Technical Documentation

Multiple Autonomous Vehicles in Complex Scenarios

January 14, 2022

Version 1.1

Project Identity

Group E-mail: teamschannel
Homepage: https://tsrt10.gitlab-pages.liu.se/2021/auto_sys/

Orderer: Erik Frisk
Phone: +46 (0)13 - 28 5714
E-mail: erik.frisk@liu.se

Supervisor: Theodor Westny
Phone: +46 (0)13-28 15 81
E-mail: theodor.westny@liu.se

Course Responsible: Daniel Axehill
Phone: +46 (0)13-28 40 42
E-mail: daniel.axehill@liu.se

Project members

Name	LiU-id	Responsibility
Robert Sehlstedt	robse205	Project leader
Thomas Andersson	thoan459	Design, Scrum-master
Johan Forsman	johfo207	Documents
Philip Sjövik	phisj589	Visualisation
Erik Wahledow	eriwa458	Perception
Ludvig Hansson Granström	ludha385	Software
David Grahn	davgr413	Testing
Mohamed Faraj	mohfa185	Planning & Control

CONTENTS

1 INTRODUCTION

This project, *Multiple Autonomous Vehicles in Complex Scenarios* was performed at Linköping University during the fall of 2021. This document aims to describe the system architecture, implemented algorithms, the acquired results and finally discussions on areas of improvement.

1.1 Parties

There were three parties involved in this project. The orderer was Professor Erik Frisk at Linköping University, the advisor and PhD Theodor Westny at Linköping University and the project group specified in the project identify above.

1.2 Background information

The research on autonomous vehicles has become a popular topic over recent years, engaging academia and industry alike. Although the development has matured to a degree such that several companies already offer fully autonomous taxi services and freight transport solutions, there still exist numerous challenges in the field. In particular, the interaction with other traffic agents is an especially challenging aspect, due to the stochastic behaviour of human drivers.

1.3 Aim

The goal was to deliver a complete solution for a car to perform overtakes safely while taking the risk of oncoming cars in the opposite driving lane into account. This was implemented on a *Cascar*, a modified RC-car platform.

1.4 Definitions

- **ROS** - Robotic Operating System, an open software library which simplifies implementing code for robotic applications.
- **Rviz** - 3D visualization tool for ROS.
- **Visionen** - A test and research arena at Linköping University.
- **QualiSys** - Positioning system used in Visionen.
- **GUI** - Graphical user interface, which allows users to interact with system configurations.
- **EKF** - Extended Kalman Filter.

- **IMU** - Inertial measurement unit.
- **Ego car** - The main car on which the project's algorithms are implemented.
- **Cascar** - A modified remote control car platform equipped with a LIDAR, IMU, Hall wheel sensors, Raspberry Pi, Arduino, and motion capture markers.

2 SYSTEM OVERVIEW

The system consist of the modules shown in Figure 1. The system relies on Qualisys for the measurement of the Cascars. This information and sensor data from the Cascar is filtered by the Perception layer. The Behaviour layer uses this information to make decisions if to overtake other cars or not. The Local path planner generates a path for the Cascar influenced by the Behaviour layer. The controller is responsible for following the planned path. The GUI can be used to give further instructions to the Behaviour layer, like allowing overtakes or stopping all cars. The Visualisation module projects the desired information on the floor in the test area to facilitate validation that the physical Cascar interacts with the environment and the other cars as intended. The communication between the modules is done with ROS (Robotic Operating System), described in [1]).

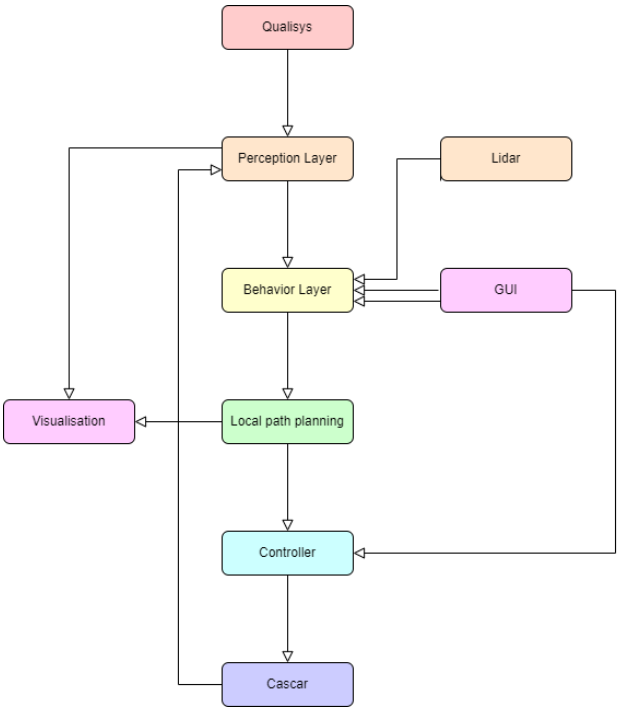


Figure 1: High level system overview.

2.1 Hardware

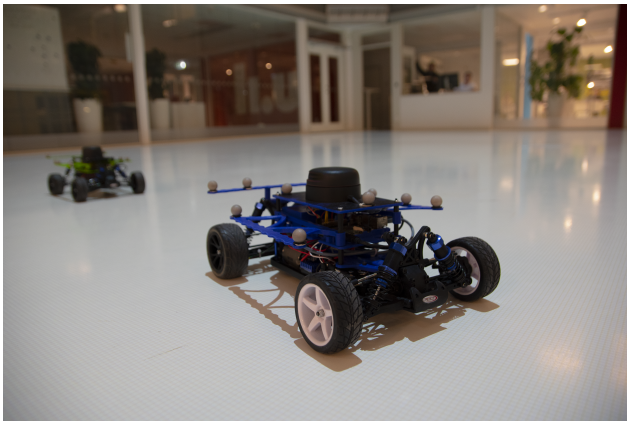


Figure 2: Picture of Cascar, the modified RC-car platform.

The modified RC-car platform referred to as **Cascar**, was equipped with the following items.

- **Rplidar A2 (LIDAR)** - To detect physical objects without motion capture markers [2].
- **Adafruit MPU-6050 (IMU)** - To measure the acceleration of the Cascar and its angular rate [3].
- **Allegro A1120 Hall effect sensors** - To measure the speed of the wheels [4].
- **Raspberry Pi 4** - To be the intermediary agent between ROS and the Arduino [5].
- **Arduino Pro mini** - Sensor data collection and car manoeuvring [6].
- **Motion capture markers** - To allow Qualisys to capture the location and motion of the Cascar [7].

2.2 Qualisys

Qualisys is a localisation and motion capturing system available in a designated area at Linköping Univeristy. Qualisys uses passive marker spheres (small reflective balls) to track objects [7]. This information is published to the vehicle using ROS.

2.3 Simulation

The simulation environment was used to test the functionality of the system during the development process. The simulation environment was also used in tandem with the physical Cascar to simulate other cars. By utilising ROS, the same information that is published to the physical Cascar can be subscribed to by the

simulated car, making the transition from simulation to physical experiments easy. One obvious difference when using simulated cars is that the model is ideal, in comparison to using the physical Cascar. The simulated model is a kinematic single track model with its origin on the center of the rear axle and the x-axis points toward the travel direction. The configuration transitions are represented by Equations (1a), (1b) and (1c) where u_s is the speed, u_Φ steering angle and θ direction of travel [8].

$$\dot{x} = u_s \cos(\theta) \tag{1a}$$

$$\dot{y} = u_s \sin(\theta) \tag{1b}$$

$$\dot{\theta} = \frac{u_s}{L} \tan(u_\Phi) \tag{1c}$$

Another difference is the input to move the car. The motion model in the simulation environment is acceleration based while the physical Cascar is actuated by a PWM (Pulse Width Modulation) signal, which can be thought of as velocity based. One benefit of using ROS is that information about simulated cars can be published, making it possible for the physical Cascar to interact with simulated cars on the same topics.

2.4 The map

The environment built to test the Cascar and its functionality, was a circular road with two lanes. A circular road facilitates testing since the Cascar can loop indefinitely and fewer restarts are necessary. An illustration of the map can be seen in Figure 3. The radius of the circle dividing the two lanes was set to 3 [m] and the width of each lane was set to 0.4 [m].

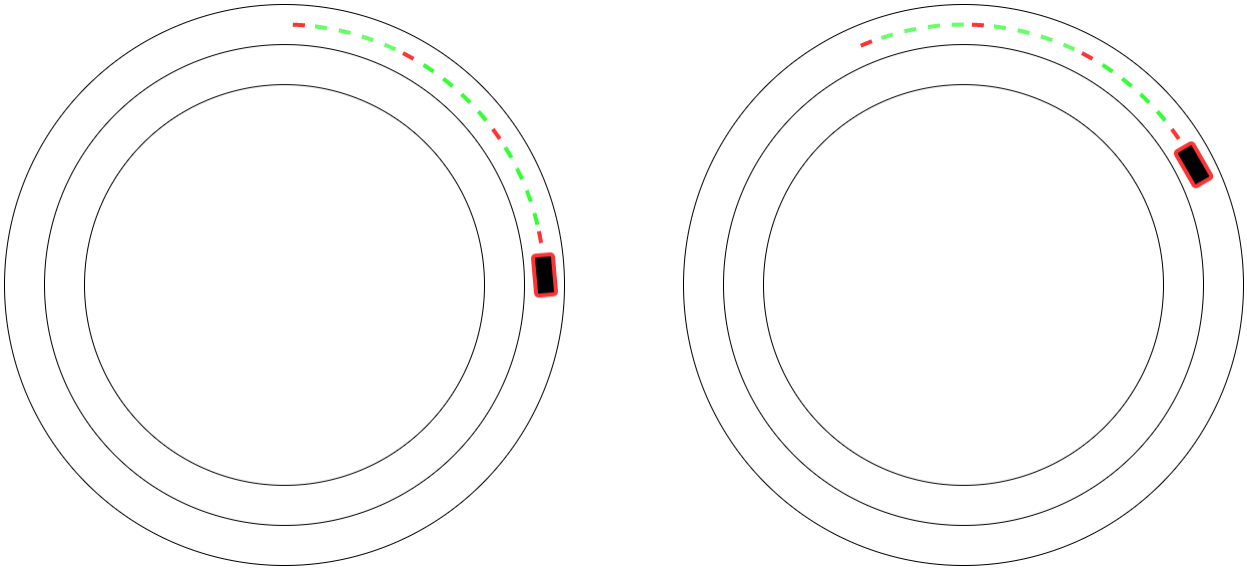


Figure 3: Illustration of the circle segments that make up the planned path at two different time steps and how they are moved forward. Red dots indicate the start and end of an segment. The rectangle represents the car.

3 EXTENDED KALMAN FILTER

To estimate position, velocity and orientation of the physical cars, an Extended Kalman Filter (EKF) [9] was used. The EKF was chosen as there is a non-linearity in the motion model, see the sine and cosine terms in Equation (2a). This non-linearity is well described by its Taylor series. As such, more advanced filters such as Particle Density Filters were not investigated. This section aims to describe the EKF in detail. The Extended Kalman Filter is updated every 0.1 seconds. This time interval was obtained after a tuning process where a compromise between the accuracy of the state estimation and computation time. Since the car's maximum speed is around 1 m/s and we know intuitively that it can't manoeuvre too much on a travelled distance of 10 cm (0.1 s $\cdot$ 1 m/s), a time step of 1.0 s seems sound. Each sensor has its own measurement update. As there are different measurement frequencies for each sensor the latest sensor data is taken when the update is done.

3.1 Motion model

An IMU is used as an input to the EKF, hence a modified constant polar velocity model [10] was used as presented in Equation (2a) which simplifies the motion model (1).

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{v} \\ \dot{\theta} \\ \dot{\beta} \end{pmatrix} = \begin{pmatrix} v\cos(\theta) \\ v\sin(\theta) \\ 0 \\ (u-\beta) \\ 0 \end{pmatrix} + \begin{pmatrix} \cos(\theta)T_s & 0 \\ \sin(\theta)T_s & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \tag{2a}$$

$$f(X) = \begin{pmatrix} v\cos(\theta) \\ v\sin(\theta) \\ 0 \\ (u-\beta) \\ 0 \end{pmatrix} \tag{2b}$$

Where w is white Gaussian process noise for $\begin{bmatrix} v, & \theta \end{bmatrix}$ with covariance matrix $Q = \begin{pmatrix} 0.1 & 0 \\ 0 & 0.2 \end{pmatrix}$. The covariance matrix was obtained by tuning the system manually, comparing the estimated states with the indoor positioning system Qualisys. All variables can be seen in Table 1

Table 1: Variables in the Extended Kalman Filter.

x	position along x-axis
y	position along y-axis
v	speed
θ	yaw
T_s	sample period
u	measured yaw rate
β	bias in yaw rate measurements

Here, u is seen as an input signal. The trigonometric terms are the reason an Extended Kalman Filter is required as they are non-linear functions of the state. The jacobian of the transition function $f(X)$ is needed in the EKF, see Subsection 3.2. The time step T_s is set to 0.1 seconds. The state vector X (with time indices omitted) is defined in Equation (3).

$$X = \begin{pmatrix} x & y & v & \theta & \beta \end{pmatrix}^T \tag{3}$$

3.2 Jacobian

The jacobian of the transition function $f(X)$ in Equation (2b), referred to as the A-matrix in [9], has to be evaluated for the latest state estimate in every time step in order to update the estimated state covariance P. The resulting matrix can be seen in Equation (4).

$$f'(X) = \begin{pmatrix} 0 & 0 & \cos(\hat{\theta}) & -\sin(\hat{\theta})\hat{v} & 0 \\ 0 & 0 & \sin(\hat{\theta}) & \cos(\hat{\theta})\hat{v} & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{4}$$

Where $\hat{\theta}$ and $\hat{v}$ are the estimated yaw and speed from the state obtained in the last update.

3.3 Discretisation

As the algorithm will be run in iterations on a computer it is necessary to discretise the motion model. Two types of discertisations were used. Equation (4) was discretised using the Euler Forward method and Equation (5) was obtained.

$$f'(X_{k|k}) = \begin{pmatrix} 1 & 0 & \cos(\hat{\theta})T_s & -\sin(\hat{\theta})T_s\hat{v} & 0 \\ 0 & 1 & \sin(\hat{\theta})T_s & \cos(\hat{\theta})T_s\hat{v} & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -T_s \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{5}$$

$f(x)$ was discretised using the Runge-Kutta method [11]. It is more computationally heavy but also a more accurate discretisation method. Equations (6b) to (6e) describe the obtained model.

$$x_{k+1} = x_k + \frac{T_s}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{6a}$$

$$k_1 = f(x, u) \tag{6b}$$

$$k_2 = f(x + \frac{T_s}{2}k_1, u) \tag{6c}$$

$$k_3 = f(x + \frac{T_s}{2}k_2, u) \tag{6d}$$

$$k_4 = f(x + T_s k_3, u) \tag{6e}$$

Where $f(x, u)$ is the continuous equation that is being discretised, the first argument is the state vector and the second is the input signal.

3.4 Measurement model

The measurements used in the Extended Kalman Filter are from Qualisys which measures position, velocity and yaw, an IMU to measure acceleration and yaw rate (again, yaw rate is considered an input), as well as wheel rotation frequency to measure speed. Qualisys is a motion capture 3D positioning system that utilises cameras to detect markers on the car.

3.4.1 *Qualisys update*

Qualisys provides angles representation in quaternions. These are transformed into Euler angles and only the yaw is used. It also provides translational velocity in x- and y-directions. It is important to note that the directions of x and y depends on how Qualisys is calibrated. The velocity is transformed into speed by taking the 2-norm of the velocity. Finally, it also provides x- and y-position which are taken as is. After these transforms the measurement $y_{qualisys}$ is assumed to be related to the state X and Gaussian noise $v_{qualisys}$ according to Equation (7).

$$y_{qualisys} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} X + v_{qualisys}, \ \mathrm{Cov}(v_{qualisys}) = \begin{pmatrix} 0.1 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{7}$$

A problem with Qualisys is that it has blind spots at some locations, where no measurements or (even worse) potentially very bad measurements are acquired.

3.4.2 *Speed update*

There are evenly spaced magnets in the wheels, and by measuring the time it takes between two magnets to pass a certain point one can measure the frequency with which the wheel spins. With the radius of the wheels these can be translated into a speed measurement. Note that if the car is driving at higher speeds there will be a higher measurement frequency. When the car is standing still, there are no measurements. The conversion of the rate of the magnets and the time comparison is not done in the Extended Kalman Filter but in the microprocessor that reads the sensor data. The measurement y_{speed} is assumed to be related to the state X and Gaussian noise v_{speed} according to Equation (8).

$$y_{sensor} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \end{pmatrix} X + v_{sensor}, \ \mathrm{Cov}(v_{sensor}) = 0.1 \tag{8}$$

3.4.3 *IMU update*

As previously mentioned previously, the yaw rate from the IMU (excluding its bias) is taken as is and used in the process model and time update. The IMU also provides acceleration, but the data is inaccurate and cannot be used to estimate acceleration. However, when the car is standing still it provides oscillatory measurements with low amplitude. When the car is moving, it provides oscillatory measurements, but with

a higher amplitude. An algorithm has been developed that utilises this to provide a measurement of the speed, i.e $v = 0$ when the accelerometer provides oscillations with amplitude lower than a certain threshold. The measurement y_{IMU} is related to X according to Equation (9). To be able to use a normal Kalman Filter update we also add some Gaussian noise v_{IMU}. Note that this measurement update only takes place if the speed is 0 m/s.

$$y_{IMU} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \end{pmatrix} X + v_{IMU}, \ \mathrm{Cov}(v_{IMU}) = 0.01 \tag{9}$$

3.5 Outlier rejection

If a measurement for some reason is deemed bad, it should be rejected. This is done by using outlier rejection as described in [9]. A test function is defined according to Equation 10.

$$T(y_k) = (y_k - H_k \hat{x}_{k|k-1})^T (H_k P_{k|k-1} H_k^T + R_k)^{-1} (y_k - H_k \hat{x}_{k|k-1}) \tag{10}$$

If $T(y_k) > \chi^2_{\alpha,n_y}$ the measurement is rejected. χ^2 is the chi-square test. This is a tuning parameter that corresponds to the likelihood that a measurement is valid. If for example four measurements are used and the measurement should be within a 95% interval the chi-square test with four degrees of freedom becomes $\chi^2 = 9.49$. [12]

3.6 The algorithm

The full name of the algorithm is DARE-based EKF1 and can be seen below. The measurement update is implemented in three ways, one for each measurement and the general structure can be seen in eqs. (11b) to (11e).

$$S_k = R + h'(\hat{X}_{k|k-1}) P_{k|k-1} (h'(\hat{X}_{k|k-1}))^T \tag{11a}$$
$$K_k = P_{k|k-1} (h'(\hat{X}_{k|k-1}))^T S_k^{-1} \tag{11b}$$
$$\epsilon_k = y_k - h(\hat{X}_{k|k-1}) \tag{11c}$$
$$\hat{X}_{k|k} = \hat{X}_{k|k-1} + K_k \epsilon_k \tag{11d}$$
$$P_{k|k} = P_{k|k-1} - P_{k|k-1} (h'(\hat{X}_{k|k-1}))^T S_k^{-1} h'(\hat{X}_{k|k-1}) P_{k|k-1} \tag{11e}$$

Where $h'(\hat{X}_{k|k-1})$ is simply the matrix in front of each X in Equations (7) to (9). R is the covariance for Equations (7) to (9), in practice R is a tuning parameter. The algorithm was initiated when Qualisys sent the first message. These states combined with setting the bias to zero were used as X_0. The initial value of P, P_0, was a diagonal matrix (independent uncertainties) according to Equation (12).

$$P_0 = \begin{pmatrix} 0.01 & 0 & 0 & 0 & 0 \\ 0 & 0.01 & 0 & 0 & 0 \\ 0 & 0 & 0.01 & 0 & 0 \\ 0 & 0 & 0 & 0.01 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{12}$$

The values in P_0 were chosen like this because the states measured by Qualisys is fairly accurate but the gyro bias is still unknown and needs a larger uncertainty. The time update part was described by Equations (13)

$$\hat{x}_{k+1|k} = f(\hat{x}_{k|k}) \tag{13a}$$
$$P_{k+1|k} = Q + f'(\hat{x}_{k|k}) P_{k|k} (f'(\hat{x}_{k|k}))^T \tag{13b}$$

See Equations (4) and (5).

3.7 Notes on implementation

The algorithm for the accelerometer is done separately in a C++ script. The rest of the algorithm is made in a Python script. The subscribed ROS topics can be seen below.

Subscribed topics - state_estimator_node

Topics	Message type	Description
/qualisys/car_name/odom	nav_msgs/Odometry	Contains position, pose and velocity of the car (from qualisys).
/sensor/imu	CarMeasurement	Custom message containing data from sensors on the Cascars. For the yaw rate.
/sensor	CarMeasurement	Contains the wheel speed.
/isMoving	std_msgs/Bool	Contains bool with value False if the car is standing still

Published topic - state_estimator_node

Topics	Message type	Description
odom	nav_msgs/Odometry	Contains position, pose and velocity of the car.

Subscribed topic - MoveOrNot_estimator_node

Topics	Message type	Description
/sensor/imu	CarMeasurement	Used for obtaining acceleration

Published topic - MoveOrNot_estimator_node

Topics	Message type	Description
/isMoving	std_msgs/Bool	Contians bool with value False if the car is standing still

3.8 Performance

Inn Figure 4 and 5 we see the performance of the EKF on two intentionally challenging data sets. In the figures, blue indicates the measurements from Qualisys and green the estimated positions by the EKF. The red X is the starting position and the green X is the end position. We see that the EKF estimates the positions very well even where there are periods of missing Qualisys measurements.

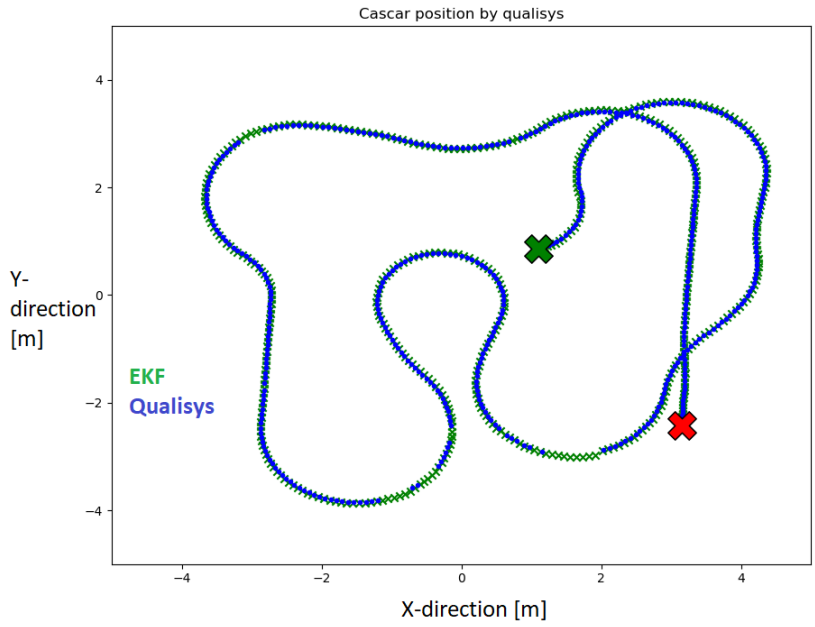


Figure 4: Performance on the first data set.

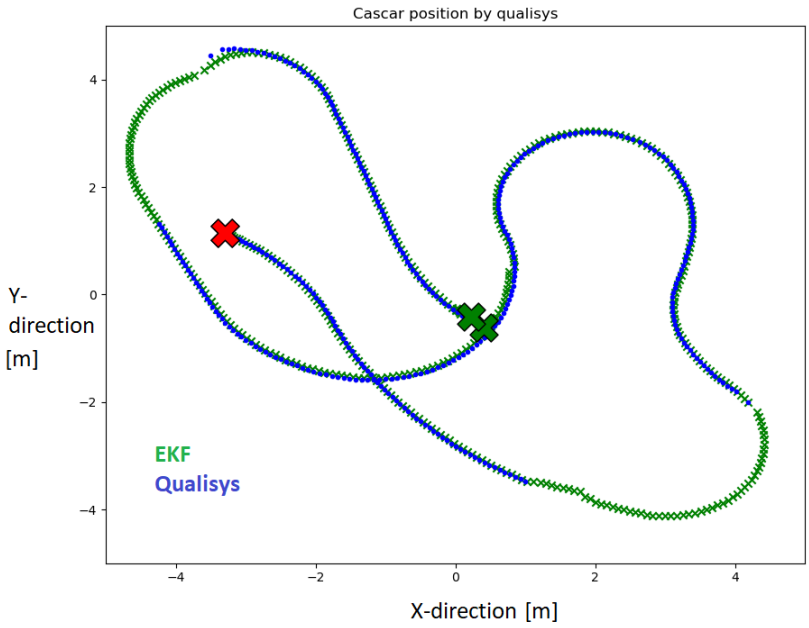


Figure 5: Performance on the second data set.

4 PLANNER

The planner consists of a behaviour layer and local planner. Due to the constraints and attributes of a circular road where the goal is to traverse this circle indefinitely, there is no need for global planning. Otherwise, the decision-making hierarchy used in ego car is based on and very similar to the typical hierarchy presented in [13], see Figure 6. With the help of the car's currently planned route and information about its surroundings, decisions are made in the behaviour layer about how the car should behave, and based on these decisions, the car's motions are performed, which include speed adjustment and lane changes. Decisions about lane changes are sent to the local path planner, which gives the ego vehicle a path to follow and the speed is sent directly to the controller. For these desired motions to be fulfilled, both speed and path tracking are controlled and this part is discussed in more detail in the next section, *Controller*. This section aim to describe the planner in detail.

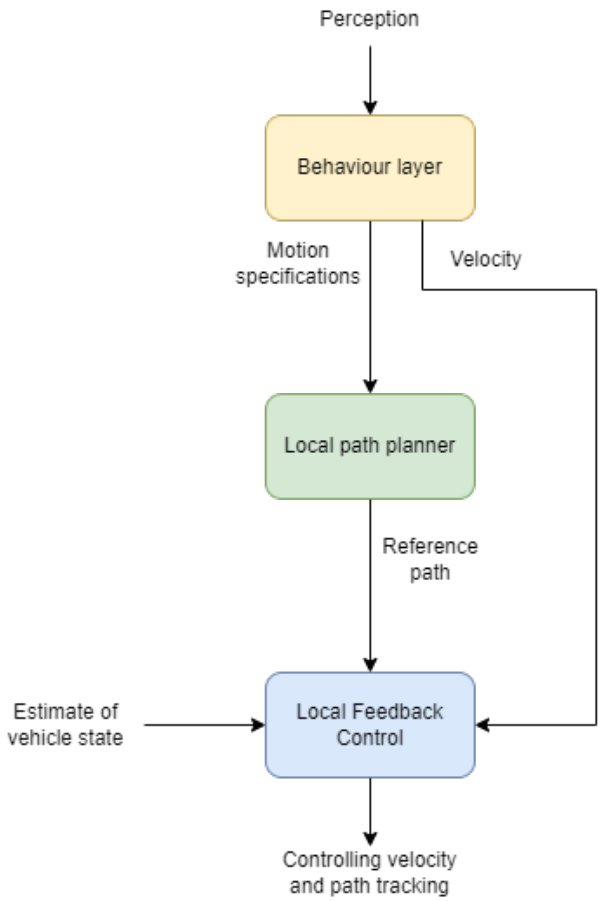


Figure 6: An overview of the decision-making hierarchy for ego vehicle adapted from [13].

4.1 Behaviour layer

The behaviour layer handles the decision making part of the system, dictating what path to follow, making emergency stops based on LiDAR measurements and having final say for overtaking. Two versions of the behaviour layer is available depending on user preference. The user may select if the cars should just traverse the map without overtaking or engage in overtakes. This is to make it easier to test the system when there only one car performing overtakes.

How the ego car decides whether an overtaking is appropriate is presented in the decision-structure in Figure 7. The first thing that the ego car considers is the speed difference and the distance to the car ahead, i.e., the ego car will overtake if the preceding car drives slower and is closer than a pre-defined reference. Based on tests, it has been decided that the ego car should to overtake if it drives 20% faster than the car in front with less than two meters distance, otherwise it should follow the car in front. This was primarily because of the constraints of a small circular map. An overtake takes a long time if the cars have almost the same speed. Before overtaking, the ego car must also consider whether a potential collision with oncoming traffic could occur or not. If not, the car decides to overtake.

To know if an overtake is possible with an oncoming car, the time to drive to the next gap is compared the time until a collision between the ego car and the overtaking car occurs. The relation to determine this is shown in Equation (14). The right side of the equation represents the time to the closest gap based on the distance to the gap, the requested speed of the ego car, and the speed of the car being followed. The left side of the equation represents the time to a collision based on the distance to the oncoming car, the requested speed of the ego car, and the speed of the oncoming car. The details of the distance calculations is further detailed in Section 5.5. The offset is manually set as a safety margin and the account for the fact that the time to actually turn left and right is not calculated. If the overtaking can be done faster than the time it takes for the cars to collide, the ego car should perform an overtaking. Otherwise, the ego car should follow the car head to find the next opportunity to perform an overtaking. In the current implementation the car cannot abort the overtake attempt by braking and falling back into the old gap or by quickly finding another gap between two of the cars the ego car is overtaking and squeezing between them.

$$\frac{d_{gap}}{v_{requested} - v_{following}} < \frac{d_{oncoming}}{v_{requested} + v_{oncoming}} - offset \tag{14}$$

Variable	Description
$v_{requested}$ **[m/s]**	Requested velocity
$v_{following}$ **[m/s]**	Velocity of car ahead
$v_{oncoming}$ **[m/s]**	Velocity of oncoming car
d_{gap} **[m]**	Distance to the found gap
$d_{oncoming}$ **[m]**	Distance to the oncoming car

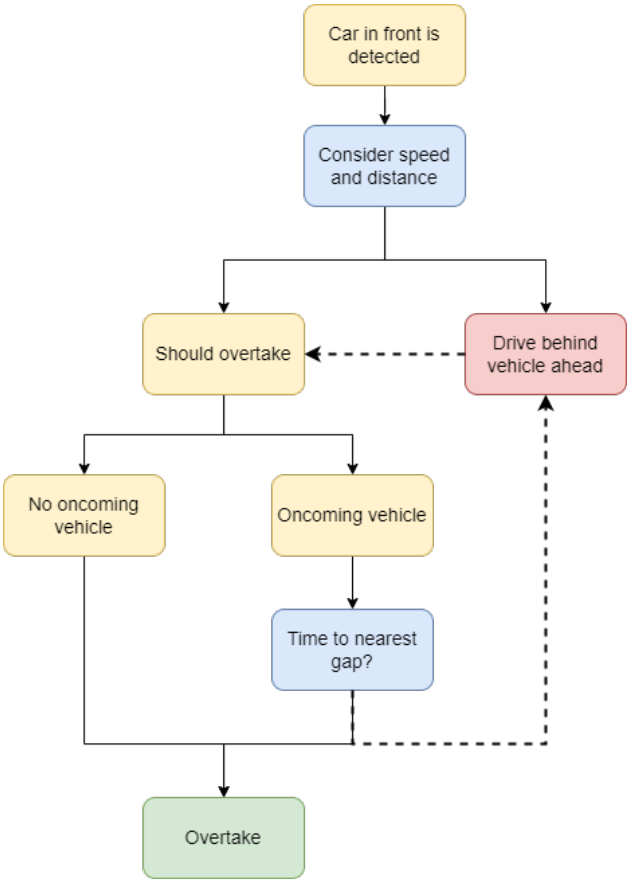


Figure 7: Decision structure of the autonomous vehicle.

During the overtaking, the ego car needs to decide when to drive back to the right lane. It does this by ensuring that it is in front of the car it is overtaking, which it does when the dot product between the other car's speed vector, v and the distance vector, d is positive, see Figure 8. In addition, the distance between the cars needs to be at least 1.5 m, which is based on tests, and no other car within a certain radius may be in the target lane.

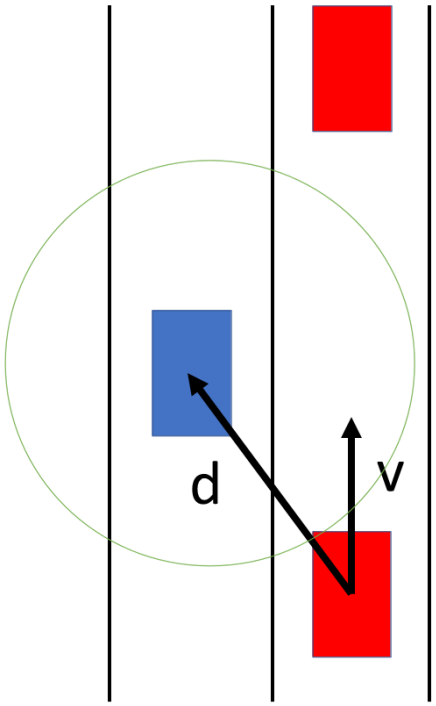


Figure 8: Lane-change back to the right lane. The ego vehicle shown in blue and surrounding vehicles in red.

4.2 Local path planner

The local planner is constructed as segments of the circular road that sequentially moves forward, as the ego vehicle traverses the path. This is illustrated in Figure 3 in section 2.4. Both the number of segments that make up a path, and their respective lengths may be changed according to user preference. Overtaking other cars is performed by fixed lane changes. The segments are moved forward just as they normally would as the ego vehicle traverses but placed in the other lane instead. To create the path from the current position to the lane segments in the other lane, a fixed straight line is computed to merge the current position of the ego vehicle to the new segment. This can be seen in Figure 9. The segments updates indefinitely as the car traverses, resulting in the car looping around the road for as long as desired.

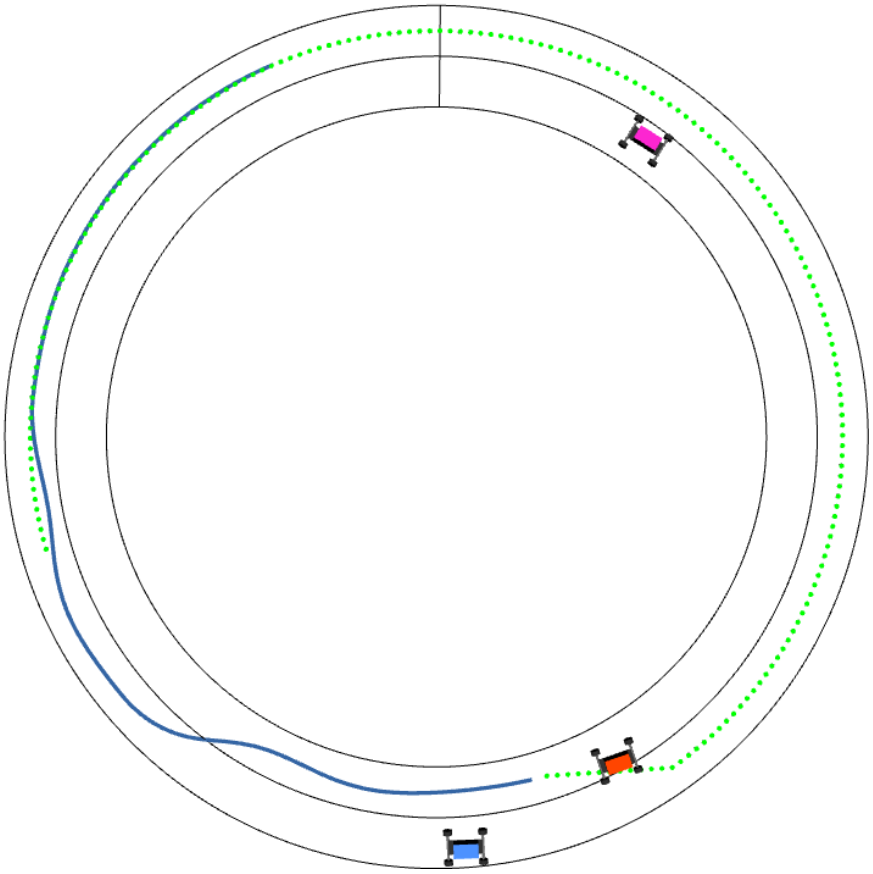


Figure 9: Snapshot of simulation where the green dotted line are the segments including lane changes. The red car is the ego vehicle and the blue line is its traversed path. The blue and purple cars are simulated cars.

4.3 ROS - Behaviour layer

The behaviour layer package consists of two nodes, a *determine_overtake_node* and a *behaviour_layer_node*. In *determine_overtake_node*, decisions about lane changes are made and further published to *behaviour_layer_node* that integrates these decisions with other behavioural parts, such as Lidar stop and GUI configurations.

Published topics - determine_overtake_node

Topics	Message type	Description
/change_lane	std_msgs/Bool	Contains decisions about lane changes.
/decision_msg	std_msgs/String	Contains status update on all decisions made.

Subscribed topics - determine_overtake_node

Topics	Message type	Description
/position_list	find_ahead/position_list	Contains positions of surrounding cars.
/velocity_list	find_ahead/velocity_list	Contains velocities of surrounding cars.
/car0/odom	nav_msgs/Odometry	Position and velocity of ego car.
/follow_goal	find_ahead/follow_goal	If there is a car ahead, gives the cars index, distance to it and its velocity.
/opposing_car	find_ahead/follow_goal	If there is a opposing car, gives the cars index, distance to it and its velocity.
/gap_exist	find_ahead/gap_exist	Provides information on whether a gap exists, and if so, the distance to it.

Published topics - behaviour_layer_node

Topics	Message type	Description
global_path	nav_msgs/Odometry	The path the car follow at any given time, made up of a series of path segments.
/lidar_stop	std_msgs/Bool	If the LIDAR detectes an obstacle in front of the car, disable the car.
dir	std_msgs/Int8	What direction a certain car is moving in.
park_goal	geometry_msgs/Point	Unused, previously used for a mini-project.

Subscribed topics - behaviour_layer_node

Topics	Message type	Description
/change_lane	std_msgs/Bool	Contains decisions about lane changes.
/allow_overtake	std_msgs/Bool	Is overtaking allowed from the GUI.
odom	nav_msgs/Odometry	Odometry information from the cars.

5 CONTROLLER

To be able to follow the planned path, a controller had to be implemented. The controller is decoupled into a longitudinal controller and a lateral controller.

5.1 Longitudinal controller

The longitudinal controller is a platooning PD controller [14] as described by Equation (15)

$$u = -K_p \cdot e - K_d \cdot \dot{e} \tag{15}$$

The output from the PD equation u is the requested acceleration, e and $\dot{e}$ is the distance error and relative velocity error to the car in front. K_p and K_d are parameters used to tune the response of the controller. When no car is detected in front or the car is not using intelligent control, a simple velocity to acceleration controller is used to keep the requested speed instead.

Design variable	Value
K_p	1
K_d	1

5.2 Lateral controller

For lateral control, a pure pursuit controller was used [15]. It was chosen due to its simplicity and popularity in the literature [13]. The controller targets a point ahead on the planned path and outputs a steering angle δ that steers the car to toward the goal target. The point is selected as the furthest point inside a lookahead-horizon l, which was tuned for the map and (physical-) cars. Increasing the look-ahead parameter l makes the cars converge more gradually and with less oscillation. The response of a pure pursuit controller looks similar to the step response of a second order system where the look-ahead l can be compared to a damping factor [15]. This behaviour is illustrated in Figure 10.

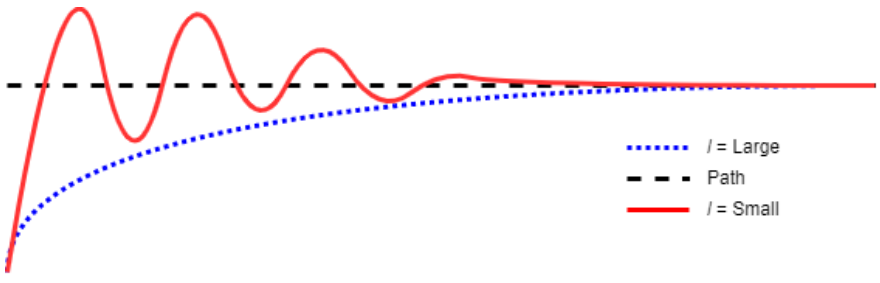


Figure 10: Illustration of the response from a pure pursuit controller. The red line corresponds to a small look-ahead. The blue dotted line corresponds to a large look-ahead.

The simulated cars were more robust to altering the horizon compared to the physical cars. This might be due to the steering mechanism of the physical car. In the physical car, the maximum steering angle was measured manually and tuned to be as symmetrical as possible. However this tuning might have suffered from human error, leading to a lesser robustness fro the physical car. The steering angle δ is calculated as in Equation (16) and an illustration is provided by Figure 11 [16].

$$\delta = \arctan\left(-L\frac{2x}{l^2}\right) \tag{16}$$

Variable	Description
x **[m]**	Displacement from the path
δ **[rad]**	Steering angle
L **[m]**	Length of the car
l **[m]**	Look-ahead horizon

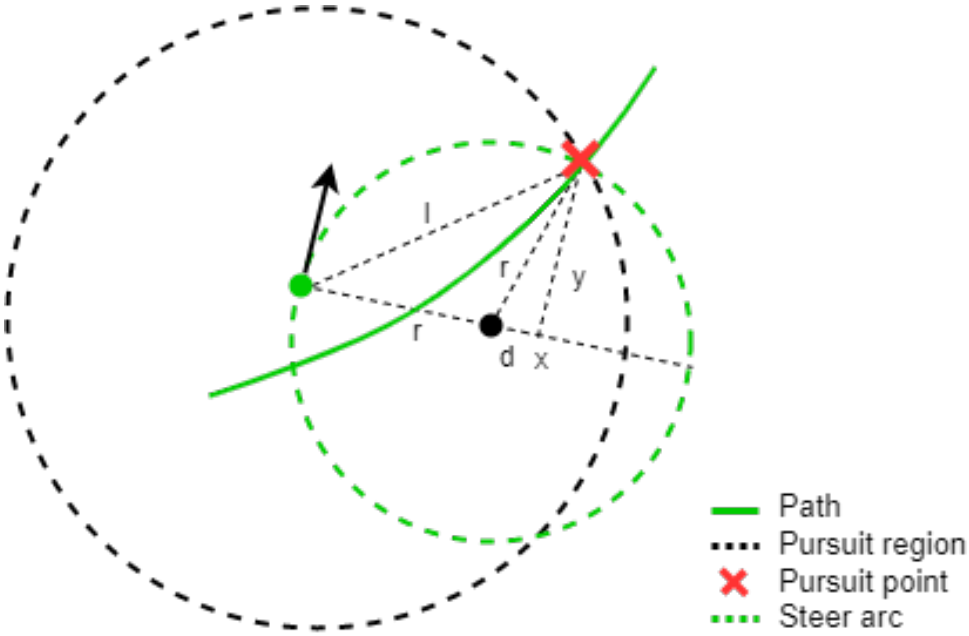


Figure 11: Pure pursuit controller and required geometry for calculations, based on geometry presented in [16].

There are also implementations of a PID-controller, which steers based on the difference in angle of track and car. The wanted lateral controller can be chosen and altered live through the GUI extension.

5.3 Performance

The controllers perform well and the car manages to follow the desired path and the desired speed and distance to cars ahead, if there are any. Figure 12 shows the absolute displacement error from the path for the physical car. Figure 13 shows the absolute displacement error from the path for a simulated car. The peaks in the plots is due to a bug in the local path planner where the first segment in the road is moved too far, leading to a large displacement even though the car is in the middle of the lane. The data was recorded without overtakes and the large displacement in the beginning in Figure 13 is due to the initial position of the car. The displacement is calculated by projecting onto the discretised path, explaining the behaviour in the plots. In reality the error rate would be smoother if the displacement was calculated projecting onto a continuous path.

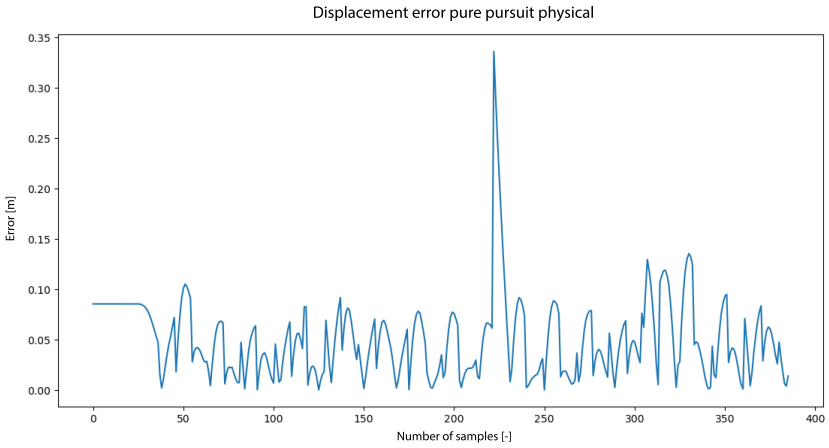


Figure 12: Absolute displacement error from the path with physical car.

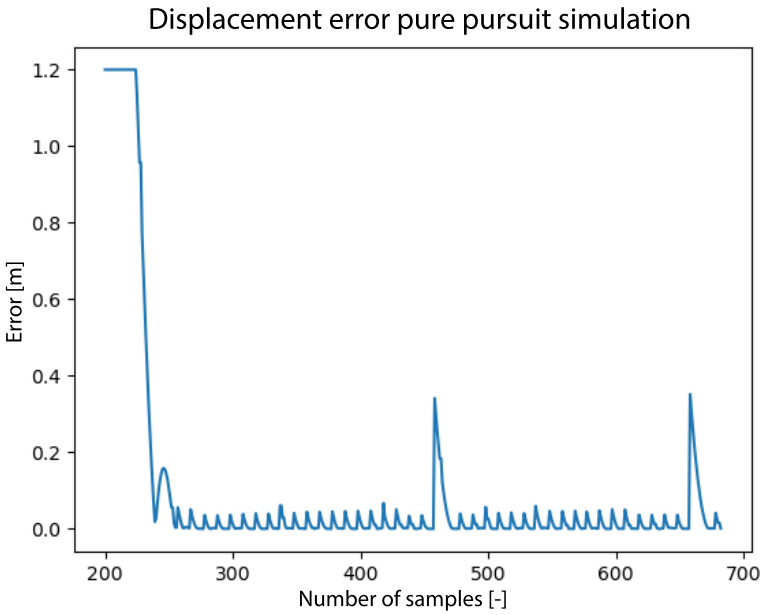


Figure 13: Absolute displacement error from the path with simulated car.

The controller deviates a bit while overtaking. It might sound counter-intuitive, but since the lane changes is planned using a fixed straight path to the other lane and to achieve a smooth lane change, the car has to deviate a bit from the path. This means that the slight deviation from the path while changing lanes is expected and desired. With a feasible curved path to other lanes, the deviation could potentially be decreased. The difference of a straight path and a more feasible curved path can be seen in Figure 14.

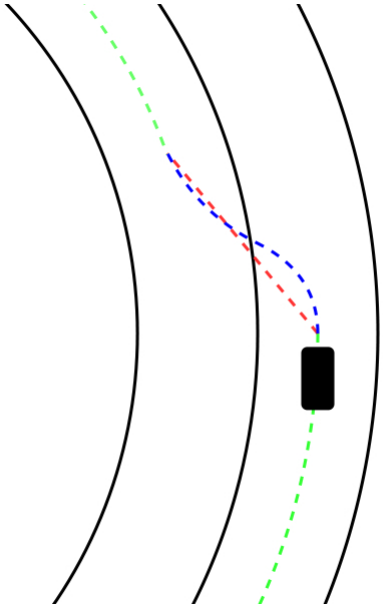


Figure 14: Illustration of two planed lane changes. The red corresponds to a straight path, similar to the one implemented. The blue corresponds to a more feasible curved path.

The performance of the longitudinal controllers are presented in Figure 15 and 16.

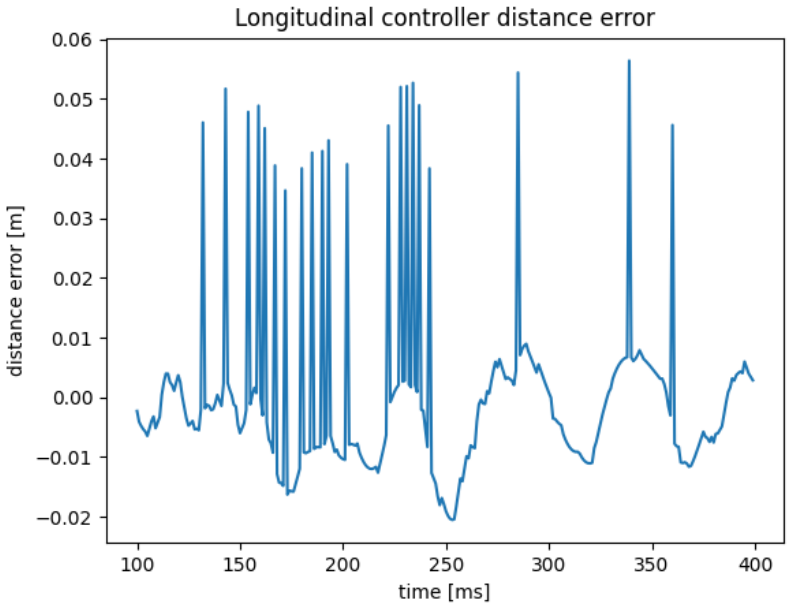


Figure 15: The simulated distance error e when the ego car is following another car. The spikes in the distance error is most likely due to noise in the distance finding function.

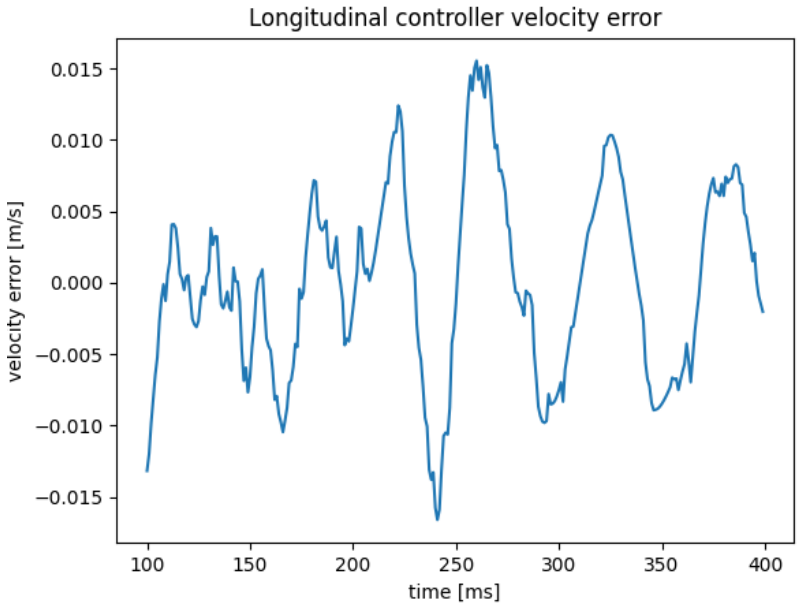


Figure 16: The simulated velocity error $\dot{e}$ when the ego car is following another car.

5.4 Implementation notes

The above mentioned controllers are implemented in Python and their topics are presented below.

Published topics

Topics	Message type	Description
control	cascar_utilities/ ControlRequest	Velocity and steer commands to the simulated cars.
car_command	cascar_utilities/ CarCommand	Velocity and steer commands to the physical cars.

Subscribed topics

Topics	Message type	Description
/change_lane	std_msgs/Bool	Contains decisions about lane changes.
/allow_overtake	std_msgs/Bool	Is overtaking allowed from the GUI.
odom	nav_msgs/Odometry	Odometry information from the cars.
move_base_simple/goal	geometry_msgs/ PoseStamped	A continuously updated x, y coordinate from the local planner.
points2d	cascar_utilities/ Points2D	Points list of the planned path.
controller_setting	std_msgs/Int8	Chosen controller setting from the GUI.
/lidar_stop	std_msgs/Bool	If the LIDAR detectes anobstacle in front of thecar, disable the car.
follow_goal	find_ahead/follow_goal	If there is a car ahead, gives the cars index, distance to it and its velocity.
enable	std_msgs/Bool	Whether the cars are allowed to move or not (enabled/disabled). Published from both the GUI as a setting and from the Lidar in case of collision.

5.5 Detection of other cars

To facilitate safe following and overtaking, a method to detect cars in front was needed. By utilising the path points already laid out for the car to follow one can detect cars in the same or opposite lane as the ego car. A circular region around a selection of path points is checked to determine if another car shares that space. This can then be used to find targets to look out for or gaps for overtaking into. For now the direction i.e. clockwise or counter-clockwise circulation around the track is assumed to be known.

6 VELOCITY MODEL FOR CASCAR

The speed of the car is determined by a PWM-signal (pulse-width modulated signal) which determines the average voltage from the battery to the motor. For several reasons, one being consistency between simulation and physical driving, it is advantageous to request an acceleration and use a model that converts the acceleration into a PWM-signal. This section aims to describe how such a model was derived and implemented.

The model was derived in Matlab using the *System Identification* toolbox [17]. This required data from PWM-signal to velocity on the vehicle, and was logged during several repeated tests. The input signal (PWM-signal) was constructed of different step responses as seen in Figure 17, also showing the corresponding velocity. Each step was long enough for the speed to stabilise. Using the toolbox, different models

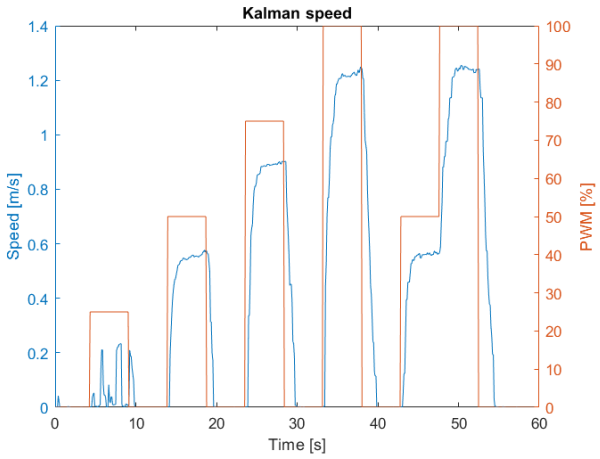


Figure 17: Speed related to PWM during the time steps.

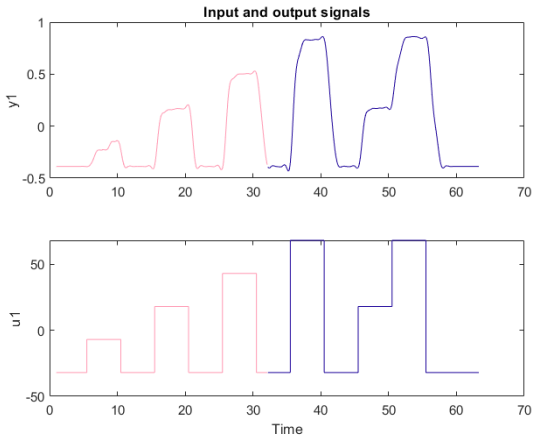
were compared as described in [18]. The best model had a fit of 80% and was an ARX311 model. The model fit can be seen in Figure 18b. Note that the signal has reprocessed by removing means and trends, and can be seen in Figure 18a (as seen in Figure 17 the speed was positive during the whole run).

The implemented model descends from an inverse of the ARX311-model which relates PWM to velocity. As it is a causal relationship from PWM to velocity there is a need to know the velocity in the future. This is solved by sending a requested acceleration (which is obtained from the controller) and then using Euler forward. The model can be seen in Equation (17) and (18).

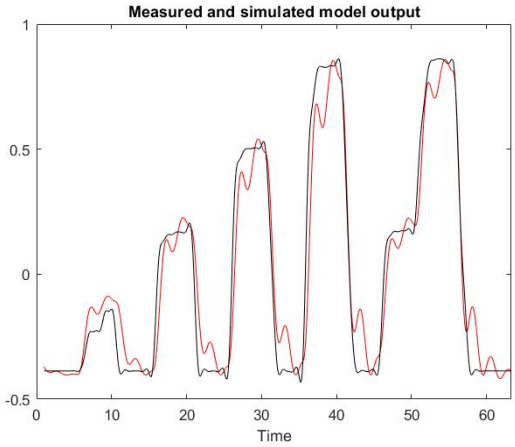
$$u_t = \frac{1}{b_1}(c_1 y_{t-1} + c_2 y_t + c_3 y_{t+1}) \tag{17}$$

$$y_{t+1} = y_t + aT_s \tag{18}$$

Where c_1, c_2, c_3, b_1 are constants determined by the identification, y_t is measured velocity at time t, a is requested acceleration, T is sample time and u_t is the resulting PWM-signal. The model constants are



(a) The models output compared to real output.



(b) The models output compared to real output.

Figure 18: Signals and derived model output from *System Identification*.

identified for one of the physical cars but has been tested and verified to work on another. The values of the constants can be seen in Table 2.

Table 2: Constants in cascar model.

c_1	1
c_2	-1.8170
c_3	0.8375
b_1	$\frac{2.4145}{10000}$
T_s	0.1

7 ROS STRUCTURE

Certain parts of the code are supposed to run once per active vehicle. Things like the behaviour layer, the controller, the state estimator and (for simulated cars) the simulator must run in several independent threads and be associated with its respective car. This is solved with a structure of ROS launch files.

On the lowest level, there are two launch files called *simulated_car.launch* and *physical_car.launch*. These files start all the ROS nodes that are needed for a single car to be able to run. They are in turn launched from either *launch_all_simulated_cars.launch* or *launch_all_physical_cars.launch*.

An important thing happens in the *launch_all_xxx_cars* files, namely that all cars are launched in their own unique namespace. This allows the code that concerns a single car to be written as if there only were one car. And when the code is launched multiple times in different namespaces, each ROS subscriber and publisher will automatically find the ROS topic that is located on the correct namespace.

On the top of the structure is the file *init.launch*. This is the only file that the end user has to care about, and it is launched with parameters that can decide how many physical and simulated cars to have and if the ego car should be physical or simulated.

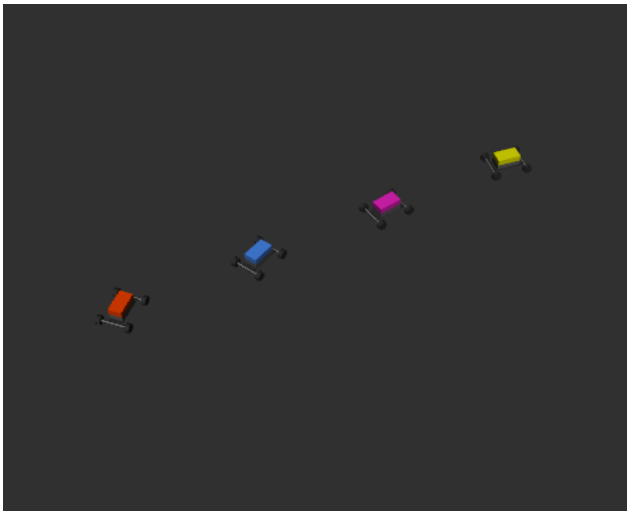


Figure 19: Multiple 3D models of the vehicles rendered in Rviz.

8 VISUALISATION

The visualisation is done in both the ROS package Rviz and in a Matlab plot window. Additionally certain functionality such as starting and stopping all vehicles, and displaying decision making of the ego vehicle is done through a graphical interface.

8.1 Rviz

The map and both physical and simulated cars are displayed in a 3D window, see Figure 19 and 9. The planned and travelled path of the ego vehicle is also displayed, as well as the environment. The map and paths are published topics that Rviz subscribe to. The vehicles are described in one Xacro file that in turn generates one URDF (Universal Robot Description Format) model file per car.

8.2 Matlab

The Matlab plot is projected on the floor in Visionen. The plot contains the map and is updated regularly with the position of every simulated car and the paths of the physical ego vehicle, see Figure 20. This is done by creating subscribers via the ROS Toolbox that listens for updates of topics published by nodes in ROS.

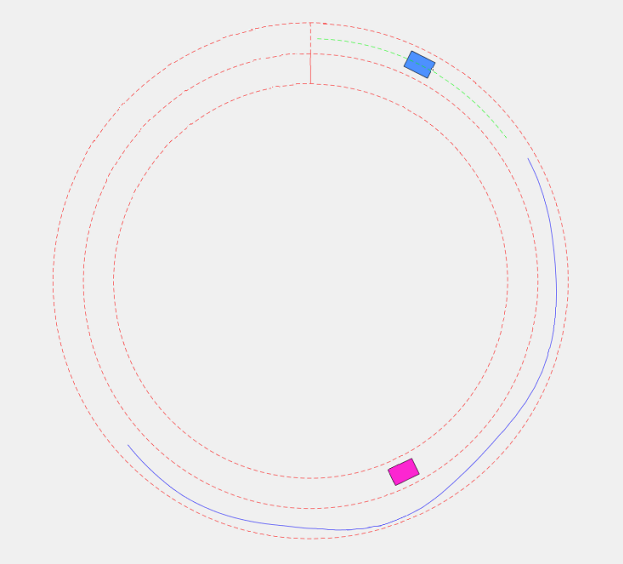


Figure 20: Matlab plot window. The blue line is the travelled path of the ego vehicle and the green path is the planned path.

8.3 Graphical User Interface

The graphical user interface (GUI) is created with the open source project QT through the python API PyQt5. The GUI contain buttons for altering the controller and planner when testing, and for starting and stopping all vehicles if something unexpected happens. Allowing overtakes is also toggled through a button. Decisions and information such as if the ego vehicle detects a car ahead of it and the distance from said car, whether it's possible to overtake and intent are displayed in a status window. Figure 19 shows all the functionalities of the GUI.

9 FUTURE IMPROVEMENTS

In this section we reflect on some potential improvements that we have identified so far.

9.1 Perception

The cars perception of the environment have not been explored in this project. To make the car fully autonomous the car would need some way of determine its environment in run-time. Some form of computer vision could be implemented to allow the car to explore more scenarios other than the hard-coded scenarios in this project and even unexplored environments. The system could be extended with a camera to detect

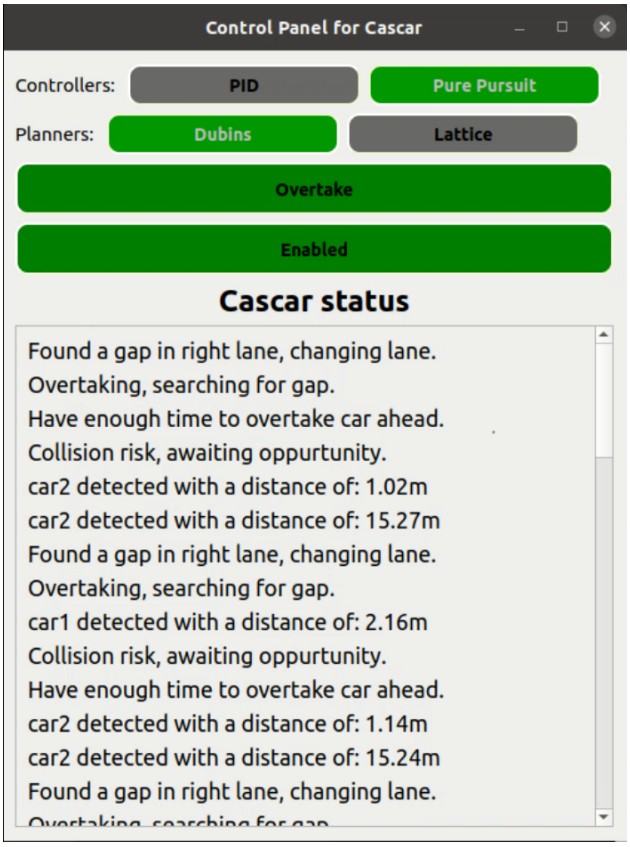


Figure 21: GUI control panel.

the lanes and the already implemented LIDAR could be extended to detect physical objects and integrate that information in the representation of the environment.

9.2 Behaviour layer and Local path planning

The behaviour layer should be further developed to include handling of more scenarios. As the behaviour layer is difficult to generalise and there are many unpredictable events that can occur in a generalised environment, the behaviour layer has only focused on meeting the set scenario goals. However, there is an important safety aspect that can be added to the current behaviour layer to increase the robustness, which is the possibility of being able to interrupt an overtaking if necessary. The behaviour layer bases its decisions on the fact that surrounding cars always drive at a constant speed and that all measurements are perfect, which is not certain and should therefore include the possibility to interrupt an overtaking.

There have been lots of experimentation with local path planning. The focus have mainly revolved around lattice planning. The lattice planning was deemed incompatible in an environment such a circular road with its constraints. However, the lattice planner worked well in an open environment. This is why the local path planning in this project uses segments of the circular road. To make the autonomous system able to function in other environments the local path planner should be extended to work in environment with less fixed constraints then a circular road.

9.3 Controller

A lot of work was put into developing an MPC controller for trajectory control of both the longitudinal and lateral behaviour. The main problem in implementing an MPC in run-time is the computing power of the system. With some tuning and a short look-ahead-horizon, the MPC was able to run but not well enough to meet the performance requirements. With an MPC implemented the obstacle avoidance could be improved greatly.

9.4 Visualisation

The visualisation in the Matlab plot window had substantial delays. This is believed to be due to the computational power required in handling the rather large amount of coordinates being updated at small time intervals. There are a couple of things that can be improved, with varying difficulty. The easiest ones being a reduction in resolution, i.e. less coordinates, and replacing the instances where plot objects are deleted and created anew for every new set of data to instead replacing it with the XData and Ydata properties. The Matlab script receives the coordinates in lists of a fixed length which results in a large portion of the same data points being deleted and then added back to the plot. In order to improve this, new publishers could be created in ROS that only publish the data points that were created from the last publish.

REFERENCES

[1] M. Quigley, K. Conley, B. Gerkey, *et al.*, "Ros: An open-source robot operating system," in *ICRA workshop on open source software*, Kobe, Japan, vol. 3, 2009, p. 5.

[2] T. Huang, *Rplidar-a2 laser range scanner solid laser range scanner*. [Online]. Available: https://www.slamtec.com/en/Lidar/A2.

[3] A. Industries, *Adafruit mpu-6050 6-dof accel and gyro sensor - stemma qt qwiic*. [Online]. Available: https://www.adafruit.com/product/3886.

[4] *Chopper-stabilized precision hall-effect switches*. [Online]. Available: https://www.allegromicro.com/en/products/sense/switches-and-latches/three-wire-hall-effect-switches/a1120-1-2-5.

[5] *Raspberry pi 4 product page*. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/.

[6] *Arduino pro mini product page*. [Online]. Available: https://www.arduino.cc/en/pmwiki.php?n=Main/ArduinoBoardProMini.

[7] Qualisys, *Motion capture – mocap*, 2021-11-22. [Online]. Available: https://www.qualisys.com/.

[8] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[9] F. Gustafsson, *Statistical sensor fusion*, Third edition. Lund: Studentlitteratur, 2018, ISBN: 9789144127248.

[10] X. Rong Li and V. Jilkov, "Survey of maneuvering target tracking. part i. dynamic models," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 39, no. 4, pp. 1333–1364, 2003. DOI: 10.1109/TAES.2003.1261132.

[11] A. Iserles, *A First Course in the Numerical Analysis of Differential Equations*. Cambridge University Press, 1996, ISBN: 978-0-521-55655-2.

[12] J. Byström and J. Byström, *Grundkurs i statistik*. Natur och Kultur Läromedel, 2011, ISBN: 9789127122369.

[13] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, pp. 33–55, 2016. DOI: 10.1109/TIV.2016.2578706.

[14] R. Rajamani, *Vehicle Dynamics and Control*. Springer-Verlag New York Inc., 2011, ISBN: 9781441938893.

[15] C. R. Coulter, *Implementation of the pure pursuit path 'hcking algorithm*, 1992. [Online]. Available: https://www.ri.cmu.edu/pub_files/pub3/coulter_r_craig_1992_1/coulter_r_craig_1992_1.pdf.

[16] E. Frisk, *Ground vehicle motion control*, 2021. [Online]. Available: https://gitlab.liu.se/vehsys/tsfs12/-/blob/master/Lecture_notes/lecture_06_ground_vehicle_motion_control.pdf.

[17] *System identification toolbox*. [Online]. Available: https://se.mathworks.com/products/sysid.html.

[18] L. Ljung, T. Glad, and A. Hansson, *Modelling and Identification of Dynamic Systems*, Second edition. Lund: Studentlitteratur, 2021, ISBN: 9789144153452.