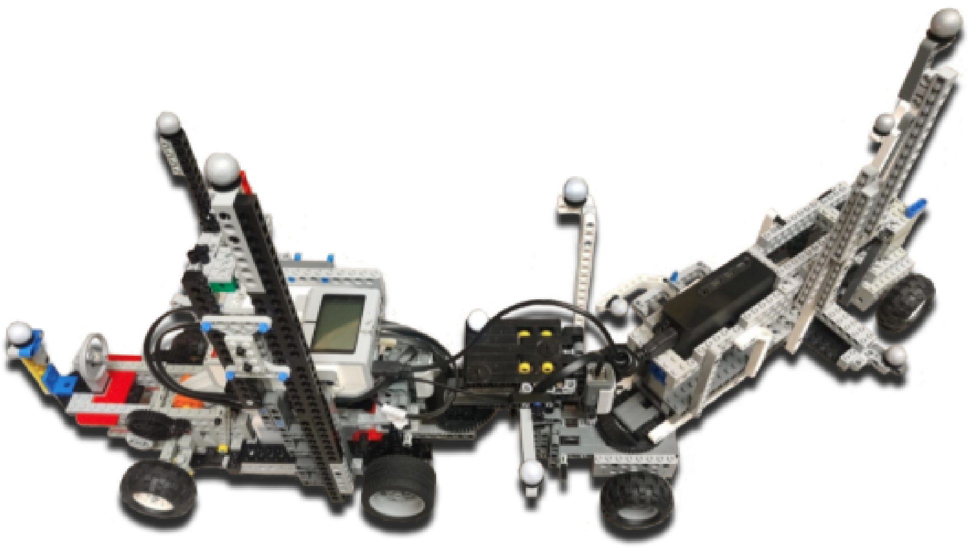


Figure 1: The truck and trailer used in the project

Previous year's groups has developed a visualization system which utilizes the existing positioning and projection system in place at Visionen (QualiSys). This unit visualizes the truck's planned path and obstacles in real time. This has not been used in this project due to lack of time. Instead, a simulation environment has been used for testing and development. The simulation enables faster development of the subsystems. The simulation can be visualized in ROS own simulation environment RViz.

2.1 Problem formulation

The truck and trailer should be able to drive autonomously between two states in an environment with both static and dynamic obstacles, without crashing or causing any dangerous situations. This year a new IMM-filter was developed to better predict and track the motion of a dynamic obstacle. Also a new motion planner was developed to handle the uncertainty when predicting motion for pedestrians and ground vehicles. Lastly a new MPC-controller was developed to fix performance issues with the existing one.

2.2 System description

The system consists of several different subsystems that work together. For an overview of the system, see Figure 2. The subsystems are:

- A control system to compute a feasible path for the truck, this system can further be divided into four subsystems.
  - A state observer, which receives world data and estimates the state of the truck.
  - A motion planner, that should plan a feasible path from a start state to a goal state. It should handle uncertainty and both static and dynamic obstacles.
  - An MPC-controller, which computes feasible control signals so that the truck will follow the planned path.
  - An IMM-filter, which purpose is to track and predict the future trajectory of dynamic obstacles. This information can then be used for the motion planner to re-plan the route if a collision is at risk.
- A LEGO truck, that moves according to the control signals from the MPC.
- A visualization system, which displays the current path and obstacles in Visionen.
- A positioning system, which is used to transmit the position of the truck when running the system in Visionen.
- A simulation environment that utilizes RViz for visualization. Also a obstacle generator that provides the IMM-filter with proper measurements.

The representation of the system seen in Figure 2 has been inspired by previous year's group work [1]. Note also that there are modifications compared to the proposed system in [2].
In the subsequent sections will the different subsystems be discussed in more detail. Also their current status in the project is described.

2.2.1 *Obstacle simulator*

The obstacle simulator is a ROS node that provides the IMM-filter with measurements. The obstacle simulator has been developed by [1] and has been extended to work with the IMM-filter in this project. The obstacle simulator works properly with the IMM-filter.

2.2.2 *IMM-filter*

The IMM-filter is implemented as a ROS node which receives measurements from the obstacle simulator. Given these measurements, the IMM-filter will utilize several Kalman filters to track the current state of the dynamic obstacle and predict the future states. The IMM-filter is working properly and has been tested with the Obstacle simulator in simulation. It has not been integrated so that it can receive measurements from Visionen.

2.2.3 *POMDP Motion planner*

The dynamic planner in the system uses *Partially Observable Markov Decision Process* (POMDP) to take into account the uncertainty of dynamic obstacles. For online solving the package DESPOT is used which provides a belief tree from which actions (slow down, maintain, accelerate) can be extracted to calculate a trajectory from the given planned path [3].

2.2.4 *State Observer*

The state observer fuses measurements from QualiSys (or the simulator) and internal measurements from the EV3. This results in an estimation for all the states in the truck. This has not been modified during this project.

2.2.5 *MPC-controller*

This years MPC controller was developed with acados in order to speed up the solving process of each iteration. This is due to the singularities that occurred with the previous generation MPC controller. The MPC controller is implemented as a ROS node which receives measurements from the motion planner and the state observer. It solves the optimization problem and sends the steering commands to the EV3. The node is working but are not integrated and the underlying solver has some issues, resulting in faulty steering commands.

2.2.6 *EV3*

The EV3 controls the motors and reads different sensor signals. It receives commands from the RPi and reads back the measurements. The system on the EV3 has not been modified during this project.

2.2.7 *Simulator*

The simulator enables testing and development of the different subsystems that does not require the use of Visionen directly. It has mainly been developed by previos year's groups.

2.2.8 *Simulation GUI*

The simulation GUI enables visualization with RViz when running the system in the simulator. It also gives the user the opportunity to input a start and goal state as well as reset the controller. This has not been modified during this project. It was developed by [1].

2.2.9 *Visualization in Visionen*

The visualization system runs on a windows computer located in Visionen and displays information from the subsystems. This has not been developed during this project. It was developed by [1].

2.2.10 *QualiSys*

QualiSys is the positioning system in Visionen. The QualiSys blocks in Figure 2 represents a ROS node that receives measurements from QualiSys. This has not been developed during this project.

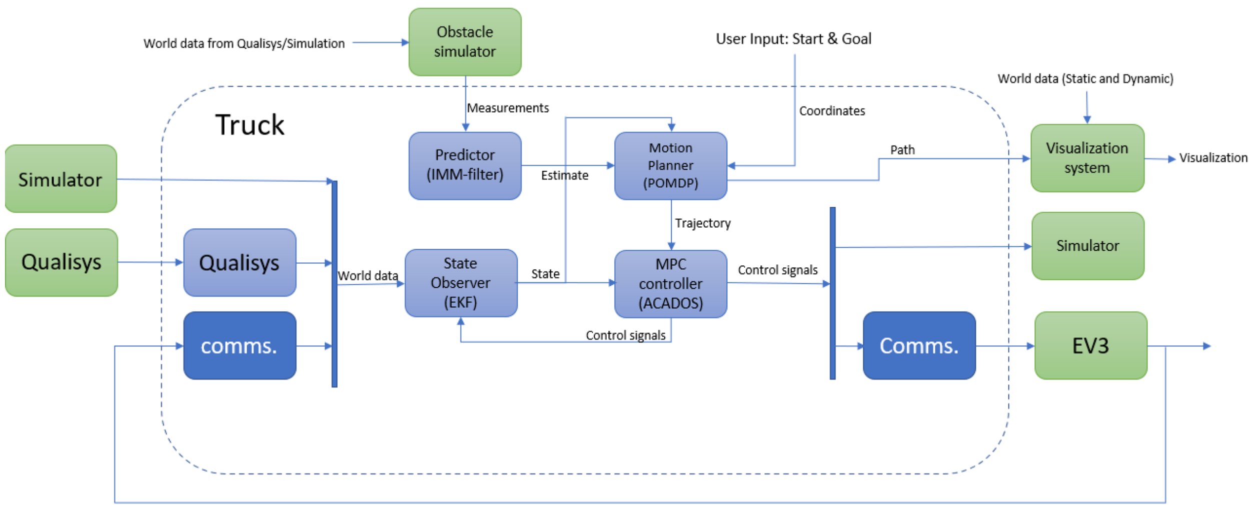


Figure 2: Overview of the system

3 KINEMATIC MODEL

The truck-trailer system is presented in Figure 3, obtained from [4]. Figure 3 contains the bodies of the truck, dolly and semitrailer. These bodies can be represented by a state vector, $x = \begin{bmatrix} x_3 & y_3 & \theta_3 & \beta_3 & \beta_2 & v_3 & \kappa \end{bmatrix}^T$. 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 velocity at the rear axle of the trailer is denoted with v_3 and the curvature of the trajectory is denoted with κ.

The different lengths, L_1, L_2 and L_3 seen in Figure 3, represent the LEGO truck's wheelbase, distance between dolly and LEGO truck, distance between dolly and trailer. M_1 represents the the hitching offset, meaning the distance from the LEGO truck's drive axle to the hitch. The control signal u for the truck is the acceleration of the truck, $\dot{v}$ and the derivative of the curvature $\dot{\kappa}$. The complete kinematic model is given by (1) - (5).

$$\dot{x}_3 = v_3 cos\beta_3(cos\beta_2 + M_1 sin\beta_2\kappa)cos\theta_3 \tag{1}$$

$$\dot{y}_3 = v_3 cos\beta_3(cos\beta_2 + M_1 sin\beta_2\kappa)sin\theta_3 \tag{2}$$

$$\dot{\theta}_3 = v_3\frac{sin\beta_3}{L_3}(cos\beta_2 + M_1 sin\beta_2\kappa) \tag{3}$$

$$\dot{\beta}_3 = v_3(\kappa - \frac{sin\beta_2}{L_2} + M_1 cos\beta_2\frac{\kappa}{L_2}) \tag{4}$$

$$\dot{\beta}_2 = v_3(\frac{1}{L_2}(sin\beta_2 - M_1\kappa cos\beta_2) - \frac{sin\beta_3}{L_3}(cos\beta_2 + M_1 sin\beta_2\kappa)) \tag{5}$$

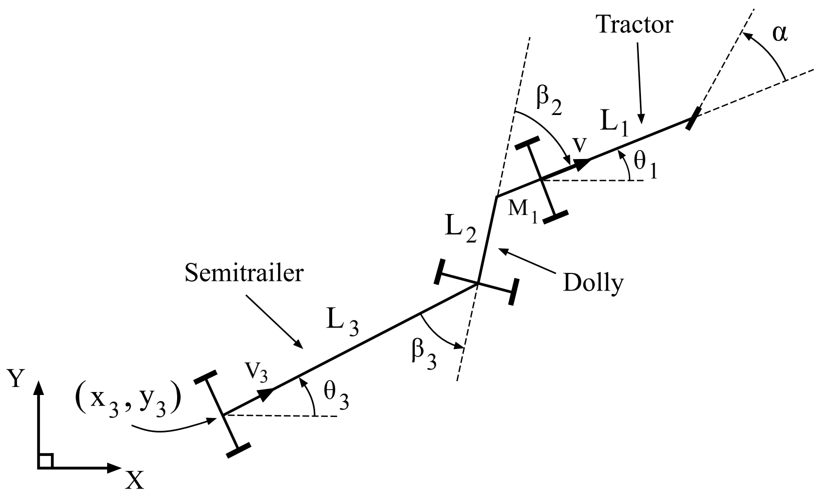


Figure 3: Overview of the kinematic model used for the truck with both a dolly and a trailer.

An overview of the parameters for the kinematic model is presented in Tables 1 - 2.

Table 1: 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 2: 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]
v_3	Rear axel velocity of the trailer	-	[m/s]
κ	Curvature	-	[rad]

4 IMM-FILTER AND MOTION MODELS

The main goal of the IMM-filter [5], is to track the current position of dynamic obstacles, such as pedestrians and ground vehicles. It should also predict the future states (positions) of these dynamic obstacles using some type of motion model to describe the motion of the object. This will result in a predicted trajectory that can be fed to the motion planner for re-planning purposes. An IMM-filter has been developed to achieve this, which utilizes a modified version of the current obstacle generator to generate measurements of the dynamic obstacle.

4.1 Dynamic Obstacles

Two types of dynamic obstacles has been implemented, namely pedestrians and ground vehicles. Their corresponding motion models can be found in Section 4.2. A pedestrian is defined by a state vector **P** as

$$\mathbf{P} = \begin{bmatrix} x & y & v & h & \omega \end{bmatrix}$$

where x and y are the coordinates of the pedestrian, v is the velocity, h is the heading and ω is the angular velocity. In a similar fashion, the ground vehicle is defined by a state vector **V** given as

$$\mathbf{V} = \begin{bmatrix} x & y & v & a & h & \omega \end{bmatrix}$$

where x and y are the coordinates of the ground vehicle, v is the velocity, a is the acceleration, h is the heading and ω is the angular velocity. The reason for adding acceleration as an extra state is to better capture the dynamics of a small ground vehicle (for example a RC-car) since they usually accelerate quite fast. Also, the obstacle generator has supplied the IMM-filter with measurements, given as

$$\begin{bmatrix} x & y & h & i \end{bmatrix}$$

where x and y form the coordinate for the current dynamic obstacle, h is the heading and i is an indicator that is zero if it is a pedestrian and one if it is a ground vehicle. Naturally, the measurement model for both types of dynamic obstacles is given as

$$\mathbf{y}_k = h(\mathbf{x}_k) + \mathbf{e}_k = \begin{cases} y_{1,k} = x_k + e_{1,k} \\ y_{2,k} = y_k + e_{2,k} \\ y_{3,k} = h_k + e_{3,k}. \end{cases} \tag{6}$$

4.2 Motion models

This section presents the different motion models that have been used to capture the motion dynamics of the dynamic obstacles. These have been implemented in the IMM-filter for tracking and predicting a future trajectory.

4.2.1 *Pedestrian*

The pedestrian motion models describes how to update the state vector **P**. The pedestrian has three motion models, IDLE, CV (Constant Velocity [6]) and CTPV (Coordinated Turn Polar Velocity [7]). The aim of these motion models is to describe the movement of a generic pedestrian.

$$f_{\text{IDLE}}(\mathbf{x}_k)\begin{cases} x_{k+1} = x_k \\ y_{k+1} = y_k \\ v_{k+1} = 0 \\ h_{k+1} = h_k \\ \omega_{k+1} = 0 \end{cases} \quad f_{\text{CV}}(\mathbf{x}_k)\begin{cases} x_{k+1} = x_k + T_s v_k \cos(h_k) \\ y_{k+1} = y_k + T_s v_k \sin(h_k) \\ v_{k+1} = v_k \\ h_{k+1} = h_k \\ \omega_{k+1} = 0 \end{cases} \quad f_{\text{CTPV}}(\mathbf{x}_k)\begin{cases} x_{k+1} = x_k + \frac{2v_k}{\omega_k}\sin\left(\frac{\omega_k T_s}{2}\right)\cos\left(h_k + \frac{\omega_k T_s}{2}\right) \\ y_{k+1} = y_k + \frac{2v_k}{\omega_k}\sin\left(\frac{\omega_k T_s}{2}\right)\sin\left(h_k + \frac{\omega_k T_s}{2}\right) \\ v_{k+1} = v_k \\ h_{k+1} = h_k + \omega T_s \\ \omega_{k+1} = \omega_k \end{cases} \tag{7}$$

4.2.2 *Ground Vehicle*

The ground vehicle motion models describes how to update the state vector **V**. The ground vehicle has three motion models, IDLE, CTPV (Coordinated Turn Polar Velocity [7]) and CTPVA (Coordinated Turn Polar Velocity and Acceleration) which can be seen as a modified version of CTPV where constant acceleration is assumed. Here the CV has been removed since we want the ground vehicle to be non-holonomic, meaning it cannot travel in any direction at any time. The aim of these motion models is to describe the movement of a generic ground vehicle.

$$f_{\text{IDLE}}(\mathbf{x}_k)\begin{cases} x_{k+1} = x_k \\ y_{k+1} = y_k \\ v_{k+1} = 0 \\ a_{k+1} = 0 \\ h_{k+1} = h_k \\ \omega_{k+1} = 0 \end{cases} \quad f_{\text{CTPV}}(\mathbf{x}_k)\begin{cases} x_{k+1} = x_k + \frac{2v_k}{\omega_k}\sin\left(\frac{\omega_k T_s}{2}\right)\cos\left(h_k + \frac{\omega_k T_s}{2}\right) \\ y_{k+1} = y_k + \frac{2v_k}{\omega_k}\sin\left(\frac{\omega_k T_s}{2}\right)\sin\left(h_k + \frac{\omega_k T_s}{2}\right) \\ v_{k+1} = v_k \\ a_{k+1} = 0 \\ h_{k+1} = h_k + \omega T_s \\ \omega_{k+1} = \omega_k \end{cases} \tag{8}$$

$$f_{\text{CTPVA}}(\mathbf{x}_k)\begin{cases} x_{k+1} = x_k + \frac{2v_k}{\omega_k}\sin\left(\frac{\omega_k T_s}{2}\right)\cos\left(h_k + \frac{\omega_k T_s}{2}\right) \\ y_{k+1} = y_k + \frac{2v_k}{\omega_k}\sin\left(\frac{\omega_k T_s}{2}\right)\sin\left(h_k + \frac{\omega_k T_s}{2}\right) \\ v_{k+1} = v_k + a_k T_s \\ a_{k+1} = a_k \\ h_{k+1} = h_k + \omega T_s \\ \omega_{k+1} = \omega_k \end{cases} \tag{9}$$

4.3 IMM-filter

The IMM-filter consists of several Kalman filter that operate in parallel, see Figure 4. The idea is to use probability theory to determine which filter is the most accurate given some measurement. This is referred to as the mode probability, meaning that each filter has a certain probability of being the correct one at a certain time step. The algorithm is divided into four steps.

1. *Input mixing* - Given the previous estimate, weight each state and covariance according to the previous mode probability and the transition probability matrix. This gives us a mixed initial state for each filter.
2. *Mode matched prediction update* - This corresponds to the time update in a Kalman filter. The only difference is that we call each Kalman filter to make a new estimate given the mixed state.

3. *Mode matched measurement update* - This corresponds to the measurement update in a Kalman filter. The only difference is that we call each Kalman filter to make a new estimate given the previous estimated state and a measurement. Also we update each filters likelihood given the measurement. This likelihood is then used to update the mode probabilities.

4. *Output mixing* - Final step where we weight each estimate given the mode probabilities and output the most likely one.

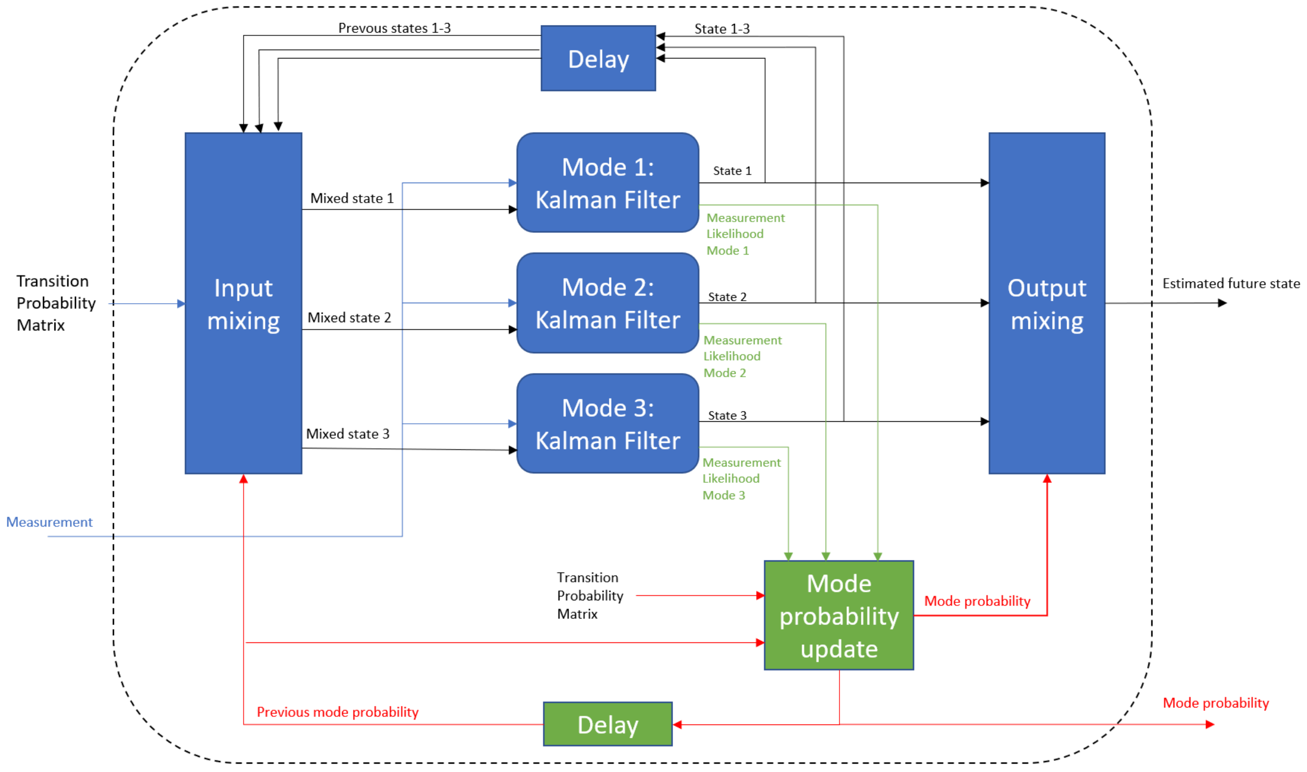


Figure 4: Overview of the IMM algorithm and its components. In this example is the IMM-filter implementing three filters but this can be chosen arbitrary.

4.3.1 *Tracking and initializing the filter*

The first task for the IMM-filter is to track the dynamic obstacle. This is done in a similar manner as if one were to use a standard Kalman filter. The advantage is here, as previously stated, is that several filters are used with different motion models. This will give a better estimate of the motion and therefore tracking than just utilizing one filter. The tracking starts after the filter have received two measurements, since that is needed to properly initialise the filter. Since x, y and h are measured directly, they can be set directly. The initial velocity is set as

$$v_0 = \frac{\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}}{T_s} \tag{10}$$

where index 0 and 1 correspond to the first and second received measurement and T_s the sampling time. Similarly is the initial angular velocity set as

$$\omega_0 = \frac{h_1 - h_0}{T_s} \tag{11}$$

where index 0 and 1 correspond to the first and second received measurement and T_s the sampling time. The initial covariance was set to the identity matrix and the initial acceleration was considered to be zero.

4.3.2 *Prediction of trajectory*

The second task of the IMM-filter is to give a future trajectory of where the dynamic obstacle will be in some time. Given a measurement, the IMM-filter first updates the tracking of the dynamic obstacles and then predicts a trajectory 30 states ahead which corresponds to 6 seconds ahead with $T_s = 0.2$ s. This prediction step consits of calling the most likely time update 30 times in a row to get a proper trajectory, see Figure 5. The algorithm used can be found in Algorithm 1.

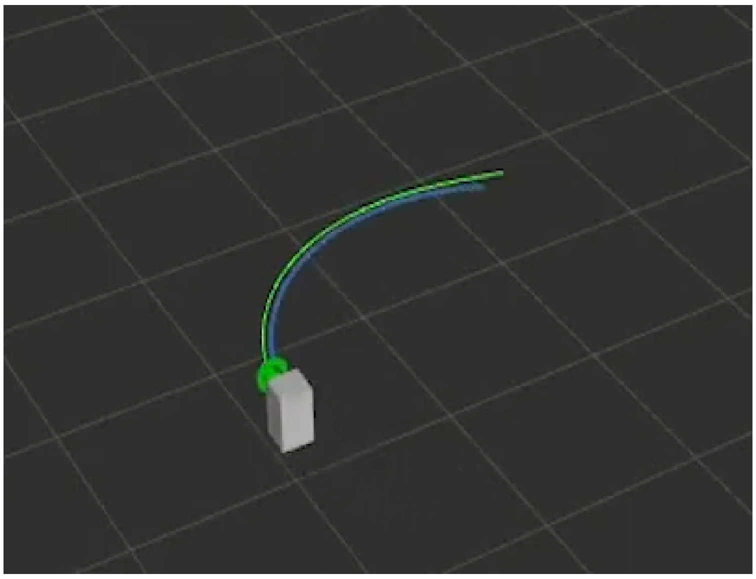


Figure 5: Tracking and prediction of trajectory for the ground vehicle. The white box is the ground vehicle and the green arrow is the estimate of the tracking. The green line is the ground truth and the blue line is the predicted trajectory.

4.4 Output to the Motion Planner

The output from the IMM-filter is represented by seven lists. The first two lists contains the x, y-coordinates in the coming 30 states from the prediction step, see 4.3.2. The following four lists represents each element in the covariance matrix for the x, y-coordinates. The final list corresponds to when in time each of these states will occur. See equation 12 for clarification.

Algorithm 1 Predict Trajectory

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

$$\begin{pmatrix} \text{x:} & [x_1, x_2, \dots, x_N] \\ \text{y:} & [y_1, y_2, \dots, y_N] \\ P_{xx}: & [P_{xx1}, P_{xx2}, \dots, P_{xxN}] \\ P_{xy}: & [P_{xy1}, P_{xy2}, \dots, P_{xyN}] \\ P_{yx}: & [P_{yx1}, P_{yx2}, \dots, P_{yxN}] \\ P_{yy}: & [P_{yy1}, P_{yy2}, \dots, P_{yyN}] \\ \text{time:} & [t_1, t_2, \dots, t_N] \end{pmatrix} \tag{12}$$

5 MOTION PLANNER

The motion planner consists of two ROS nodes, *lego_truck_planner* which is the static planner and *lego_truck_dynamic_planner* which is the dynamic planner. The static motion planner has not been modified this year. See previous years' documentation [1] for information on how it works. The dynamic planner has been implemented from scratch this year.

5.1 Dynamic motion planner

The main task for the dynamic motion planner is to adjust the velocity profile so that the truck won't collide with obstacles crossing the path. The dynamic motion planner utilizes a *Partially Observable Markov Decision Process* (POMDP) to plan a velocity profile along the path, planned by the static planner. The planner can thus only take longitudinal action to avoid a dynamic obstacle, laterally it will only follow the pre-planned path. The planner utilizes the online POMDP solver DESPOT to solve the problem. This solver has previously been used by a research team at Singapore National University for a similar application [8].

5.1.1 *Input*

The dynamic planner receives the following inputs:

- The planned path as a message of the type *Reference.msg* on the ROS topic *reference*
- Predicted path of the dynamic obstacle as a message of the type *PredictionIMM.msg* on the ROS topic */predictor/predicted_trajectory_imm*
- The state of the truck as a message of the type *State.msg* on the ROS topic *estimated_state*

5.1.2 *Output*

The dynamic planner outputs a trajectory as a message of the type *PomdpTrajectory.msg* on the topic */pomdp/velocity*.

5.1.3 *POMDP Model definition*

In each step, the POMDP solver DESPOT outputs one out of three actions (accelerate, maintain speed or slow down). The consequence of each action is simulated and a reward is recieved depending on the consequence. The rewards are used with the online solving in DESPOT with the goal to maximize the reward. The rewards are tune-able and give the possibility to change the behaviour of the dynamic planner, e.g. more passive/agressive planning with regards to speed and risktaking. Rewards are given accordingly:

- Reaching the goal: 10
- Being closer than 1m to a dynamic obstacle: -1000
- High speed: $-3+3*|speed|, speed=[-1,1]$

Observation of dynamic obstacles are done using the input from the IMM-filter. The predicted path of a dynamic obstacle includes position and a timestamp starting from zero (present time) and increases with the time step used in the IMM-filter. Positions of the dynamic obstacles are compared to positions of the Truck simulated in the online solver. An obstacle observation is considered if they are within a radius of 1m. The uncertainty of correctness in dynamic obstacle predictions are then used in the observation probability function in the POMDP model. Using the covariance elements of the obstacle state the probability of observation is calculated linearly starting from the covariance boundary and increasing towards 0.99 as the distance between the truck and the given position of the obstacle shrinks to 0.

5.1.4 *Belief tree*

During run-time the DESPOT online solver runs simulations using the current state and the action space (slow down, maintain, accelerate). After each action an observation is made and a reward is received. This will result in a tree like structure called belief tree. From this belief tree the solver chooses the best action to take in the current state [9]. The belief tree can be multiple levels deep and to create a trajectory all actions in the best belief scenario is extracted.

5.1.5 *Trajectory*

The dynamic motion planner combines the path planned by the static motion planner with the acceleration profile by first converting the acceleration profile into a velocity profile. When the planner knows which velocities the truck has at different discrete points in time, it can compute the length driven at each time stamp. After this, it interpolates positions on the original path for each time stamp. At this point, a trajectory can be constructed by combining the velocities with corresponding states at each time stamp.

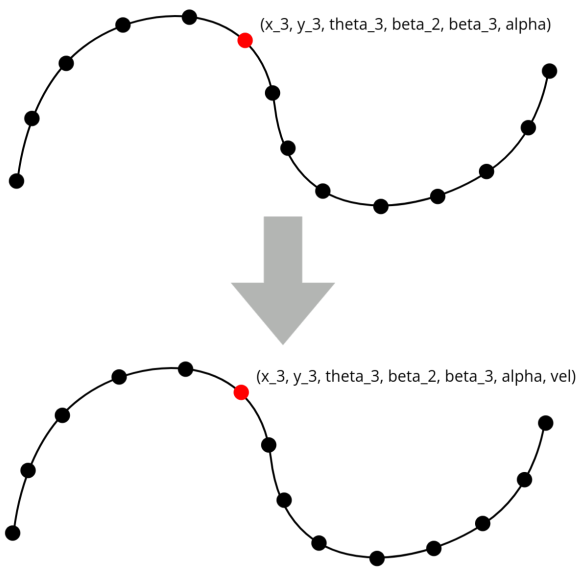

(x_3, y_3, theta_3, beta_2, beta_3, alpha)
(x_3, y_3, theta_3, beta_2, beta_3, alpha, vel)

Figure 6: Illustration of turning the path into a trajectory

5.1.6 *Implementation in C++*

The dynamic motion planner has been implemented in C++ as DESPOT is written in C++ and ROS is compatible with this language. Three .cpp files has been written, which can be found in the *src* folder in the node *lego_truck_dynamic_planner*.

dynamic_planner.cpp defines the problem in DESPOT, *dynamic_planner_world.cpp* is used to communicate with ROS and *main.cpp* is the main file for both the ROS node and DESPOT. The file *planner.cpp* in the DESPOT source has also been modified to make the planner run indefinitely. Please refer to the DESPOT GitHub for instructions on how to define a POMDP model in DESPOT [3].

6 MPC CONTROLLER

This section will present the MPC controller used in this project.

6.1 Problem description

In this project, the MPC controller was developed using the interface provided by acados. For a more detailed view of the interface, visit [10].

From the motion planner the controller will receive seven reference states $[x_3, y_3, \theta_3, \beta_2, \beta_3, v_3, \alpha]$ or more specific a trajectory. Due to the different working frequencies between the motion planner and the MPC, the trajectory sent by the motion planner is interpolated with the current state provided by the state observer. The controller's objective is to follow the trajectory and decelerate or accelerate in order to avoid the dynamic obstacles. This is done via the velocity state sent over by the motion planner which contains information regarding when to decelerate or accelerate. The calculated steering commands necessary to follow the trajectory are sent to the EV3 that handles the motors of the truck.

6.2 MPC problem formulation

In this section the problem formulation for the MPC is presented. In 13, x are the states, u is the control input, f is the dynamics of the system, T is the time horizon, L is the Lagrange integrand and Γ is the Mayer term.

$$\begin{aligned} \text{minimize} \quad & \int_0^T L(x(t), u(t))dt + \Gamma(x(T)) \\ \text{subject to} \quad & x(0) = x_0 \\ & \dot{x}(t) = f(t, x(t), u(t)) \\ & x(t) \in \mathbb{X} \\ & u(t) \in \mathbb{U} \\ & x(T) \in \mathbb{X}(T) \\ & t \in [0, T] \end{aligned} \tag{13}$$

The trajectory following MPC controller is after discretization formulated as:

$$\begin{aligned} \text{minimize} \quad & \sum_{k=0}^{N-1} \left(\tilde{x}_k^T Q \tilde{x}_k + \tilde{u}_k^T R \tilde{u}_k \right) + \tilde{x}_N^T P \tilde{x}_N \\ \text{subject to} \quad & \tilde{x}(0) = x(0) - x_r(0) \\ & \tilde{x}_{k+1} = F_k \tilde{x}_k + G_k \tilde{u}_k \\ & \tilde{x}(k) + x_r(k) \in \mathbb{X} \forall k \in \{0 ... N-1\} \\ & \tilde{u}(k) + u_r(k) \in \forall k \in \{0 ... N-1\} \\ & \tilde{x}(N) + x_r(N) \in \mathbb{X}(N) \end{aligned} \tag{14}$$

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 $\tilde{x}_k^T Q \tilde{x}_k + \tilde{u}_k^T R \tilde{u}_k$ [11].

6.3 Trajectory tracking with MPC

The idea of the trajectory tracking is to minimize the tracking error $\tilde{x}(t) = x(t) - x_r(t)$ where $x(t)$ is the actual position of the truck and $x_r(t)$ is the reference provided by the motion planner. This idea can be seen in Figure 7 even though the states do not match the actual states used.

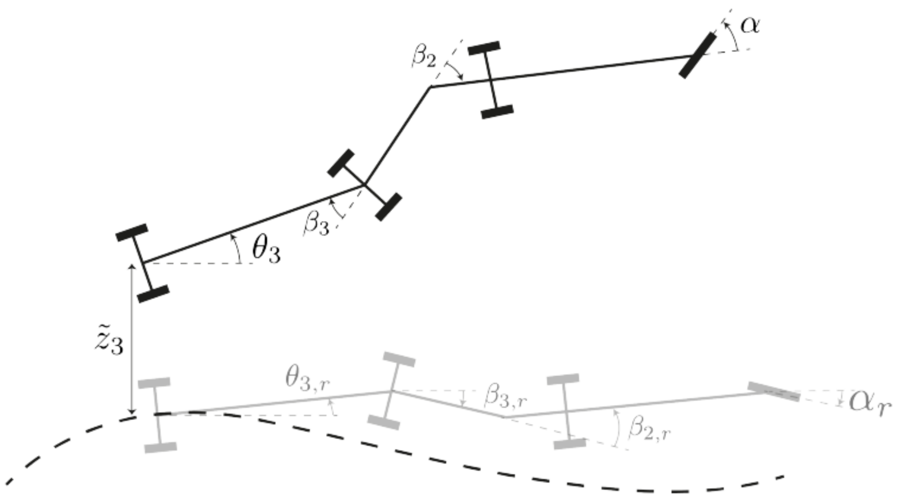


Figure 7: The trailers relation to the trajectory.

6.4 Constraints

In this section the constraints for the system dynamics are presented.

6.4.1 *Steering and curvature*

The constraints of the steering angle of the truck will be limited to:

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

The same constraints value as previous year [12] will be used, with $\alpha_{max} = 0.65$ radians.
Since the implementation does not use the steering angle as a constraint, but rather the curvature, the following calculation is made.

$$\kappa_{min,max} = \frac{\tan(|\alpha|)}{L_1} = \frac{\tan(0.65)}{0.19} \approx 4 \Rightarrow |\kappa| \leq 4$$

6.4.2 *Jack-knifing*

In order to prevent jack-knifing during running, constraints on the parameters β_2 and β_3 were added. The dolly angles need to have a slack variable ϵ_β which allows the angles to deviate slightly from the set values, this is to prevent the truck getting stuck during operation. 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 and minimum angles in these constraints are:

$$|\beta_2| = 0.65$$

$$|\beta_3| = 0.75$$

6.4.3 *Acceleration*

Without information about the performance limitations of the Lego Truck, the maximum and minimum acceleration were set to $|a| = 1$. These constraints are restricted by the physical abilities of the Lego truck itself and therefore an ad hoc-implementation of the constraint were used. The constraints were formulated as:

$$-a_{max} \leq a \leq a_{max}$$

6.4.4 *Change rate of steering angle*

As for the acceleration, the maximum change rate of the steering angle is unknown. To solve this, an ad hoc-solution was found to be $|\dot{\kappa}| = 10\ rad/s^2$. The constraints were formulated according to:

$$-\dot{\kappa}_{max} \leq \dot{\kappa} \leq \dot{\kappa}_{max}$$

6.5 Parameters

In this section the parameters used in this project are presented.

6.5.1 *Tuning parameters*

The tuning parameters used in the implementation are shown in Table 3.

Table 3: Tuning Parameters

Name	Value	Unit
N	100	[-]
T	1	[s]
z_l	1000	[-]
z_u	1000	[-]

Where N is the prediction horizon, T is the time horizon and z is the slack variable for the states and the control inputs.

6.5.2 *Weights*

The weights for the states can be seen in 15.

$$Q = \begin{bmatrix} 1000 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1000 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1000 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.001 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.001 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.001 \end{bmatrix} \tag{15}$$

The weights for the end states can be seen in 16.

$$Q_e = \begin{bmatrix} 10000 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10000 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10000 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.001 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.001 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.001 \end{bmatrix} \tag{16}$$

.

The weights for the control signals can be seen in 17.

$$R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{17}$$

6.5.3 *Limits*

The limits for the different states can be seen in Table 4

Table 4: Limits

Name	**Value**	**Unit**
$\lvert\alpha\rvert$	0.65	[rad/s]
$\lvert\kappa\rvert$	4	[-]
$\lvert\beta_2\rvert$	0.65	[rad]
$\lvert\beta_3\rvert$	0.75	[rad]
| Throttle rate change |	10	[m/s^2]
| Curvature rate change |	10	[rad/s^2]

6.6 Implementation

This years MPC is built using the acados framework. One defines the states, dynamics and constraints in a MATLAB model using CasADi. After that, everything is defined using the pre-existing functions in acados. This includes e.g if it should be a linear or non-linear cost matrix, defining constraints and setting the initial states.

Once the MATLAB part of the implementation is complete acados allows for generation of C-code which will be used in the underlying solver in the MPC node. Much of the code can be generated, however the unique MPC problem needs to be defined once again but this time it needs to be defined in C++ for the solver to work with the MPC node. The MPC node allows for communication with the rest of the system.

7 SOFTWARE IMPLEMENTATION

This project uses ROS as a big part of the implementation, since it allows all the subsystems to communicate with each other. The implemented code this year was written in Python, C++ and MATLAB.

7.1 ROS

Each of the subsystems in the project are implemented as a package in ROS. Each of these will host at least one node that will be running the respective subsystem. The nodes will communicate using *topics* in the ROS framework. This will make the input-output structure of the nodes manageable. The subsystem that needs input from another subsystem can then subscribe to the topic which the subsystem publishes.

Besides this, ROS also provides other tools used in this project. The simulation environment visualization is using RViz that visualizes the data from different topics.

7.2 Packages

The main packages that has been developed or modified during this year's project are presented below.

- lego_truck_obstacle_simulator
  - Developed in Python
  - Modified version from last year to work with the IMM-filter.
- lego_truck_imm
  - Developed in Python
  - Created during this project. For more info, see the repository.
- lego_truck_dynamic_planner
  - Developed in C++
  - Created during this project. Utilizes the POMDP motion planner DESPOT to create an action tree. For more info, see the repository.
- lego_truck_mpc_acados
  - Developed first in MATLAB, then transcribed to C++. Utilizes the optimization solver acados to solve the optimization problem. For more information, see the repository.
  - Created during this project.

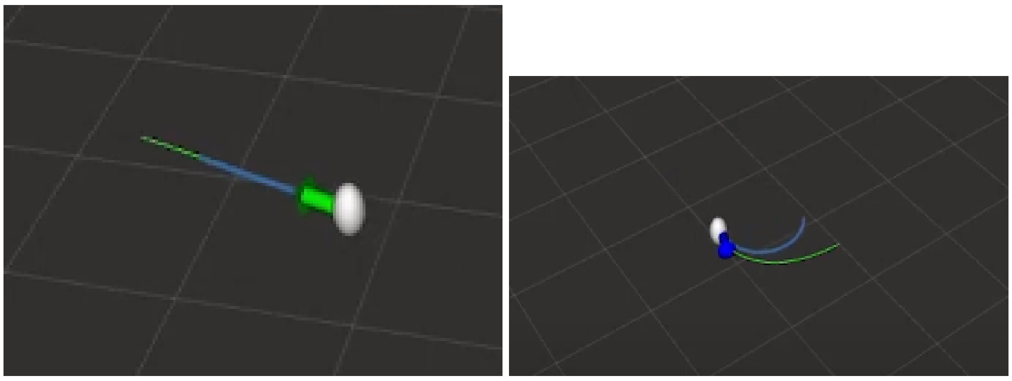
8 RESULTS

8.1 IMM-filter

After implementation of the IMM-filter it has been evaluated using RViz, see Figure 8 and 9. The results in RViz show that the IMM-filter works as expected. The IMM-filter is able to track both pedestrian and ground vehicles and it can utilize all different motion models to do so. In Figure 8a can it be seen that the IMM-filter tracks the position of the pedestrian and that the predicted path is consistent with the ground truth. This also applies for Figure 8b. Here the blue line has not yet converged to the green line since the algorithm requires some measurements to better predict future states. The IDLE mode was omitted here but work similairly as in Figure 9c.

The motion models for the ground vehicles works appropriately as can be seen in Figure 9a, 9b and 9c. The reason for not having a Constant Velocity model here is to better model the non-holonomic constraints a ground vehicle has. The ground vehicle can still move in a straight line, given that the angular velocity is zero. The CTPVA model assumes constant acceleration which explains the overshoot in Figure 9b.

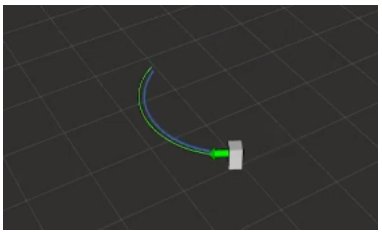
The IMM-filter has also successfully been integrated in the ROS systems and can utilize the measurements from the obstacle generator. It has not been tested in Visionen and it has not been integrated with QualiSys to receive raw measurements from dynamic obstacles.

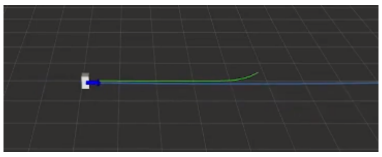


(a) Tracking of pedestrian using the CV model. The white sphere is the position of the pedestrian and the green arrow is the estimate. The green line is the ground truth, meaning where the pedestrian will go in the future and the blue line is the predicted trajectory.

(b) Tracking of pedestrian using the CTPV model. The white sphere is the position of the pedestrian and the blue arrow is the estimate. The green line is the ground truth, meaning where the pedestrian will go in the future and the blue line is the predicted trajectory.

Figure 8: Pedestrian tracking and prediction of trajectory


(a) Tracking of ground vehicle using the CTPV model. The white cube is the position of the ground vehicle and the green arrow is the estimate. The green line is the ground truth, meaning where the ground vehicle will go in the future and the blue line is the predicted trajectory.


(b) Tracking of ground vehicle using the CT-PVA model. The white cube is the position of the ground vehicle and the blue arrow is the estimate. The green line is the ground truth, meaning where the ground vehicle will go in the future and the blue line is the predicted trajectory.

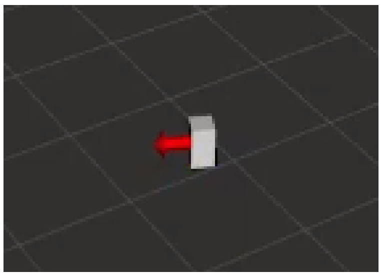

(c) Tracking of ground vehicle using the IDLE model. The white cube is the position of the ground vehicle and the red arrow is the estimate. Here no lines are produced since the obstacle is standing still.

Figure 9: Ground vehicle tracking and prediction of trajectory

8.2 Motion Planner

Results of the motion planner comes in two parts, the originally planned dynamic motion planner and a simplified dynamic motion planner.

8.2.1 *Simplified dynamic motion planner*

After implementation of the dynamic motion planner it has been simulated and visualized using MATLAB. To visualize the result of the dynamic motion planner a straight line mission was done. Simulating a mission with a straight path from $(x, y)_{start} = (0, 0)$ to $(x, y)_{goal} = (10, 0)$ with an obstacle moving as follows: $(x, y)_{obs,start} = (6, 0), time < 25s$ the moving towards $(x, y)_{obs} = (2.875, 0), 25 <= time < 50s$ and lastly moving towards $(x, y)_{obs_{goal}} = (11.125, 0), 50s < time$. The mission results are visualized in Figure 10. In the figure the result shows the truck reference trajectory simulated with a time step of 1 second. As currently implemented the truck is keeping the safety distance to the dynamic obstacle as it moves towards the truck and the truck reverses to avoid a crash (getting to close results in a large negative reward). As the obstacle then moves away and passes the goal the truck keeps progressing until the goal is reached. The small waves in the truck curve is a result of the time step for the truck state transition function.

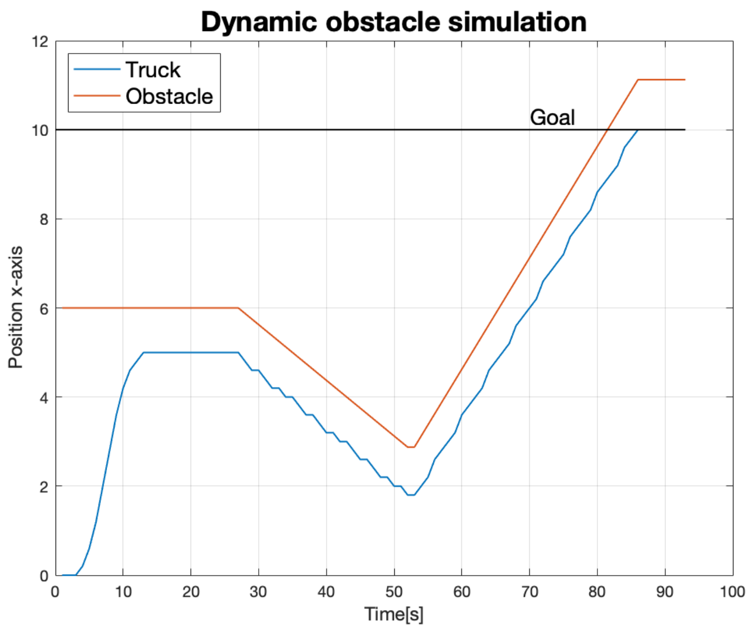


Figure 10: Truck reference output for a straight line mission with a dynamic obstacle.

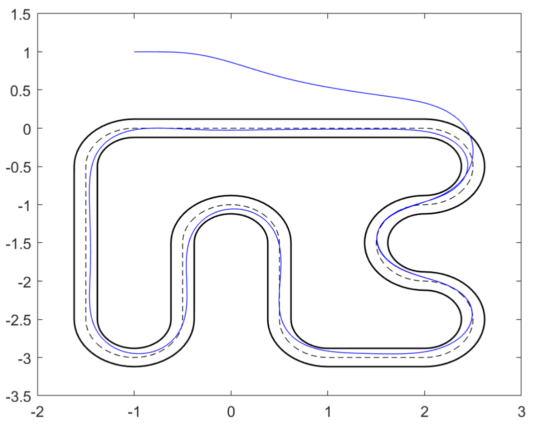
8.2.2 *Dynamic motion planner*

8.3 MPC Controller

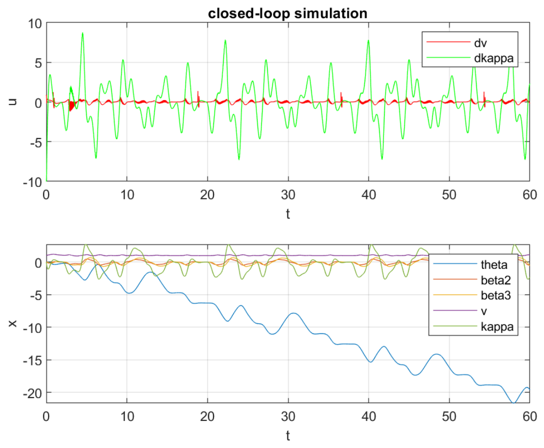
After implementation of the MPC controller in MATLAB, it was evaluated using an example track existing in the acados repository. The track was used to test the MPC controller's ability to send the right steering commands to make the LEGO truck follow a certain reference trajectory in simulation. In this case the reference consisted of a reference curvature displayed by a dashed line in the center of the example track. This example track can be seen to the left in Figure 11 and 12. These tests were made in order to make sure that the truck dynamics were correctly implemented in the model. Two different missions were executed around the track, one for forward driving and one for reverse driving. These missions are described in Section 8.3.1 and 8.3.2.

8.3.1 *Forward driving*

In the forward driving test the LEGO truck's start position was set a bit off relative the race track. The purpose of this initialization was to see whether the truck would approach the reference curvature. The forward driving test is shown in Figure 11a which shows that the truck follows the reference with only small deviations. In Figure 11b the upper plot shows the control signals during the run and the lower figure shows the truck's states except for the position in x and y-direction.



(a) Truck following track driving forward.

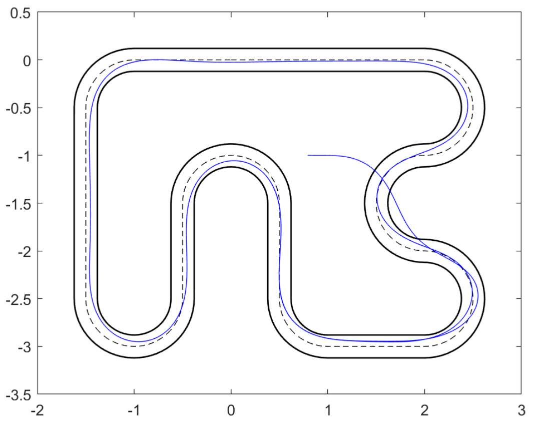


(b) Control signals and the states variation during the run.

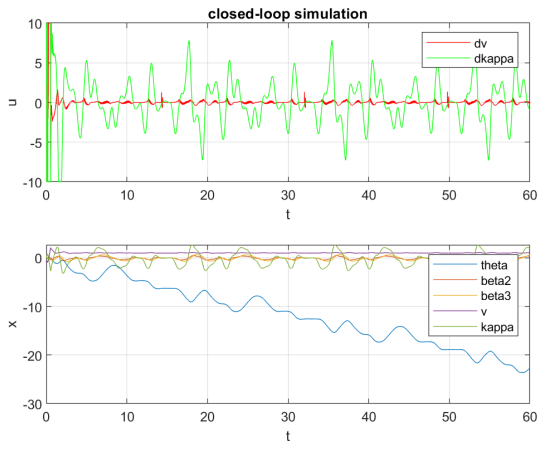
Figure 11: Forward driving test.

8.3.2 *Reverse driving*

The result of the truck reversing around the pre-defined track can be seen in Figure 12. In the left-most picture one can see that the starting position was set to x_3 = 1 and y_3 = -1, this was as with the forwards driving test used to see the behaviour of the truck when it was started with an offset. The goal state was set to zero in both x and y direction. It follows the reference with a satisfying result, however the result could be better if one were to tune the weights of the controller a bit better. The right-most picture showcase how the acceleration and curvature change varied thorugh out the run. It keeps within the constraints formulated in MATLAB. It also shows how the different states varied though out the run.



(a) Truck following track reversing.



(b) Control signals and the states variation during the run.

Figure 12: Revers driving test.

9 FUTURE IMPROVEMENTS

In this section, future improvements are presented.

9.1 IMM-filter

The IMM-filter is working properly in Python with the provided motion models. It has been tested with the obstacle simulator and it works as expected. There are however some fundamental drawbacks with the IMM-filter, that can be improved upon.

- The motion models provided can be improved. The motion models provided can be improved upon to better capture the *true* dynamics of a pedestrian and a ground vehicle. The motion models that are used right now can only capture a limited range of motions. Either more motion models can be used (this might however have an impact on the performance) or the motion models themselves can be improved.
- The IMM-filter have fundamental problems. The IMM-filter is a good solution if the dynamic obstacle is moving slow and steady and without rapid change in direction. For example it works well on aircrafts or big cruise ships. A pedestrian however is usually moving in a more spontaneous manner, and here the IMM-filter might not be the best solution. Also the IMM-filter does not consider static obstacles meaning that it cannot predict that pedestrians will walk around such obstacles and not into them. An alternative approach might be suitable here. There exists a lot of related work, for example [13] or [14] that can be used as inspiration to alternative solutions.
- As the filter is designed now it can only track one target. A natural extension is to make it track and predict the trajectory of several targets at once. If it is clear from which dynamic obstacle we receive the measurements it is just a matter of creating another IMM-filter, which is simple with the current implementation (basically just create a new instance of the IMM-filter class). If it is unclear however from which dynamic obstacle we receive measurements we must consider this. This requires a Joint Probabilistic Data Association Filter (JPDA) [15]. This filter aims to associate data with a certain object.

9.2 POMDP Motion planner

The dynamic motion planner can be improved in multiple ways. Here are some examples.

- Initially, it was intended to extract the full action sequence. It was not possible to implement this successfully in time and thus only the next action is extracted from the tree. The velocity resulting from this action is published on the topic */pomdp/velocity*. The controller *lego_truck_mpc_controller_trajectory* was modified to use this velocity and the planned path as references. However, there exists code in *dynamic_planner_world.cpp* which can create a proper trajectory given that the full action sequence can be extracted.
- The rewards which are specified in *dynamic_planner.cpp* can be updated and refined.
- If the IMM is updated to support multiple obstacles the array *obstacle_state* can be a vector of arrays with one array for each obstacle. This was seen as an option in the beginning of the project but was not implemented due to lack of time.
- The motion planner is currently not tested for reversing. This should be relatively easy to implement as information about direction is received from the static motion planner.

9.3 MPC controller

The MPC controller is currently not working in C++. Both the node and the underlying solver can be built using *catkin_make*. The node is also working with ROS, however the underlying solver contains some errors resulting in the steering commands for the EV3 are not calculated in a satisfying way.

The future work for the MPC controller is hence to make the underlying solver work and test the subscribers and publishers. Making sure that the controller is doing what it is supposed to do.

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] M. Axelsson, J. Barreng, I. Bokne, C. Elf, T. Johansson, A. Sundstedt, and E. Wiman. (2022) Design specification, autonomous truck with a trailer. [Online]. Available: http://www.isy.liu.se/edu/projekt/tsrt10/2022/rev_truck/

[3] AdaCompNUS. (2015) Despot. [Online]. Available: https://github.com/AdaCompNUS/despot

[4] 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

[5] TargetTracking. (2021) Derivation of the imm filter. [Online]. Available: http://www.control.isy.liu.se/student/graduate/TargetTracking/IMMderivation.pdf

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

[7] M. Roth, G. Hendeby, and F. Gustafsson, "Ekf/ukf maneuvering target tracking using coordinated turn models with polar/cartesian velocity," in *17th International Conference on Information Fusion (FUSION)*, 2014, pp. 1–8.

[8] H. Bai, S. Cai, N. Ye, D. Hsu, and W. S. Lee, "Intention-aware online pomdp planning for autonomous driving in a crowd," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 454–460.

[9] L. Pack Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," in *Artificial Intelligence*, 1995, pp. 99–134.

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

[11] 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/

[12] 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

[13] Y. Luo, P. Cai, A. Bera, D. Hsu, W. S. Lee, and D. Manocha, "Porca: Modeling and planning for autonomous driving among many pedestrians," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3418–3425, 2018.

[14] Y. Luo, P. Cai, Y. Lee, and D. Hsu, "Gamma: A general agent motion model for autonomous driving," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 3499–3506, 2022.

[15] Wikipedia. (2022) Joint probabilistic data association filter. [Online]. Available: https://en.wikipedia.org/wiki/Joint_Probabilistic_Data_Association_Filter