Motion Planning using Positively Invariant Sets on a Small-Scale Autonomous Vehicle

Richard Bai
Karl Fredrik Erliksson

LUND UNIVERSITY
Department of Automatic Control

MSc Thesis
TFRT-6053
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Tryckeriet i E-huset
Lund 2018

Abstract

Self-driving technology has the opportunity to increase safety in automotive transportation by reducing the impact of human error. Motion planning is a key component of an autonomous system, responsible for providing reference trajectories and paths that the vehicle should follow. This thesis studies a motion-planning algorithm based on positively invariant sets. The focus is on design, implementation, and evaluation of the algorithm on a small-scale ground-vehicle robot platform. By incorporating a gain-scheduling approach into the motion planner, guaranteed safe reference trajectories, capable of navigating the vehicle in a dynamic environment of static and moving obstacles, can be computed for time-varying velocities.

This thesis also deals with sensor-fusion aspects for autonomous vehicles. Through a localization system based on an Extended Kalman Filter (EKF), reliable and robust state estimates can be obtained from inertial sensor data, without the use of an external positioning system. It is shown that the motion-planning algorithm together with the localization system is capable of performing safe overtaking maneuvers for time-varying velocities. Simpler urban driving scenarios involving traffic signs and intersections are used to illustrate the ability of the proposed motion-planning algorithm to also handle more complex driving scenarios. By using laser scans from Light Detection and Ranging (LIDAR) equipment, it is shown that obstacles can be detected and avoided in real-life driving experiments.

Acknowledgments

We would like to acknowledge Dr. Karl Berntorp at AB Berntec for providing us with this Master's Thesis opportunity and for supervising us throughout this journey. His dedication and interest in our work has been a great source of inspiration. We would also like to thank our main supervisor Dr. Björn Olofsson for his continuous guidance and support during the thesis studies. We are grateful to our examiner Prof. Anders Robertsson for providing valuable insights and inputs to our work. Furthermore, we would like to extend our gratitude to the Control and Dynamical Systems group at Mitsubishi Electric Research Laboratories (MERL) in Boston, for providing us with the necessary code and equipment needed for this thesis. Lastly, our gratitude goes to Pontus Andersson for helping us with the experimental setup in the RobotLab and to Marcus Greiff for providing us with positioning equipment. It has been a tremendously educative experience with quite a few challenges and a whole bunch of fun.

Richard and Karl
Lund, June 2018

Contents

List of Abbreviations

ABS	Anti-Lock Braking Systems
EKF	Extended Kalman Filter
ESC	Electronic Stability Control
ESP	Electronic Stability Program
EV	Ego Vehicle
GPS	Global Positioning System
IMU	Inertial Measurement Unit
LIDAR	Light Detection and Ranging
LQI	Linear Quadratic Integral Regulator
LQR	Linear Quadratic Regulator
MERL	Mitsubishi Electric Research Laboratories
MPC	Model Predictive Control
MSE	Mean Squared Error
NMPC	Nonlinear Model Predictive Control
OV	Other Vehicle
ROI	Region of Interest
ROS	Robot Operating System
SLAM	Simultaneous Localization and Mapping
STD	Standard Deviation
UKF	Unscented Kalman Filter

1

Introduction

1.1 Motivation

Research within the field of autonomous vehicles has gained more attention, both in academia and in industry, in the last decade. Advancements in the performance of computers and sensors together with smaller component sizes allow today's vehicles to explore new possibilities for improved vehicle perception and control. The automobile industry has throughout history focused a lot of research on the safety aspects of vehicles. In the early days of automotive history, the safety systems onboard vehicles usually consisted of mechanical apparatuses. An example is the three-point seat-belt developed by Volvo in the 1959 that significantly reduced the likelihood of serious injures in traffic collision [Volvo, 2009].

The introduction of Anti-Lock Braking Systems (ABS) in 1978 marked the beginning of control systems for active safety in ordinary production cars [Burton et al., 2004]. The Electronic Stability Program (ESP) introduced in 1995 that is used to avoid excess understeering and oversteering is another example [Liebemann et al., 2004]. More advanced systems similar to ESP have been developed in the 2000s, and they fall under the category of Electronic Stability Control (ESC) systems. Although these safety systems have reduced the fatality rate in traffic significantly, traffic accidents are still one of the most common causes of death in most countries. In 2015 there were 35,092 traffic-related fatalities, 2.4 million injuries and 6.3 million collisions reported in the United States [NHTSA, 2017]. It is estimated that a significant majority of the accidents are attributed to driver error with intoxication and distracted drivers being the most common causes. The Swedish Government has together with the Swedish Transport Administration introduced the *Vision Zero* concept [Trafikverket, 2017] that aims to reduce the number of traffic-related fatalities to zero. Even with improvements in active safety technology, the existence of human driver errors will still prove a liability to the safety in traffic. Autonomous vehicles could bring us one

step closer to achieving *Vision Zero* by eliminating human errors from traffic.

Autonomous vehicles have also in recent time gained a lot of attention in mainstream media. The spotlight is often directed on technological breakthroughs by big tech-companies. An example is Tesla's autopilot system that allows Tesla drivers to take off their hands from the steering wheel for a prolonged period of time [NY Times, 2016]. Other major tech-companies commonly covered by the news are Google, Intel, and Uber who are also conducting a lot of research in the area of autonomous driving [Wired, 2018]. However, not all news are positive as there were recently two fatal traffic accidents involving autonomous vehicles. A driver in California was killed while using Tesla's autopilot system [NY Times, 2018a] and a pedestrian was fatally injured when being struck by an Uber autonomous vehicle in Arizona [NY Times, 2018b]. Although human errors were key factors in both accidents, the safety aspects of autonomous vehicles are of big concern for the companies involved and society, and it is evident that a lot of research remains before safe autonomous vehicles will be driving around on the streets.

Even though research on autonomous vehicles has come a long way, the numerous unsolved problems and challenges in terms of safety and reliability still need to be addressed before we see a mass-scale production of fully autonomous vehicles. In this thesis, a motion-planning algorithm proposed in [Berntorp et al., 2017; Berntorp et al., 2018b] is investigated, further developed and evaluated. Under mild assumptions, the algorithm uses feedback control and positively invariant sets to generate guaranteed collision-free closed-loop trajectory tracking for safe lane changing maneuvers and obstacle avoidance. Reliable motion planning is a key component for autonomous vehicles, since all low-level control components rely on the trajectory determined by the motion planner.

1.2 Background

In 2014, the Society of Automotive Engineers (SAE) introduced a standard for grading autonomous vehicles in the scale of 0 to 5 [SORAVS, 2014]. The bottom of the scale starts at level 0, where all driving tasks are under human control. Level 1 autonomy includes basic assistance features such as adaptive cruise control, ABS, and ESC. At level 2, more advanced assistance systems involving longitudinal and lateral control, and emergency braking are included. A vehicle with Level 3 autonomy is capable of fully autonomous driving and monitors its surrounding without any human interaction under certain conditions. However, once leaving the specified conditions it requires a human driver to take over. Level 4 autonomous vehicles are capable of

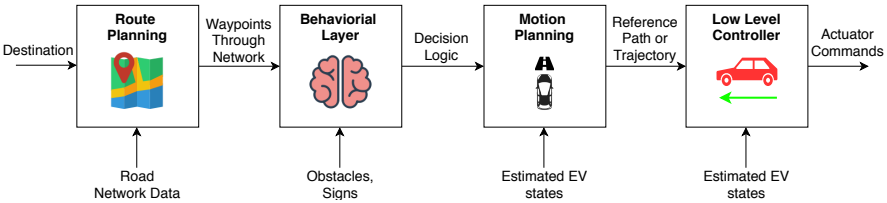


Figure 1.1 The decision-making hierarchy of autonomous driving.

autonomous driving under certain conditions but can also safely navigate the vehicle even if human intervention is requested. Level 5 autonomous vehicles are able to achieve full autonomy under all conditions.

One of the major milestones in the field of autonomous driving is the DARPA Grand Challenge in 2004, where the goal was to navigate a 150-mile off-road course as fast as possible. In the first race ever held, none of the vehicles finished the race but only a year later 5 of 23 teams made it to the finish line [Buehler et al., 2007]. This challenge and the subsequent ones held were paramount in bringing up the pace of the research within driverless technologies in both academic and industrial settings.

The decision-making hierarchy of autonomous driving is often divided into four different levels as illustrated in Figure 1.1. At the highest level, route planning is done based on user-specified destinations using road data and onboard sensors such as a Global Positioning System (GPS). The road network can often be represented using directed graphs with weighted edges representing the cost of traveling along a specific road. These problems can then be formulated into a shortest-path problem to find an optimal route between two destinations. However, due to the nature of road networks, the graphs are often very large and complex and thus classical shortest-path algorithms may prove impractical. There have been a lot of success in algorithms that use pre-processing steps to speed up calculations. In some cases, the algorithm is able to return an optimal route in a continental-sized road network in milliseconds [Goldberg and Harrelson, 2005].

The second layer is a behavioral level responsible for the local driving tasks that take the car towards the destination by following guidelines with respect to traffic rules and signs. The road given by the route planner can often be divided into segments, where each segment follows a specific set of rules and driving conventions. The traffic on highways often traverses with nearly constant velocity and there are specific rules for overtaking. The rules could differ significantly in an urban environment where an abundance of traffic

signs, crossing vehicles and pedestrians interact in a more complex manner. Since the driving scenarios are often very specific for each driving context, behaviors of each driving context can be modeled as finite sets. Hence, the decision making can be easily automated by modeling each behavior as a state in a finite state machine and having well-defined transitions between the states governed by the perceived driving context. This method of using finite state machines coupled with heuristic methods was shown to be very effective in the DARPA Urban Challenge [Buehler et al., 2009]. However, the traffic may not always be as well defined in urban driving environments. The behavior and intentions of other traffic participants can sometimes be uncertain, causing difficulties in decision making.

The motion-planning layer is the next layer in the decision-making hierarchy. Motion-planning systems in autonomous vehicles are responsible for planning a path or trajectory that is dynamically feasible for the vehicle, comfortable for the passenger, and is able to avoid collisions with obstacles on the road whilst following a specific set of rules given by the behavioral layer. In this thesis, a motion-planning algorithm based on positively invariant sets is investigated and further improved to handle urban driving environments with time-varying velocities.

The fourth and final layer is the low-level tracking control system that is responsible for following the trajectory or path given by the motion planner. Feedback controllers are often used to compute appropriate inputs to the actuators and to correct tracking errors. The tracking errors are often results of mismatch in vehicle models and as such these controllers often emphasize robustness and stability. Depending on the reference provided by the motion planner, different types of controllers are used. Both pure-pursuit controllers and predictive controllers, such as Model Predictive Control (MPC), are viable choices [Paden et al., 2016].

1.3 Scope

Aim and Delimitations

The four layers of the decision-making hierarchy are contextualized, in terms of the limitation of study of this thesis, and are illustrated in Figure 1.2. The blue region represents the parts that are included in the thesis to help illustrate how the motion-planning algorithm works. The green region specifies the main area of focus. The aim of this thesis is to further develop and experimentally evaluate the proposed motion-planning algorithm as described in [Berntorp et al., 2017] on a ground-vehicle robot platform. By performing

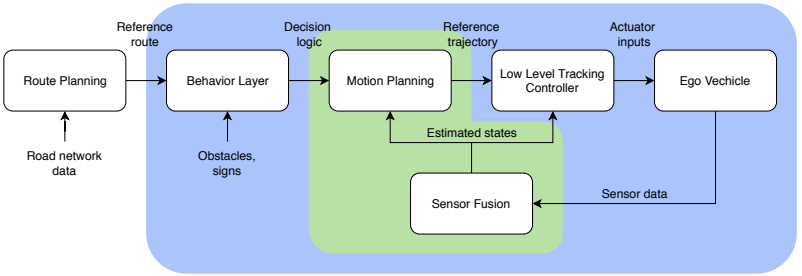


Figure 1.2 Overview of thesis. Blue represents parts that are included in the thesis and green represents the main areas of focus.

small-scale experiments on ground-vehicle robots, the goal is to better understand and experimentally evaluate the algorithm before it is implemented on a full-scale car. This process significantly reduces both cost and time to market, which are crucial for such a complex development process.

The thesis revolves around the requirements of producing a functioning motion planner. Even though the sensor-fusion aspect is one of the main areas of focus, the thesis should not be regarded as a pure sensor-fusion project. It should rather be considered as a necessary component in bringing the motion planner to fruition. As such, the main concern was only to create a functioning sensor-fusion implementation. It is also important to note that the motion planner is not intended to work as an all extensive planner capable of providing motion plans for each and every scenario. In real-life situations, other safety systems should work in conjunction with the motion planner, e.g., systems for emergency braking, to guarantee safe vehicle motion. The purpose of the motion planner is to provide safe lane keeping and overtaking in highway driving scenarios. By modifying the behavioral layer in the decision-making hierarchy, we are able to produce decision logics that can extend the motion-planning capabilities to include simpler urban driving scenarios.

Problem Formulation

This thesis aims to answer the following questions:

- How well does the considered motion-planning algorithm perform on a ground-vehicle robot platform using a localization system based on inertial sensors?

- Can the motion-planning algorithm be extended to safely navigate the vehicle for time-varying velocities?

By successfully answering the stated questions, this thesis will contribute to the research conducted on autonomous vehicle control at Mitsubishi Electric Research Laboratories (MERL) and AB Berntec. Ideally, the empirical evaluation and further algorithmic development will bring us one step closer to a fully automated car. Other state-of-the-art algorithms for autonomous driving can also be compared to our findings, hence benefitting the overall research community. This includes the work on autonomous vehicles conducted within the ELLIIT and WASP research programs at the Department of Automatic Control, Lund University.

1.4 Outline

In Chapter 2, the different vehicle models that are used throughout this thesis are introduced, ranging from simple kinematic models used to simulate data to more advanced models used for controlling the actual robot. In Chapter 3, the control concepts relevant for the motion-planning algorithm are brought up. The theory behind Lyapunov functions and positively invariant sets are introduced as well as the basics of Linear Quadratic Control. Chapter 4 deals with the sensor-fusion aspects of the thesis, where the regular Kalman filter, the Extended Kalman Filter (EKF), and the Unscented Kalman Filter (UKF) are described in detail. In Chapter 5, the investigated motion-planning algorithm is presented and derived. Chapter 6 presents the experimental procedure along with detailed accounts of the performed experiments. Some implementation aspects and algorithmic design choices are presented in Chapter 7. Afterwards follows Chapter 8, where simulation and real-life experimental results from the sensor-fusion and motion-planning experiments are presented and discussed. In Chapter 9, the most important findings of this thesis are concluded and areas of future work are brought up.

1.5 Related Work

The main area of focus in this thesis is motion planning for autonomous vehicles. Several current state-of-the-art motion-planning algorithms are covered in the survey article [Paden et al., 2016]. By defining an optimal planning problem, one can extract an optimal path using various shortest-path finding algorithms [Agarwal et al., 2002; Boissonnat and Lazard, 1996]. These methods all suffer from a common limitation: the computational complexity is often very demanding and difficult to do in real-time [Reif, 1979]. Therefore, a lot of research has been devoted to methods that find satisfactory solutions or a sequence of feasible solutions that converge to the optimal solution. An example is variational methods [Betts, 1998; Polak, 1973] that describe the

problem as a nonlinear optimization in a function space.

Sampling-based methods such as the use of Probabilistic Roadmaps (PRM) and Rapid Random Graphs (RRG) [Karaman and Frazzoli, 2011] can be used to provide a discretized graph for motion planning. To obtain an optimal path from the discretized graph, a graph search algorithm is required. Strategies such as Dijkstra's Algorithm [Dijkstra, 1959] and A* [Hart et al., 1968] are commonly used. More advanced strategies based on A* such as Anytime Repairing A* (ARA*) [Likhachev et al., 2004] and Anytime Dynamic A* (ADA*) [Likhachev et al., 2005] have also been suggested. Incremental search methods such as Rapidly-Exploring Random Trees (RRT) [Lavalle, 1998] are other popular methods, and with the improved RRT* [Karaman and Frazzoli, 2010] one can guarantee convergence toward an optimal solution.

A probabilistic framework for online motion planning of vehicles in dynamic environments has been proposed in [Berntorp and Di Cairano, 2016]. Through the usage of task specifications as artificial measurements, the exploration phase in the planner can be casted as a nonlinear, multimodal, estimation problem which can be effectively solved using particle filtering. For certain parameter choices, the approach is equivalent to solving a nonlinear estimation problem using particle filtering.

The theory of nonlinear state estimators such as the Extended and Unscented Kalman Filters is mature and has been around for a while. For highly nonlinear dynamical systems, particle filter methods are popular and have beed demonstrated to outperform the standard Kalman methods, see for example [Doucet et al., 2001; Liu and West, 2001]. More recent research activities include feedback particle filters, which are for instance studied in [Yang et al., 2013; Berntorp, 2015].

The robotic setup by CogniTeam [CogniTeam, 2018] that is used for autonomous driving experiments in this thesis has been used previously by other research teams. These applications mainly include scenarios with multi-agent communication and artificial intelligence for Search and Rescue missions [Rosenfeld et al., 2017; Sinay et al., 2017]. A similar setup is used in [Berntorp et al., 2018a] for driving experiments using an external positioning system. The sensor-fusion approach for localization and motion-planning that is adopted in this thesis has, to the best of our knowledge, not been investigated before.

2

Vehicle Modeling

This chapter presents the relevant vehicle motion models that are used in the thesis. Based on [Rajamani, 2011], the kinematic single-track model is first presented. Then the lateral dynamics is further investigated to account for more advanced vehicle parameters and converted into a road-aligned coordinate frame. Lastly, we present a simple model for the longitudinal dynamics based on a double integrator.

2.1 Kinematic Single-Track Model

The kinematic model of a four wheeled vehicle can be made arbitrarily complex. In this thesis, a simplified single-track model is used, which replaces the front and back wheels with a single central front wheel and a single central back wheel as described in [Rajamani, 2011]. The model assumes that steering is only possible at the front wheel, where δ is used to denote the steering angle. A visual illustration of the single-track model is shown in Figure 2.1. The points A and B represent the front and rear axles, and the center of mass is at the point C. The length of the wheelbase is given by $L = l_f + l_r$, where l_f is the distance between the front axle and center of mass and l_r is the distance between the rear axle and center of mass. The velocity vector is directed with angle β relative to the longitudinal axis of the vehicle, where β is the body slip. Since the vehicle is assumed to have planar motion, the coordinates p_x, p_y, and ψ can be used to completely describe the motion. The coordinates (p_x, p_y) represent the location of the center of mass of the vehicle and ψ is the heading angle, relative to a fixed global coordinate system. The governing equations of motion are then given by

$$\begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} v\cos(\psi + \beta) \\ v\sin(\psi + \beta) \\ v\cos(\beta)\tan(\delta)/L \end{bmatrix}, \tag{2.1}$$

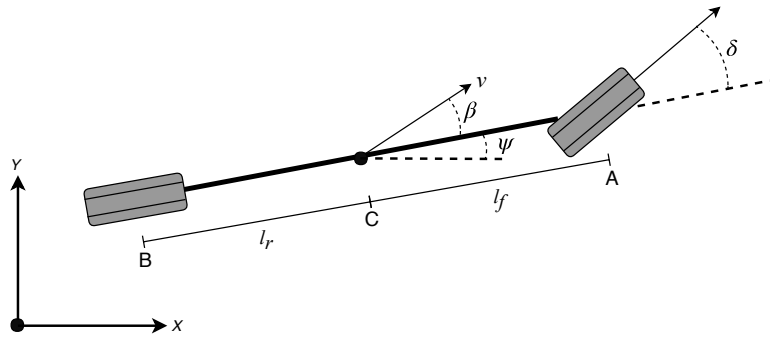


Figure 2.1 Kinematics of the single-track model.

where v and δ are considered as inputs to the system. Under mild assumptions, it can be shown that the slip angle β can be expressed as [Rajamani, 2011]

$$\beta = \arctan\left(\frac{l_r \tan\delta}{L}\right). \tag{2.2}$$

2.2 Lateral Vehicle Dynamics

Dynamic Single-Track Model

The kinematic vehicle model implicitly assumes that the velocity at each wheel is in the direction of the wheel, in other words that there is no wheel slip. This assumption is approximately valid for moderate steering. A dynamic model for the lateral vehicle motion is therefore required for more dynamic maneuvering. The lateral position, p_{lat}, is measured along the lateral axis of the vehicle with respect to the center of mass. The yaw angle, ψ, is measured with respect to the global x-axis. The longitudinal velocity, v_{lng}, is measured in the direction of the vehicle at the center of mass. Using Newton's second law of motion along the lateral axis, an equation of the forces between the vehicle and tires can be formulated as [Guldner et al., 1996]

$$ma_{\text{lat}} = F_{\text{lat},f} + F_{\text{lat},r}, \tag{2.3}$$

where a_{lat} is the inertial acceleration at the center of gravity of the vehicle along the lateral axis and $F_{\text{lat},f}$, $F_{\text{lat},r}$ are the lateral tire forces at the front and rear wheels, respectively. The inertial lateral acceleration a_{lat} consists of an acceleration $\ddot{p}_{\text{lat}}$ resulting from the motion and a centripetal acceleration $v_{\text{lat}}\dot{\psi}$. The yaw dynamics are given by moment balance about the global

z-axis, yielding

$$I_z\ddot{\psi} = l_f F_{\text{lat},f} - l_r F_{\text{lat},r}, \tag{2.4}$$

where I_z is the moment of inertia about the z-axis. The lateral tire force at the front wheel can be expressed as

$$F_{\text{lat},f} = 2C_{\alpha f}(\delta - \theta_f), \tag{2.5}$$

where θ_f is the front wheel-slip angle. Similarly, the lateral tire force at the rear wheel is given by

$$F_{\text{lat},r} = -2C_{\alpha r}\theta_r, \tag{2.6}$$

where θ_r is the rear wheel-slip angle. For each tire, proportionality constants $C_{\alpha f}$ and $C_{\alpha r}$ are defined to represent the cornering stiffnesses. Finally, using trigonometric relations to calculate θ_f and θ_r and then applying small-angle approximations, a state-space representation of the lateral vehicle dynamics can be written as [Rajamani, 2011]

$$\frac{d}{dt}\begin{bmatrix} p_{\text{lat}} \\ \dot{p}_{\text{lat}} \\ \psi \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{2C_{\alpha f}+2C_{\alpha r}}{mv_{\text{lng}}} & 0 & -v_{\text{lng}} - \frac{2C_{\alpha f}l_f - 2C_{\alpha r}l_r}{mv_{\text{lng}}} \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{2C_{\alpha f}l_f - 2C_{\alpha r}l_r}{I_z v_{\text{lng}}} & 0 & -\frac{2C_{\alpha f}l_f^2 + 2C_{\alpha r}l_r^2}{I_z v_{\text{lng}}} \end{bmatrix} \begin{bmatrix} p_{\text{lat}} \\ \dot{p}_{\text{lat}} \\ \psi \\ \dot{\psi} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{2C_{\alpha f}}{m} \\ 0 \\ \frac{2l_f C_{\alpha f}}{I_z} \end{bmatrix} \delta. \tag{2.7}$$

Dynamic Vehicle Model in a Road-Aligned Frame

The dynamic model (2.7) can be rewritten using error coordinates with respect to the road. A dynamic model with state variables expressed in error coordinates is more useful when trying to develop a steering control system. We introduce the error variables e_1 as the distance of the vehicle from its center of gravity to the center line of the lane, and e_2 as the orientation error of the vehicle with respect to the road. Assume that the vehicle is moving with constant velocity v_{lng} in the longitudinal direction of the vehicle and that the road has a constant radius R that is large, so that small-angle assumptions can be made. Then the rate of change of the desired orientation of the vehicle can be defined as [Rajamani, 2011]

$$\dot{\psi}_{\text{des}} = \frac{v_{\text{lng}}}{R}. \tag{2.8}$$

The orientation error with respect to the road, e_2, can then be defined as

$$e_2 = \psi - \psi_{\text{des}}. \tag{2.9}$$

The state-space representation obtained in (2.7) can be rewritten in terms of the road-aligned error coordinates e_1 and e_2, yielding the following lateral motion model [Rajamani, 2011]

$$\frac{d}{dt}\begin{bmatrix} e_1 \\ \dot{e}_1 \\ e_2 \\ \dot{e}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{2C_{\alpha f}+2C_{\alpha r}}{m v_{\text{lng}}} & \frac{2C_{\alpha f}+2C_{\alpha r}}{m} & \frac{-2C_{\alpha f}l_f+2C_{\alpha r}l_r}{m v_{\text{lng}}} \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{2C_{\alpha f}l_f-2C_{\alpha r}l_r}{I_z v_{\text{lng}}} & \frac{2C_{\alpha f}l_f-2C_{\alpha r}l_r}{I_z} & -\frac{2C_{\alpha f}l_f^2+2C_{\alpha r}l_r^2}{I_z v_{\text{lng}}} \end{bmatrix} \begin{bmatrix} e_1 \\ \dot{e}_1 \\ e_2 \\ \dot{e}_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{2C_{\alpha f}}{m} \\ 0 \\ \frac{2l_f C_{\alpha f}}{I_z} \end{bmatrix} \delta + \begin{bmatrix} 0 \\ -\frac{2C_{\alpha f}l_f-2C_{\alpha r}l_r}{m v_{\text{lng}}} - v_{\text{lng}} \\ 0 \\ -\frac{2C_{\alpha f}l_f^2+2C_{\alpha r}l_r^2}{I_z v_{\text{lng}}} \end{bmatrix} \dot{\psi}_{des}. \tag{2.10}$$

2.3 Longitudinal Dynamics

The longitudinal dynamics of vehicles can be modeled using advanced models with respect to the tire forces, air resistance, and drag. However, at low speeds, the impact caused by these factors is negligible and an often sufficient model for longitudinal dynamics is a simple double integrator. The state-space representation is then given by

$$\dot{x} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u, \tag{2.11a}$$

$$y = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} x, \tag{2.11b}$$

where the outputs y_1 and y_2 are the longitudinal position and longitudinal velocity, and the input u is the longitudinal acceleration.

3

Control Concepts

The aim of this chapter is to present a theoretical background to the control concepts that are used throughout this thesis. Sufficient details are provided for the reader to understand the remaining chapters without consulting further material. For a more in-depth description, we refer to the provided references in the sections.

3.1 Lyapunov Functions and Positively Invariant Sets

Consider an autonomous system

$$\dot{x} = f(x), \tag{3.1}$$

where $f : D \to \mathbb{R}^n$ is locally Lipschitz[1]. The Lyapunov stability theorem states the following important result.

Theorem 1—Lyapunov stability theorem [Khalil, 2002]
Let $x = 0$ be an equilibrium point for (3.1) and $D \subset \mathbb{R}^n$ be a domain containing $x = 0$. If we can define $V : D \to \mathbb{R}$ to be a continuously differentiable scalar function such that

$$V(0) = 0, \tag{3.2a}$$
$$V(x) > 0, \ \forall x \in D \setminus \{0\}, \tag{3.2b}$$
$$\dot{V}(x) \leq 0, \ \forall x \in D, \tag{3.2c}$$

then, $x = 0$ is stable. Moreover, if

$$\dot{V}(x) < 0, \ \forall x \in D \setminus \{0\}, \tag{3.2d}$$

then, $x = 0$ is asymptotically stable in D. □

[1] i.e., that there exists a constant $L \geq 0$ such that, for all points in D, there exists a neighborhood U where $||f(x_1) - f(x_2)|| \leq L||x_1 - x_2||, \ \forall x_1, x_2 \in U$.

A function $V(x)$ that fulfills the above requirements is referred to as a Lyapunov function.[2] If f depends on an input signal u and $V(x)$ is a Lyapunov function using some control law $u = \mu(x, t)$, then $V(x)$ is said to be a control-Lyapunov function [Amicucci et al., 1997]. There is a corresponding Lyapunov stability theorem also for autonomous systems in discrete time

$$x_{k+1} = f(x_k). \tag{3.3}$$

The derivative constraint (3.2c) is then replaced by

$$V(f(x)) - V(x) \leq 0, \ \forall x \in D \setminus \{0\}, \tag{3.4}$$

and analogously for (3.2d) [Iggidr and Mohammed, 1996].

Definition 1—Positively invariant set [Khalil, 2002]
A set M is said to be a *positively invariant set* to (3.1) if

$$x(0) \in M \ \Rightarrow \ x(t) \in M, \ \forall t \geq 0. \tag{3.5}$$

For a discrete-time system (3.3), we say that M is positively invariant if

$$x(0) \in M \ \Rightarrow \ x(k) \in M, \ \forall k = 0, 1, 2, ... \tag{3.6}$$

□

By the above definition, it follows that if we at any time instant enter a positively invariant set, then we will remain inside this region for all future time instants. Now, consider a sub-level set $\mathcal{O}$ to a Lyapunov function $V(x)$,

$$\mathcal{O} = \{x \in \mathbb{R}^n : V(x) \leq \rho\}, \tag{3.7}$$

for some $\rho > 0$. By the requirements on a Lyapunov function, it follows that $\mathcal{O}$ is a positively invariant set, since arriving at any point outside of $\mathcal{O}$ would require the value of the Lyapunov function to increase.

3.2 Motion Planning

In this section follows some general mathematical concepts that are useful for motion-planning applications.

Graphs

A (directed) graph $G = (\mathcal{V}, \mathcal{E})$ is a mathematical object defined as an ordered pair of sets $\mathcal{V}$ and $\mathcal{E}$ [Como and Fagnani, 2018]. The set $\mathcal{V}$ contains the *vertices* or *nodes* of the graph, whereas the elements of $\mathcal{E}$ are called *edges*. An

[2] In this thesis, we require that the strict inequality constraint (3.2d) is fulfilled.

edge is defined as an ordered pair (a, b) of nodes $a, b \in \mathcal{V}$. We say that a is an out-neighbor to b and that b is an in-neighbor to a if $(a, b) \in \mathcal{E}$. Commonly, a graph is illustrated as a drawing where the nodes are represented as labeled circles and the edges as arrows connecting the nodes. An example is shown in Figure 3.1.

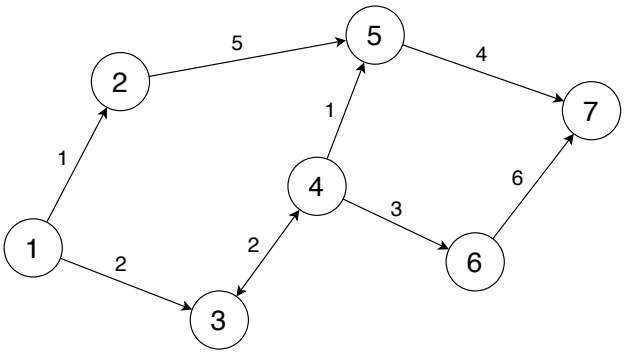


Figure 3.1 An example of an directed graph where $\mathcal{V} = \{1, 2, 3, 4, 5, 6, 7\}$ and $\mathcal{E} = \{(1, 2), (1, 3), (2, 5), (3, 4), (4, 3), (4, 5), (4, 6), (5, 7), (6, 7)\}$.

To a directed graph, one may associate a *weighted adjacency matrix* $W \in (\mathbb{R} \cup \{\infty\})^{|\mathcal{V}| \times |\mathcal{V}|}$ that assigns weights to the edges. For the purposes of this report, this can be considered as an abstraction of the cost of moving from a node to another. If $(a, b) \in \mathcal{E}$, then the corresponding element W_{ab} is finite, otherwise it is set to ∞.

Let us further define a *walk* as a sequence of nodes $P = (v_0, v_1, ..., v_k)$, where $(v_{i-1}, v_i) \in \mathcal{E},\ i = 1, ..., k$. If all nodes in the sequence are distinct, i.e., $v_i \neq v_j$ for all (i, j) such that $0 \leq i < j \leq k$, then the walk is called a *path*. The *length* of a walk or path is the sum of the edge weights

$$\text{length}(P) = \sum_{i=1}^{k} W_{v_{i-1} v_i}. \tag{3.8}$$

A path $\bar{P}$ is called a *shortest path* between two nodes $a, b \in \mathcal{V}$ if

$$\bar{P} \in \arg\inf\{\text{length}(P) : P \text{ is a path from } a \text{ to } b\}. \tag{3.9}$$

For a more thorough introduction to graph theory, see [Como and Fagnani, 2018]. In the example graph in Figure 3.1, the shortest path from node 1 to node 7 is given by $\bar{P} = (1, 3, 4, 5, 7)$ which has length 9. The problem

of finding the shortest path in an arbitrary graph is classical and there are many devoted algorithms that are designed to solve the problem efficiently, see for instance [Bertsekas, 2005]. One such algorithm is Dijkstra's Algorithm that works by exploring the nodes of the graph in a clever way. An outline of Dijkstra's algorithm is presented in Algorithm 1 [Zenklusen, 2015]. The notation $N^+(a)$ is used for the set of all edges from a node $a \in \mathcal{V}$ to its out-neighbors, i.e.,

$$N^+(a) = \{(a, b) \in \mathcal{E} : b \in \mathcal{V}\}. \tag{3.10}$$

Algorithm 1 Dijkstra's Algorithm

Input: A directed graph $G = (\mathcal{V}, \mathcal{E})$, a weighted adjacency matrix W with non-negative entries and a node $s \in V$.

Output: The shortest path P_v and its length d_v, $\forall v \in V$

function DIJKSTRA($\mathcal{V}, \mathcal{E}, W, s$)
  // Initialization:
  $M \leftarrow \emptyset$ ▷ Set of nodes that have been explored
  $d_v \leftarrow \infty,\ \forall v \in \mathcal{V} \setminus \{s\}$
  $d_s \leftarrow 0$
  $p_v \leftarrow$ `Null`, $\forall v \in \mathcal{V}$ ▷ Pointers to predecessor nodes

  // Exploration:
  while $M \neq \mathcal{V}$ **do**
    $a \leftarrow \arg\min_{v \in \mathcal{V} \setminus M} d_b$
    $M \leftarrow M \cup \{a\}$
    for $(a, b) \in N^+(a),\ b \in V \setminus M$ **do**
      if $d_b > d_a + W_{ab}$ **then**
        $d_b \leftarrow d_a + W_{ab}$
        $p_a \leftarrow b$

  // Path extraction:
  for $a \in \mathcal{V}$ **do**
    $P_a \leftarrow ()$
    $b \leftarrow a$
    while $p_b \neq$ `Null` **do**
      Append P_a with b
      $b \leftarrow p_b$
    Append P_a with s

  return $(d_v)_{v \in \mathcal{V}}, (P_v)_{v \in \mathcal{V}}$

Dynamic Programming

Dynamic programming is a general approach to solving complex optimization problems by breaking them down into smaller subproblems. We review the framework presented in [Bertsekas, 2005].

Problem Formulation Consider a discrete-time state-space model

$$x_{k+1} = f_k(x_k, u_k, w_k), \quad k = 0, 1, ..., N-1, \tag{3.11}$$

where $x_k \in S_k$ is the state, $u_k \in C_k$ the control input and $w_k \in D_k$ a disturbance acting on the system. Here, w_k is a random process that is assumed to be temporally white but may depend on x_k and u_k, i.e., $w_k \sim P_k(w_k|x_k, u_k)$.

Furthermore, we consider control inputs u_k that are allowed to depend on the current state x_k and are constrained to a subset $U_k(x_k) \subset C_k$. The aim is to find a control strategy that minimizes some additive cost function

$$g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k, w_k), \tag{3.12}$$

where $g_k(x_k, u_k, w_k)$, $k = 0, ..., N-1$, are referred to as stage costs and $g_N(x_N)$ as the terminal cost. A control strategy or *policy* is described by a sequence

$$\pi = \{\mu_0, ..., \mu_{N-1}\}, \tag{3.13}$$

where $\mu_k : x_k \to u_k$ are functions that determine the control action for any possible state we may find ourselves in. To a policy π and initial condition x_0 we associate an expected cost

$$J_\pi(x_0) = \mathrm{E}\left\{ g_N + \sum_{k=0}^{N-1} g_k(x_k, \mu_k(x_k), w_k) \right\}. \tag{3.14}$$

The optimization problem can thus be formulated as finding one optimal policy π^* (of possibly many) that minimizes the expected cost, i.e.,

$$\pi^* \in \arg\min_\pi J_\pi(x_0), \tag{3.15}$$

where the minimum is taken over all admissible policies.

Principle of Optimality Let $\pi^* = \{\mu_k^*\}_{k=0}^{N-1}$ be a solution to the optimization problem (3.15). Then, the truncated policy $\{\mu_k^*\}_{k=i}^{N-1}$ is a solution to the corresponding truncated problem where one wishes to minimize

$$\mathrm{E}\left\{ g_N + \sum_{k=i}^{N-1} g_k(x_k, \mu_k(x_k), w_k) \right\}, \tag{3.16}$$

starting from state x_i at time instant i.

The Dynamic Programming Algorithm We now formulate the Dynamic Programming Algorithm, a recursive algorithm proceeding backwards in time [Bertsekas, 2005].

Initialization:

$$J_N(x_N) = g_N(x_N), \quad \forall x_N \in S_N. \tag{3.17}$$

Recursion:

$$J_k(x_k) = \min_{u_k \in U_k(x_k)} \mathrm{E}\left\{g_k(x_k, u_k, w_k) + J_{k+1}(f_k(x_k, u_k, w_k))\right\}, \quad \forall x_k \in S_k,\ k = 0, 1, ..., N-1, \tag{3.19}$$

where the expectation is taken with respect to the distribution $P(w_k|x_k, u_k)$.

The cost $J_0(x_0)$ after the last iteration is then the optimal cost $J^*(x_0)$ in (3.15). If $\mu_k^*(x_k) = u_k^*$ is the control input that minimizes the expectation in (3.19) for all $x_k \in S_k,\ k = 0, ..., N-1$, then the sequence $\pi^* = \{\mu_k^*\}_{k=0}^{N-1}$ is an optimal policy that solves the optimization problem (3.15). The proof of optimality follows directly form the principle of optimality and by induction.

As shown in [Bertsekas, 2005], any deterministic dynamic-programming problem can be posed as a shortest-path problem by representing each state $x_k \in S_k,\ k = 0, 1, ..., N$, as a node in a graph. The edge weights are then chosen as the corresponding stage cost $g_k(x_k, u_k)$ of transitioning from some state x_k to $f_k(x_k, u_k)$, using the control input u_k. Notice that, since the problem is deterministic, g_k and f_k do not depend on w_k. Similarly, it is also possible to solve any shortest-path problem using dynamic programming.

3.3 Linear Quadratic Control

Optimal control is a field of modern control theory that tries to find a controller that is the best possible with respect to a certain objective function. This design approach is formulated mathematically as an optimization problem where one wishes to minimize a specified cost criterion. A popular and widely studied subclass of optimal control problems is the Linear Quadratic Regulator (LQR) [Anderson and Moore, 1989], which is a key component of the motion-planning algorithm in Section 5.

We consider the discrete-time infinite-horizon LQR for a dynamical system on the form

$$x_{k+1} = Ax_k + Bu_k, \quad x_k \in \mathbb{R}^{n_x}, u_k \in \mathbb{R}^{n_u}. \tag{3.20}$$

The associated cost criterion is then chosen as the quadratic function

$$J = \frac{1}{2}\sum_{k=0}^{\infty}\left((x_k - r)^T Q(x_k - r) + u_k^T R u_k\right), \tag{3.21}$$

where Q is a positive definite matrix and R is a positive semidefinite matrix that penalize state errors and control inputs, respectively. If (A, B) is stabilizible, then there exists a unique positive definite matrix P that solves the *discrete-time algebraic Riccati equation* [Lewis et al., 2012]

$$P = A^T PA - A^T PB(B^T PB + R)^{-1}B^T PA + Q. \tag{3.22}$$

The control law $u = -K(x - r)$, where $K = (B^T PB + R)^{-1}B^T PA$, is then a minimizer of J subject to the dynamical system in (3.20). [Lewis et al., 2012; Glad and Ljung, 2000].

Since the dynamical system is asymptotically stabilized by the controller and P is a solution to (3.22), it is easy to verify that

$$V(x) = (x - r)^T P(x - r), \quad x \in \mathbb{R}^{n_x}, \tag{3.23}$$

is a control-Lyapunov function for this system. Hence, the sublevel sets of $V(x)$ define positively invariant sets around the reference point r.

3.4 Model Predictive Control

MPC is a control design methodology that uses predictions of a system to optimize the behavior of the system [Maciejowski, 2002; Rawlings and Mayne, 2009]. Consider a general state-space model on the form

$$x_{k+1} = f(x_k, u_k). \tag{3.24}$$

At time instant k, MPC solves a finite-horizon optimization problem to find an optimal control sequence $\{u^*_{i|k}\}_{i=k}^{k+T_c-1}$ up to a given control horizon T_c. Instead of using the entire control sequence, only the first control input is applied. Repeatedly, a new optimization problem is formulated and solved at each time instant using the currently available information. Even though the optimal control sequences themselves are open-loop input trajectories, the iterative recomputation provides implicit feedback. This principle is often referred to as the receding horizon principle.

To the purpose of formulating the optimization problems mathematically, let us define the cost function at time instant k as

$$J_k = J_F(x_{k+T_p}) + \sum_{i=1}^{T_p} J_x(x_{k+i}) + \sum_{i=1}^{T_c} J_u(u_{k+i}), \tag{3.25}$$

where J_x and J_u are functions that penalize state errors and control inputs, respectively, and J_F is a terminal cost. Here, $T_p > T_c$ is the prediction horizon of the MPC. The optimization problem at time instant k is then given by

$$\min_{\substack{u_k,\dots,u_{k+T_c-1}\\ x_k,\dots,x_{k+T_p}}} \quad J_k, \tag{3.26a}$$

$$\text{subject to} \quad x_{i+1} = f(x_i, u_i), \quad \forall i \in \{k, ..., k+T_p-1\}, \tag{3.26b}$$

$$x_k = x_0, \tag{3.26c}$$

$$x_i \in \mathcal{X}, \quad \forall i \in \{k+1, ..., k+T_p\}, \tag{3.26d}$$

$$u_i \in \mathcal{U}, \quad \forall i \in \{k, ..., k+T_c-1\}, \tag{3.26e}$$

$$u_i = u_{k+T_c-1}, \quad \forall i \in \{k+T_c, ..., k+T_p-1\}. \tag{3.26f}$$

Notice that the control signal is held constant after the end of control horizon. Here, x_0 is used to denote the current state of the system that is used as initial condition at time instant k. An appealing element of this formulation is that input and state constraints are naturally introduced by properly defining the sets $\mathcal{X} \subset \mathbb{R}^{n_x}$ and $\mathcal{U} \subset \mathbb{R}^{n_u}$.

In the case when the model (3.24) is linear and the cost function is quadratic, the optimization problem can be casted as a quadratic program, which can be solved efficiently, see, e.g., [Borrelli et al., 2001]. If the dynamics are nonlinear, however, the optimization problems are in general much more difficult to solve owing to their nonlinear and nonconvex nature. We refer to this control design as Nonlinear Model Predictive Control (NMPC). For real-time applications with shorter sampling times, NMPC is an open challenge and a lot of research efforts are invested in developing specialized numerical methods to make the optimization problems more tractable [Zanelli et al., 2017; Quirynen et al., 2018].

3.5 Gain Scheduling

A common and straightforward approach to deal with known nonlinearities and variations in process parameters is by so called *gain scheduling* [Åström and Wittenmark, 2013]. This can be regarded as an adaptive control method where the controller parameters are changed as a function of some operating condition in a preprogrammed fashion. In this way, it is possible to tune separate controllers for different operating regions and switch between these using an auxiliary variable that correlates well with the variations in the process dynamics. Hence, it is possible to ensure satisfactory control performance over a larger operating region without, for instance, having to trade-off speed for robustness.

To illustrate the idea of gain scheduling, a general block diagram is shown in Figure 3.2.

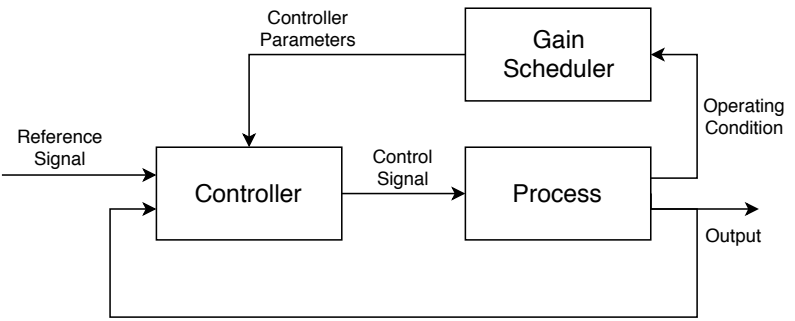


Figure 3.2 General block diagram for gain scheduling. Illustration inspired from [Åström and Wittenmark, 2013].

4

Sensor Fusion

Sensor fusion is the method of combining sensory data from different sources to compute estimates that in some sense are better than what would be possible by treating the data sources individually. In this thesis, sensor-fusion methods are used to create a localization system using inertial sensors that can be used for motion planning and low-level tracking in an autonomous vehicle. This chapter first provides a theoretical background to Kalman filter estimation theory for non-linear dynamical systems. Then, two different state observers are derived based on the vehicle models from Chapter 2. In the end of the chapter, certain implementation aspects are presented that are important in order for the localization system to work well in practice.

4.1 Kalman Filter Estimation Theory

Kalman Filter

Consider a linear state-space model on the following form

$$x_{k+1} = A_k x_k + B_k u_k + N_k v_k, \tag{4.1a}$$

$$y_k = C_k x_k + D_k u_k + e_k, \tag{4.1b}$$

where v_k and e_k are white noise processes such that $\mathrm{Cov}(v_k) = Q_k$ and $\mathrm{Cov}(e_k) = R_k$. Here, A_k, B_k, N_k, C_k and D_k are matrices of appropriate dimensions that are assumed to be deterministic and known. The standard form of the well-known Kalman filter for state estimation is given by the following recursions [Gustafsson, 2010]:

Measurement update:

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + P_{k|k-1} H_k^T (H_k P_{k|k-1} H_k^T + R_k)^{-1} (y_k - H_k \hat{x}_{k|k-1} - D_k u_k), \tag{4.2a}$$

$$P_{k|k} = P_{k|k-1} + P_{k|k-1} H_k^T (H_k P_{k|k-1} H_k^T + R_k)^{-1} H_k P_{k|k-1}, \tag{4.2b}$$

Time update:

$$\hat{x}_{k+1|k} = A_k \hat{x}_{k|k} + B_k u_k, \tag{4.3a}$$
$$P_{k+1|k} = A_k P_{k|k} A_k^T + N_k Q_k N_k^T. \tag{4.3b}$$

The subscript notation $k|l$ is to be interpreted as the estimate at time instant k given measurements up to time instant l. The filter is initialized by setting $\hat{x}_{1|0} = \mathrm{E}(x_0)$ and $P_{1|0} = \mathrm{Cov}(x_0)$.

For a linear dynamical system on the form (4.1), the standard Kalman filter described above is the optimal linear unbiased estimator of the state vector x_k [Gustafsson, 2010].

The Extended Kalman Filter

A general nonlinear state-space model with linear additive noise can be written as

$$x_{k+1} = f(x_k, u_k) + N_k v_k, \tag{4.4a}$$
$$y_k = h(x_k, u_k) + e_k, \tag{4.4b}$$

for some functions f and h. A naïve estimation approach for such a system would be to linearize the model around some operating point and use the standard Kalman filter described in the previous section. A perhaps better approach would be to instead linearize the model *online* around the current state estimate. This is the basic idea of the EKF.

Let us introduce the notation f'_x and h'_x for the Jacobians of f and h with respect to x, respectively. If the standard Kalman filter is used together with a first-order Taylor approximation, then the following recursions are obtained [Gustafsson, 2010]:

Measurement update:

$$S_k = h'_x(\hat{x}_{k|k-1}, u_k) P_{k|k-1} (h'_x(\hat{x}_{k|k-1}, u_k))^T + R_k, \tag{4.5a}$$
$$K_k = P_{k|k-1} (h'_x(\hat{x}_{k|k-1}, u_k))^T S_k^{-1}, \tag{4.5b}$$
$$\epsilon_k = y_k - h_x(\hat{x}_{k|k-1}, u_k), \tag{4.5c}$$
$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k \epsilon_k, \tag{4.5d}$$
$$P_{k|k} = P_{k|k-1} - P_{k|k-1} (h'_x(\hat{x}_{k|k-1}, u_k))^T S_k^{-1} h'_x(\hat{x}_{k|k-1}, u_k) P_{k|k-1}. \tag{4.5e}$$

Time update:

$$\hat{x}_{k+1|k} = f(\hat{x}_{k|k}, u_k), \tag{4.6a}$$
$$P_{k+1|k} = f'_x(\hat{x}_{k|k}, u_k) P_{k|k} (f'_x(\hat{x}_{k|k}, u_k))^T + N_k Q_k N_k^T. \tag{4.6b}$$

The EKF has become a widely popular estimation method for weakly nonlinear dynamical systems [Ungarala et al., 2007]. Even though it is a suboptimal estimator, optimal filtering is in general untractable, which makes the EKF an attractive estimation approach owing to its simplicity [Picard, 1991].

The Unscented Kalman Filter

Theoretically, the standard Kalman filter implementation in (4.2) and (4.3) provides optimal state estimates in a mean-squared error sense if the measurement vector y is a Gaussian random vector. This is true even if the state-space model is nonlinear. It can be shown that the EKF implementation is equivalent to approximating the state vector as a Gaussian random vector, which is then propagated through a first-order approximation of the state-space model [Wan and Merwe, 2000]. This linearization approach is known to provide poor estimates and suffer from divergence issues if the nonlinearities of the dynamical system are severe [Tilton et al., 2013].

An attempt to alleviate the inherent flaws of the EKF is by the UKF that was proposed in [Julier and Uhlmann, 2004]. The idea is to represent the state-vector distribution by a set of carefully selected sample points, called *sigma points*. The sigma points are then propagated through the actual nonlinear state-space model and are used to estimate the mean and covariance matrix of the posterior state vector. The steps of the standard UKF algorithm are outlined below [Merwe and Wan, 2001]:

Initialization:

$$\hat{x}_0 = \mathrm{E}\{x_0\}, \quad P_0 = \mathrm{E}\{(x_0 - \hat{x}_0)(x_0 - \hat{x}_0)^T\}. \tag{4.7}$$

For $k = 1, 2, ...$ do:

Calculate sigma points:

$$\mathcal{X}_{k-1} = \begin{bmatrix} \hat{x}_{k-1} & \hat{x}_{k-1} + \gamma\sqrt{P_{k-1}} & \hat{x}_{k-1} - \gamma\sqrt{P_{k-1}} \end{bmatrix}. \tag{4.8}$$

Time update:

$$\mathcal{X}^*_{k|k-1} = f(\mathcal{X}_{k-1}, u_{k-1}), \tag{4.9a}$$

$$\hat{x}^-_k = \sum_{i=0}^{2L} W_i^{(m)} \mathcal{X}^*_{i,k|k-1}, \tag{4.9b}$$

$$P^-_k = \sum_{i=0}^{2L} W_i^{(c)} \left(\mathcal{X}^*_{i,k|k-1} - \hat{x}^-_k\right)\left(\mathcal{X}^*_{i,k|k-1} - \hat{x}^-_k\right)^T + Q_k, \tag{4.9c}$$

$$\mathcal{X}_{k|k-1} = \left[\hat{x}^-_{k-1} \quad \hat{x}^-_{k-1} + \gamma\sqrt{P^-_k} \quad \hat{x}^-_{k-1} - \gamma\sqrt{P^-_k}\right], \tag{4.9d}$$

$$\mathcal{Y}_{k|k-1} = h(\mathcal{X}_{k|k-1}, u_{k-1}), \tag{4.9e}$$

$$\hat{y}^-_k = \sum_{i=0}^{2L} W_i^{(m)} \mathcal{Y}_{i,k|k-1}. \tag{4.9f}$$

Measurement update:

$$P_{\bar{y}_k\bar{y}_k} = \sum_{i=0}^{2L} W_i^{(c)} \left(\mathcal{Y}_{i,k|k-1} - \hat{y}^-_k\right)\left(\mathcal{Y}_{i,k|k-1} - \hat{y}^-_k\right)^T + R_k, \tag{4.10a}$$

$$P_{x_k y_k} = \sum_{i=0}^{2L} W_i^{(c)} \left(\mathcal{X}_{i,k|k-1} - \hat{x}^-_k\right)\left(\mathcal{Y}_{i,k|k-1} - \hat{y}^-_k\right)^T, \tag{4.10b}$$

$$K_k = P_{x_k y_k} P^{-1}_{\bar{y}_k\bar{y}_k}, \tag{4.10c}$$

$$\hat{x}_k = \hat{x}^-_k + K_k(y_k - \hat{y}^-_k), \tag{4.10d}$$

$$P_k = P^-_k - K_k P_{\bar{y}_k\bar{y}_k} K^T_k. \tag{4.10e}$$

Here, L is the dimension of the state space and the weights $\{W_i\}$ are defined as

$$W_0^{(m)} = \frac{\lambda}{L+\lambda}, \tag{4.11a}$$

$$W_0^{(c)} = \frac{\lambda}{L+\lambda} + (1 - \alpha^2 + \beta), \tag{4.11b}$$

$$W_i^{(m)} = W_i^{(c)} = \frac{1}{2(L+\lambda)}, \quad i = 1, ..., 2L, \tag{4.11c}$$

where $\lambda = \alpha^2(L+\kappa) - L$ and $\gamma = \sqrt{L+\lambda}$. The design parameter α regulates the spread of the sigma points and is commonly in the interval $[10^{-4},\ 1]$, whereas the design parameter κ is usually set to 0. For Gaussian distributions it can be shown that $\beta = 2$ is optimal.

According to [Julier and Uhlmann, 2004], the UKF yields state estimates that are accurate up to second order. It is, however, shown in [Gustafsson and Hendeby, 2012] that this is in general not true, but that the UKF performs well in many common sensor-fusion applications. It should also be noted that there is no need for calculating Jacobians in the UKF formulation, which can be troublesome in certain applications.

4.2 Ground-Vehicle Localization System

In order for the motion planner and low-level tracking system to operate, robust and accurate pose and velocity estimates of the vehicle are required. The robot platform used in this thesis has a set of different sensors onboard that can be used to estimate the current state of the vehicle. This includes a 360° sweeping Light Detection and Ranging (LIDAR) system, wheel encoders and an Inertial Measurement Unit (IMU) with an accelerometer, a gyroscope, and a magnetometer. Additionally, the platform has a built-in Simultaneous Localization and Mapping (SLAM) system that can be used to estimate the pose of the vehicle in real-time [CogniTeam, 2018]. The SLAM system has, however, a rather low update frequency and the measurements are subject to outliers. The aim of the localization system is to fuse the measurements from the available sensors to get robust and accurate state estimates at a higher update frequency.

Available Sensor Data

IMU The onboard IMU includes an accelerometer, a gyroscope, and a magnetometer that measure acceleration, angular velocity, and the magnetic field, respectively, in a 3-axis inertial coordinate frame. For the purposes of planar position and pose estimation, the longitudinal and lateral acceleration components and the yaw rate are of primary interest. However, all sensors are subject to drift and high-frequency measurement noise. Because of the different noise characteristics of the IMU sensors, fusion techniques can be used to correct for the drift terms. The software on the robot platform includes a pre-configured Madgwick filter that utilizes the magnetometer to reduce the drift of the gyroscope [Madgwick, 2010].

In the derived localization system, the filtered IMU sensor data are used to estimate the state evolution between the low-frequency updates of the SLAM system. A commonly used sensor model for gyroscopes is the Farnekopf model [Lefferts et al., 1982]

$$\dot{\psi}^m = \dot{\psi} + \dot{\psi}^b + n_{\dot{\psi}}, \tag{4.12}$$

where the superscripts m and b denote the measured value and the slow time-varying bias term, respectively. Furthermore, $n_{(\cdot)}$ is a Gaussian white noise random variable to account for high-frequency measurement noise. We use similar models also for the measured longitudinal and lateral acceleration components

$$a^m_{\text{lng}} = a_{\text{lng}} + a^b_{\text{lng}} + n_{a_{\text{lng}}}, \tag{4.13a}$$

$$a^m_{\text{lat}} = a_{\text{lat}} + a^b_{\text{lat}} + n_{a_{\text{lat}}}, \tag{4.13b}$$

which is a common approach for modeling intertial sensors [Trawny and Roumeliotis, 2005]. According to the documentation by CogniTeam, the IMU has an update frequency of 45 Hz [CogniTeam, 2018], which also has been verified empirically.

SLAM The built-in SLAM system uses the `amcl` package [Gerkey, 2018a] in ROS to provide estimates of the pose of the vehicle. A further description of ROS is provided in Chapter 6. This package is based on a probabilistic particle filter algorithm for planar motion that uses laser scans from the LIDAR to localize the vehicle against a known map of the surroundings. Internally, the SLAM system also utilizes an odometry model together with IMU sensor data to make the estimates smoother and more reliable. Even though the SLAM and the IMU data are correlated, we consider them as independent to simplify the estimation problem and make it more tractable. The update frequency of the SLAM system has been measured empirically to approximately 1.6 Hz.

Wheel Encoders The platform includes wheel encoders that measure the angular velocity of all four wheels. These raw measurements are, however, not directly available owing to restrictions in the built-in software. Instead, the raw data are processed internally and converted to a single estimate of the longitudinal velocity based on the assumption that there is no slip between the wheels and the ground. The update frequency of the encoder has been measured empirically to approximately 45 Hz.

Process Model

In order to estimate the state of the vehicle using a Kalman filter observer, a state-space model is required. To this purpose, we begin by deriving a process model, i.e., a model for the evolution of the state vector. Since the system should be generalizable to a full-scale vehicle, the observer should ideally not rely on vehicle parameters that are difficult to determine. Also, it is desirable that the estimates are expressed in the global fixed frame coordinate system, since this is what the low-level tracking controller uses to follow the reference trajectory. This motivates the use of the nonlinear

kinematic single-track model (2.1) as starting point.

The robot platform uses a simplified front-wheel Ackermann steering scheme to control the vehicle's motion. In short, only two actuator inputs are available to the tracking controller; the desired longitudinal velocity and the steering angle of a virtual wheel located at the center of the front axle. The wheel encoders are used to ensure that the velocity command is followed. The Ackemann steering geometry of the front axle implies that the vehicle will behave just as a front-wheel steered bicycle. This is convenient since this is the same geometrical interpretation as of the kinematic single-track model. We denote these command signals as $v_{\text{lng}}^{\text{ref}}$ and δ^{ref}.

Apart from the inherent states of the single-track model (2.1), the motion-planning algorithm in Chapter 5 relies on estimates of the longitudinal and lateral velocity of the vehicle. After some initial experiments, it was evident that the steering angle command signal is subject to an offset in the order of a couple degrees. Additionally, the offset is not constant but can vary slightly depending on velocity and steering angle. This turned out to be quite problematic from an estimation point of view, since even a small offset in the steering angle results in an odometry model with significantly different turning characteristics. Hence, the steering offset had to be estimated online by the observer in order to achieve satisfactory performance.

A lot of observer structures were evaluated with different bias terms included in the state vector. If all IMU sensor biases and the steering offset are included in the process model, the dynamics become practically unobservable. A reason is that the yaw rate is almost proportional to the steering angle for moderate steering action[1] and hence these two bias terms are difficult to distinguish from each other. Consequently, the biases in the sensor models (4.12) and (4.13) were instead estimated offline. In this thesis, since the observer is to be used for short driving experiments of only a couple of minutes, the drifts are small and it is reasonable to assume that the sensor biases are constant.

We therefore define the state vector to be

$$x = \begin{bmatrix} p_x & p_y & \psi & v_{\text{lng}} & v_{\text{lat}} & \delta & \delta^b \end{bmatrix}^T, \tag{4.14}$$

where v_{lng} and v_{lat} denote the longitudinal and lateral velocity, respectively. Here, δ is the actual steering angle and δ^b is the steering offset. We model the low-level tracking controller together with the steering angle dynamics

[1] Using the approximation that $\tan(\delta) \approx \delta$ for small δ.

as a first-order system G,

$$\tilde{\delta} = G(s)(\tilde{\delta}^{\text{ref}} + \tilde{\delta}^b) = \frac{1}{s\tau + 1}(\tilde{\delta}^{\text{ref}} + \tilde{\delta}^b) \quad \Rightarrow \quad \dot{\delta} = \frac{\delta^{\text{ref}} + \delta^b - \delta}{\tau}, \tag{4.15}$$

where $\tau > 0$ is the time constant of G, and $\tilde{(\cdot)}$ is used to denote the Laplace transform of a signal. The steering offset is modeled as a continuous time equivalent of a random walk, i.e., $\dot{\delta}^b = 0$ corrupted by Gaussian noise. It remains to derive a model for the derivative of the two velocity states. For a rigid body moving in a 2-dimensional plane, it holds that [Villagra et al., 2008]

$$a_{\text{lng}} = \dot{v}_{\text{lng}} - v_{\text{lat}}\dot{\psi}, \tag{4.16a}$$

$$a_{\text{lat}} = \dot{v}_{\text{lat}} + v_{\text{lng}}\dot{\psi}, \tag{4.16b}$$

where a_{lng} and a_{lat} are the longitudinal and lateral acceleration components. Both these quantities are available as measurements from the IMU. Hence, if we consider the measurements of a_{lng} and a_{lat} as input signals to the observer, we get that

$$\dot{v}_{\text{lng}} = a_{\text{lng}} - v_{\text{lat}}\dot{\psi} = a^m_{\text{lng}} - a^b_{\text{lng}} - \frac{v_{\text{lng}} v_{\text{lat}} \tan(\delta)}{L}, \tag{4.17a}$$

$$\dot{v}_{\text{lat}} = a_{\text{lat}} - v_{\text{lng}}\dot{\psi} = a^m_{\text{lat}} - a^b_{\text{lat}} - \frac{v^2_{\text{lng}} \tan(\delta)}{L}. \tag{4.17b}$$

In the single-track model, the dynamics are expressed in terms of the speed v. From the definition of the slip angle β it holds that

$$v = \frac{v_{\text{lng}}}{\cos\beta}. \tag{4.18}$$

From an estimation point of view, it is preferable to express the speed v in terms of the longitudinal velocity rather than the lateral, since β is small for the moderate steering actions that are considered here.

We finally define the input vector to the observer as

$$u = \begin{bmatrix} \delta^{\text{ref}} \\ a^m_{\text{lng}} - a^b_{\text{lng}} \\ a^m_{\text{lat}} - a^b_{\text{lat}} \end{bmatrix}, \tag{4.19}$$

using the offline estimates for the sensor biases. Putting it all together, we

arrive at the following continuous-time process model

$$\dot{x} = f(x,u) = \begin{bmatrix} x^{(4)}\cos(x^{(3)}+\beta)/\cos(\beta) \\ x^{(4)}\sin(x^{(3)}+\beta)/\cos(\beta) \\ x^{(4)}\tan(x^{(6)})/L \\ u^{(2)} - x^{(4)}x^{(5)}\tan(x^{(6)})/L \\ u^{(3)} - (x^{(4)})^2\tan(x^{(6)})/L \\ (u^{(1)} - x^{(6)})/\tau \\ 0 \end{bmatrix}, \tag{4.20}$$

where $\beta = \arctan(l_r \tan(x^{(6)})/L)$. Here, superscripts are used to denote elements of a vector. To implement the observer, it needs to be converted to discrete time. Using a forward-Euler discretization scheme [Åström and Wittenmark, 1997] with sampling time h we get that

$$x_{k+1} = \tilde{f}(x_k, u_k) := x_k + hf(x_k, u_k). \tag{4.21}$$

Measurement Model

To improve the robustness and accuracy of the state estimates, as many relevant measurements as possible should be included in the observer. Since the longitudinal and lateral acceleration components have already been introduced as input signals, we define the measurement vector to be

$$y = \begin{bmatrix} p_x & p_y & \psi & v_{\text{lng}} & \dot{\psi} \end{bmatrix}^T, \tag{4.22}$$

where the first three elements are provided from the SLAM system, the fourth from the wheel encoders and the last from the IMU. Next, the available measurements are modeled in terms of the state and input vectors. The first four measurements are also states of the model. The yaw rate $\dot{\psi}$ is modeled identically as in the process model, which yields the following measurement model

$$y = h(x,u) = \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ x^{(3)} \\ x^{(4)} \\ x^{(4)}\tan(x^{(6)})/L \end{bmatrix}. \tag{4.23}$$

The measurement equation at a specific discrete time instant k is then sampled from the continuous model as $y_k = h(x_k, u_k)$.

Asynchronous Measurement Updates

The sensors on the robot platform do not all have the same update frequency. For instance, the SLAM system has an update frequency of about 1.6 Hz,

whereas the IMU updates at 45 Hz. The computation time for a single iteration of the observer is very short and it is therefore desirable to run the observer at the same frequency as the fastest sensor, in this case the IMU. Hence, new measurements will not be available for all elements of y at every measurement update of the observer. Since the formulations of the EKF and UKF do not require the process and measurement equations to be time invariant, we can simply remove the elements of y and $h(x, u)$ that have not been updated. This is computationally straightforward for the UKF, and for the EKF it is equivalent to setting the rows in the Jacobian $h'_x(x, u)$ that correspond to these elements to zero.

Outlier Detection

To remove the degrading impact of measurement outliers on the performance of the observer, it is desirable to detect outliers and discard them from the measurement update step. For the sensor setup on the robot platform used in this thesis, especially the LIDAR scans and hence also the SLAM measurements are subject to outliers. We employ a standard outlier detection approach outlined in [Gustafsson, 2010], but generalized for nonlinear state estimation using an EKF. Let H_k be the value of the Jacobian of $h(x, u)$ at time k. Then, the central limit theorem implies that the normalized residual $\bar{\epsilon}_k$ is approximately normal distributed,

$$\bar{\epsilon}_k = (H_k^T P_{k|k-1} H_k + R_k)^{-1/2}(y_k - H_k \hat{x}_{k|k-1}) \sim \mathcal{N}(0, I). \tag{4.24}$$

Let us now define the following test statistic for element i of the measurement vector y_k

$$T(y_k^{(i)}) = \frac{(y_k^i - H_k^{(i)} \hat{x}_{k|k-1})^2}{H_k^{(i)} P_{k|k-1} H_k^{(i)T} + R_k^{(i,i)}}. \tag{4.25}$$

Notice that a single superscript (i) for a matrix is used to denote the i-th row of that matrix. Then it holds approximately that $T(y_k^{(i)}) \sim \chi^2(1)$, where $\chi^2(1)$ is the chi-squared distribution with one degree of freedom. The null hypothesis that $y_k^{(i)}$ is an inlier is rejected if

$$T(y_k^{(i)}) > \chi^2_\alpha(1), \tag{4.26}$$

for some significance level α. If a measurement is rejected, it is simply removed from the measurement update step analogously as described in the previous section.

Filter Divergence Monitoring

Similar to the outlier detection scheme described previously, it is possible to construct hypothesis tests to prevent the EKF observer from diverging. Let

us define the following test statistic for the summed squared residuals up to time N

$$T(y_{1:N}) = \sum_{k=1}^{N} (y_k - H_k \hat{x}_{k|k-1})^T (H_k P_{k|k-1} H_k^T + R_k)^{-1} (y_k - H_k \hat{x}_{k|k-1}). \tag{4.27}$$

Since the normalized residual is approximately Gaussian, it follows that $T(y_{1:N}) \sim \chi^2(N n_y)$, where n_y is the length of the measurement vector [Gustafsson, 2010]. This statistic sums up all the squared residuals since the filter was started. By introducing a forgetting factor $\lambda \in [0, 1]$, it is possible to put more weight on recent residuals to faster detect if the filter is diverging. At each filter update, the statistic is recursively updated as

$$\begin{aligned} T(y_{1:k}) =& \lambda T(y_{1:k-1}) + \frac{1-\lambda}{n_y} (y_k - H_k \hat{x}_{k|k-1})^T \\ & \cdot (H_k P_{k|k-1} H_k^T + R_k)^{-1} (y_k - H_k \hat{x}_{k|k-1}) \end{aligned} \tag{4.29}$$

It can be shown that $T(y_{1:k}) \sim \mathcal{N}(1, 2(1, \lambda))$ and the null hypothesis that the filter has not diverged is rejected if

$$(T(y_{1:k}) - 1) \sqrt{2(1-\lambda)} > \phi_\alpha, \tag{4.30}$$

where ϕ_α is the α-quantile of the standard normal distribution [Gustafsson, 2010]. The divergence test is performed after the outlier detection and only using the elements of y_k that have not been discarded. If the divergence test fails, the covariance matrix, $P_{k|k}$, is reinitialized.

5

Motion Planning Using Positively Invariant Sets

In this chapter, we review the theory and describe the motion-planning algorithm using positively invariant sets that is investigated in this thesis. We start by formulating the original version from [Berntorp et al., 2017] that assumes that the vehicle has a constant longitudinal velocity. This algorithm is based on the work in [Danielson et al., 2016] and [Weiss et al., 2014]. Then, the theory is extended to the general case with time-varying velocity. Implementational aspects and how the algorithm is actually implemented are described in Chapter 7.

5.1 Motion Planning with Constant Velocity

Problem Statement

In this thesis, the term Ego Vehicle (EV) will be used to refer to the automated vehicle that is to be controlled. Moreover, the term Other Vehicle (OV) will be used for other moving obstacles.

In the case where the EV has a constant longitudinal velocity, the dynamic vehicle models for the lateral motion that were presented in Chapter 2 are linear time-invariant state-space systems. The motion-planning algorithm is based on the state-space representation in road-aligned error coordinates, where the state vector is defined as

$$x = \begin{bmatrix} e_y & \dot{e}_y & e_\psi & \dot{e}_\psi \end{bmatrix}^T. \tag{5.1}$$

First, the dynamical system is discretized using a zero-order hold scheme [Åström and Wittenmark, 1997], resulting in a model on the form

$$x_{k+1} = Ax_k + B\delta_k + Dd_k, \tag{5.2a}$$

$$y_k = Cx_k, \tag{5.2b}$$

where the steering angle δ_k is considered as an input and $d_k = \dot{\psi}_{\text{des}}$ as a disturbance that is acting on the system. The rate of the desired orientation of the vehicle is not considered known but can be estimated from the road curvature, which is further discussed in Chapter 7.

The vehicle itself imposes constraints on the input signal δ due to physical limitations such as the maximum allowed steering angle. This can be formulated as $\delta_{\min} \leq \delta \leq \delta_{max}$, for some vehicle parameters $\delta_{\min}$ and $\delta_{\max}$. However, the constraint can be put even stricter to ensure that the implicit assumptions in the motion-model (5.2) are valid. According to [Berntorp et al., 2017; Rajamani, 2011] the dynamic single-track model describes the motion of the vehicle reasonably well for lateral acceleration up to $0.4g$ on dry asphalt[1]. The constraint set can be written as a polytope in $\mathbb{R}^{n_u}$,

$$\mathcal{U} = \{u \in \mathbb{R}^{n_u} : H_u u \leq K_u\}, \tag{5.3}$$

for some matrices H_u and K_u.

Similar to the input signal, the output y_k is also subject to constraints. In general, these constraints need not to be constant in time, i.e., $y_k \in \mathcal{Y}_k \subset \mathbb{R}^{n_y}$, where $\mathcal{Y}_k$ depends on how the environment around the vehicle changes. We consider the following constrains on the motion of the vehicle:

- The vehicle has to always stay on the road.
- The implicit assumptions of the underlying motion model should be valid.
- The vehicle should safely avoid obstacles.

In error coordinates, the first can be imposed by simply constraining the lateral position as $e_{y,\min} \leq e_y \leq e_{y,\max}$, where $e_{y,\min}$ and $e_{y,\max}$ are determined by the boundaries of the road. As discussed above, the motion model implicitly imposes constraints on the lateral acceleration and hence also the yaw rate. This implies that $\dot{e}_{\psi\,\min} \leq \dot{e}_\psi \leq \dot{e}_{\psi\,\max}$. These two output constraints form a polytope in $\mathbb{R}^{n_y}$ which can be written as

$$\tilde{\mathcal{Y}}_k = \{y_k \in \mathbb{R}^{n_y} : H_{y_k} y_k \leq K_{y_k}\}, \tag{5.4}$$

for some matrices H_{y_k} and K_{y_k}. To reduce the computational load of the motion-planning algorithm, it is assumed in this thesis that this set does not depend on time and thus can be computed offline. We can then drop the subscript k and consider that $\mathcal{Y}_k \subset \tilde{\mathcal{Y}}$. This is a reasonable assumption if the

[1] where g is the gravitational acceleration.

number of lanes and the width of the road are constant over the planning horizon.

The last output constraint listed deals with safe overtaking and obstacle avoidance, which of course has to be done online since it, for instance, involves the motion of OVs. Let us consider the motion-planning problem over a planning horizon T_p and assume that there are M obstacles in the region of interest around the EV. Let $y_{j,i}^{\text{lng}}$ and $y_{j,i}^{\text{lat}}$ be the predicted longitudinal and lateral position of obstacle i at time $t_j \in [t_k,\ t_k + T_p]$. To ensure safe driving, the EV is enforced to avoid the lateral critical zone $[y_{j,i}^{\text{lat}} - w_s^{\text{lat}},\ y_{j,i}^{\text{lat}} + w_s^{\text{lat}}]$ in the time interval $[t_j - t_s,\ t_j + t_s]$ for all t_j. The design parameters w_s^{lat} and t_s are introduced to maintain a safety margin to obstacles in case there are sensing and prediction errors present. Since the velocity is assumed to be constant, the safety time t_s can be converted into a corresponding longitudinal safety distance w_s^{lng}. By predicting the longitudinal motion of the EV we can construct a predicted set $\mathcal{S}_{j|k,i} \in \mathbb{R}^{n_y}$ at time instant t_j where the EV would be inside the critical zone around obstacle i. The combination of all obstacle sets is then constructed as

$$\mathcal{S}_{j|k} = \bigcup_{i=1}^{M} \mathcal{S}_{j|k,i}, \tag{5.5}$$

which constitutes a forbidden subset of $\mathbb{R}^{n_y}$ that the EV must never enter. At time t_k, $\mathcal{S}_{j|k}$ is predicted for all time instants t_j in the planning horizon and the constrained output set is constructed as

$$\mathcal{Y}_{j|k} = \tilde{\mathcal{Y}} \setminus \mathcal{S}_{j|k}. \tag{5.6}$$

With all the introduced notation, we are now ready to formally define the motion-planning problem at time t_k. The goal is to compute an input trajectory $\{u_j\}_{t_j \in [t_k, t_k+T_p]}$ that satisfies the input constraints and that results in an output trajectory $\{y_j\}_{t_j \in [t_k, t_k+T_p]}$ such that $y_j \in \mathcal{Y}_{j|k}$, $\forall t_j \in [t_k, t_k + T_p]$, when applied to the motion-model (5.2).

Motion Planning Using Positively Invariant Sets

Since the velocity is assumed to be constant in this simplified scenario, the only input signal is the steering angle δ. The main idea of this motion-planning algorithm is to exploit a collection of linear state-feedback controllers on the form

$$\delta_k = -L_i(x_k - r_i) + \delta_k^{\text{ff}}, \tag{5.7}$$

in order to safely drive the vehicle to a desired equilibrium point $r_i \in \tilde{\mathcal{Y}}$. Here, d_k^{ff} is a feedforward term that is designed to compensate for the disturbance $\dot{\psi}_{\text{des}}$ that is acting on the system because of the curvature of the road.

Assume that the we have an estimate $\dot{\psi}_{\text{des}}^{\text{est}}$ of the desired yaw rate; then the feedforward control law

$$\delta_k^{\text{ff}} = -B^{\dagger} D \dot{\psi}_{\text{des}}^{\text{est}}, \tag{5.8}$$

perfectly cancels the disturbance if the estimate is in fact the true value. Here, $B^{\dagger}$ is the Moore-Penrose pseudoinverse of B, which can be calculated as $B^{\dagger} = (B^T B)^{-1} B$, since the columns of B are linearly independent [Holst and Ufnarovski, 2014].

Under the assumption that the feedforward term cancels the influence of the disturbance, an equilibrium point to (5.2) must satisfy $\dot{e}_y = \dot{e}_\psi = 0$, which implies that it belongs to $\tilde{\mathcal{Y}}$ as long as the desired lateral position is within the road boundaries.

The motion-planning problem is solved by constructing a collection of setpoints $\{r_i\}_{i=1}^{M}$ corresponding to lateral positions that forms a grid on the road. LQR design is used to find a gain matrix L_i for each setpoint that asymptotically stabilizes the motion model (5.2). As described in Section 3.1, each equilibrium point then has an associated Lyapunov function

$$V_i(x) = (x - r_i)^T P_i (x - r_i), \quad x \in \mathbb{R}^{n_x}, \tag{5.9}$$

where $P_i \succ 0$ is the solution to the corresponding discrete algebraic Riccati equation (3.22). The ellipsoidal sublevel set

$$\mathcal{O}_i = \{x \in \mathbb{R}^n : V_i(x) \leq \rho_i\}, \tag{5.10}$$

is positively invariant with respect to the motion model and hence constitute safe zones inside which we know that the vehicle will remain as long as the setpoint remains fixed. As shown in [Danielson et al., 2016] , the optimization problem of finding ρ_i^* of the largest positively invariant set that satisfies the static input constraints $\mathcal{U}$ and static output constraints $\tilde{\mathcal{Y}}$ can be solved analytically, and it holds that

$$\rho_i^* = \min \left\{ \frac{K_u^{(j)} - H_u^{(j)} \bar{u}_i}{||H_u^{(j)} L_i P_i^{-1/2}||}, \frac{K_y^{(j)} - H_y^{(j)} \bar{y}_i}{||H_y^{(j)} C P_i^{-1/2}||} \right\}, \tag{5.11}$$

where the minimum is taken with respect to all rows of H and K for polytopes $\mathcal{U}$ and $\tilde{\mathcal{Y}}$, respectively. Here, $\bar{u}_i$ and $\bar{y}_i$ are used to denote the input and output vectors corresponding to equilibrium point r_i.

Once $\mathcal{O}_i$ has been computed, a connectivity graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is formed to determine which positively invariant sets that are connected, meaning that there is a safe trajectory between the pair of sets. The planning horizon is

discretized into $N+1$ time steps, where the discretization time h_{MP} is a multiple l of the sampling time h_{MM} of the motion model. The node set is defined to be $\mathcal{V} = \{\{r_i^n\}_{i=1}^M\}_{n=0}^N$, where the superscript is used to denote the same setpoint at different time steps $n = 0, 1, ..., N$. There is an edge connecting node r_i^n to r_j^{n+1} if $\mathcal{O}_i \subseteq \mathcal{O}_j^l$, where

$$\mathcal{O}_j^l = \{x \in \mathbb{R}^4 : (x - r_j)^T (\bar{A}_j^l)^T P_j \bar{A}_j^l (x - r_j) \leq \rho_j\}, \tag{5.12}$$

and $\bar{A}_j = A - BL_j$ is the closed-loop system matrix for setpoint r_j. This condition should be interpreted as that all points in $\mathcal{O}_i$ can reach $\mathcal{O}_j$ in l time steps of motion model. A sufficient condition for r_i^n to be connected to r_j^{n+1} is that

$$(r_i - r_j)^T (\bar{A}_j^l)^T P_j \bar{A}_j^l (r_i - r_j) \leq \rho_j - \rho_i ||P_j^{-1/2} \bar{A} P_j^{1/2}||_F, \tag{5.13}$$

where $||\cdot||_F$ denotes the Frobenius norm[2] [Berntorp et al., 2017]. This sufficient condition is much easier to check than that $\mathcal{O}_i \subseteq \mathcal{O}_j^l$ directly.

An advantage of this approach is that the connectivity graph $\mathcal{G}$ defined previously can be computed offline, which significantly reduces the online computational load. The only constraint that needs to be taken care of online is the obstacle avoidance. Certainly, the optimal thing to do would be to recompute the largest scaling factor ρ for all setpoints $\{r_i\}_{i=0}^M$ at each planning step such that $\mathcal{O}_i$ does not overlap with the critical zones in $\mathcal{S}_{j|k}$. This would, however, require the entire graph to be recomputed and would be too costly for real-time motion-planning applications. An alternative is to instead only check which $\mathcal{O}_i$ that intersect with $\mathcal{S}_{j|k}$ and simply remove the corresponding nodes from $\mathcal{G}$ to form an *adjusted connectivity graph* $\tilde{\mathcal{G}}$. The online motion-planning problem then results in finding a feasible path in $\tilde{\mathcal{G}}$ from a starting node r_i^0 corresponding to the current position to some terminal node r_j^N at the end of the planning horizon.

If there exists a feasible path in $\tilde{\mathcal{G}}$, then the resulting trajectory of reference points is guaranteed to provide a closed-loop motion of the vehicle that will never enter the critical zones $\mathcal{S}_{j|k}$, hence avoiding all nearby obstacles.

Any feasible path in $\tilde{\mathcal{G}}$ will correspond to a guaranteed safe motion plan. However, in reality it is important that the EV also follows expected driving norms. For instance, the vehicle should not remain between lanes longer than necessary or pursue an overtaking in dangerous situations. The latter can partly be resolved by setting appropriate safety margins. In general,

[2] For a $m \times n$ matrix S, the Frobenius norm is defined as $||S||_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n |s_{ij}|^2}$.

the solution is to associate suitable weights to the edges of the connectivity graph so that the shortest path corresponds to a desirable driving behavior. A discussion regarding the cost function design that we have adopted is provided in Chapter 7.

Algorithm Outline

The proposed motion-planning algorithm in [Berntorp et al., 2017] works in a fashion similar to receding horizon control. At the planning step k, a motion plan is computed using the currently predicted obstacle sets $\{\mathcal{S}_{j|k}\}_{t_j=t_k}^{t_k+T_p}$ using graph search. The vehicle is controlled using (5.7) and the trajectory of reference points is used until the next planning iteration. Then, the predicted obstacle sets are recomputed and a new motion plan is generated. Even though the motion plan itself is an open-loop reference trajectory, the recomputation provides implicit feedback of prediction errors and model mismatches of the EV's actual motion.

5.2 Motion Planning with Time-Varying Velocity

The motion-planning algorithm presented in the previous section guarantees safe trajectories provided that the longitudinal velocity is constant. In reality, however, the vehicle must be able to adjust its speed in order to avoid collisions, respect traffic rules and drive responsibly. The original idea presented in [Berntorp et al., 2017] was to decouple the motion planner from the velocity controller and consider the longitudinal velocity as an output from a higher-level controller logic. In this thesis, another approach is considered where the velocity is included directly in the motion planner. Our work is based on the ideas in [Berntorp et al., 2018b] and further extends the theoretical justification to ensure safe motion planning.

The main idea to generalize the motion planner in Section 5.1 is to discretize the operating region for the longitudinal velocity and iteratively search for a feasible trajectory at different *constant* velocity levels. If it is not possible to find a safe trajectory at the nominal velocity, the search is continued. We still assume, however, that the reference velocity is constant over the planning horizon to make the planning problem more tractable. This approach, of course, requires more computations to be carried out online compared to the algorithm in Section 5.1, but since the connectivity graphs at the different velocity levels can be computed offline it is still feasible to solve the corresponding graph-search problem online for real-time motion-planning applications.

The construction of the positively invariant sets implicitly ensures that the vehicle respects the static input and output constraints. The derivation assumes a constant velocity and our approach is to employ a gain scheduling control design to find positively invariant sets that also apply when the actual velocity deviates from the nominal case. Let us consider the Lyapunov function candidate $V(x) = x^T P x$, where P is the solution to the discrete algebraic Riccati equation (3.22) for some nominal longitudinal velocity v_0. We ask ourselves if this could in fact also be a Lyapunov function for the motion model (5.2) at some deviated velocity $v_1 \neq v_0$. First, note that $V(x)$ does not explicitly depend on the actual velocity and since $P \succ 0$, it still holds that $V(x) > 0,\ \forall x \in D \setminus \{0\}$. If $V(x)$ is to be a Lyapunov function it must be non-decreasing with respect to the deviated motion model. Under the assumption that the feedforward controller completely cancels the effect of the disturbance $\dot{\psi}_{\text{des}}$, the closed-loop system is given by

$$x_{k+1} = Ax_k - BL(x_k - r) = (A - BL)x_k + BLr. \tag{5.14}$$

From a stability point of view, we can assume without loss of generality that $r = 0$, since we can otherwise apply a homogeneous state transformation to move r to the origin. By using (5.14), it follows that

$$\begin{aligned} V(x_{k+1}) - V(x_k) &= x_{k+1}^T P x_{k+1} - x_k^T P x_k \\ &= ((\tilde{A} - BL)x_k)^T P(\tilde{A} - BL)x_k - x_k^T P x_k \\ &= x_k^T \underbrace{\left[(\tilde{A} - BL)^T P(\tilde{A} - BL) - P\right]}_{:=S} x_k = x_k^T S x_k. \end{aligned} \tag{5.16}$$

Here, $\tilde{A}$ is used to denote the system matrix A for the deviated velocity, whereas B is independent of the longitudinal velocity. We conclude that $V(x)$ is in fact also a control-Lyapunov function for the deviated dynamics if $S \prec 0$. This conclusion holds as long as the deviated velocity is constant over the sampling time of the discretized motion model. If the velocity controller is not tuned too aggressively, it is reasonable to assume that the longitudinal velocity varies slowly enough for (5.16) to hold approximately for moderate accelerations.

For some given vehicle parameters, it is possible to numerically check if S is negative definite by computing the sign of the eigenvalues of S. Let the speed limit of the road be $v_{\max}$ and restrict the longitudinal velocity to be greater than some $v_{\min} > 0$.[3] It can easily be verified that S becomes indefinite very

[3] The open-loop gain of the motion model tends to infinity as $v \to 0$, which makes the system uncontrollable. It is therefore beneficial to set a lower velocity limit and separately treat $v = 0$ as the only possibility if no other feasible velocity profiles exist.

soon for deviated velocities that are greater than the nominal velocity used to design the feedback gain L. This is not very surprising since it should be more difficult to control the vehicle at higher speeds.

We use a bisection search method to find the smallest deviated velocity for which the Lyapunov function associated with the nominal controller at $v_{\max}$ still holds. A new LQR is then designed for the motion model at this limiting case and another bisection search is used to find the velocity interval where its Lyapunov function guarantees stability. This is repeated until the entire interval $[v_{\min},\ v_{\max}]$ has been searched through.

The velocity is thus used as an auxiliary variable for gain scheduling of the controller of the lateral dynamics. Depending on the velocity, different feedback gains L are used in the steering control law (5.7). The discretized velocity levels can then be assigned to a corresponding gain scheduling interval, where the LQR guarantees stability and the exact same Lyapunov functions apply. Since the analytical expression for ρ^* in (5.11) is independent of the deviated velocity, the same positively invariants sets hold for all longitudinal velocities in the respective gain-scheduling interval.[4] For each velocity level, the closed-loop system matrix $\bar{A}$ can be formed for its feedback gain L and system matrix A. This approach results in slightly different expressions for $\mathcal{O}_j^l$ in (5.12), and therefore also a specific connectivity graph $\mathcal{G}_v$ for each velocity level v.

At each planning step, a search is performed over the discretized velocity levels, starting with the speed limit of the road. If, at some iteration, a feasible path is found from the current position to the center node of some lane using the adjusted connectivity graph $\tilde{\mathcal{G}}_v$ at this velocity level, the search is terminated and the found motion plan is used. If no feasible path is found for any velocity level, the only possibility is to command the vehicle to stop. A further discussion on implementation aspects of the motion planner is presented in Chapter 7.

[4] Provided that there are no explicit constraints on v_{lng} introduced in $\mathcal{U}$ and $\tilde{\mathcal{Y}}$.

6

Experimental Procedure

In this chapter, a detailed description of the experimental setup and procedure is presented. In the experimental setup section, the Hamster robot platform, the Robot Operating System (ROS), and the various test tracks that have been used in this thesis are brought up. Then follows the experimental procedure, in which the sensor-fusion and motion-planning experiments are described.

6.1 Experimental Setup

HamsterV5

The HamsterV5 is a small robust autonomous vehicle capable of indoor localization that is mainly used for research and development purposes [CogniTeam, 2018]. The vehicle features two Raspberry Pi computers and an Arduino-based low-level controller. The Raspberry Pis act as a master and slave in a ROS symbiosis to drive the vehicle. The master connects to the slave via ethernet cable and connects to the Hamster network through Wi-Fi, whilst also providing network access to the slave. The primary task of the slave is to execute SLAM and localization algorithms and the Arduino low-level controller board is responsible for the interaction with the hardware components. Specifications of the Hamster are provided in Table 6.1 and a picture of the vehicle is shown in Figure 6.1.

The Robot Operating System

ROS is a flexible framework for writing robotics software originating from the Stanford Artificial Artificial Intelligence Laboratory in 2007 [ROS.org, 2018]. Contrary to its name, ROS is not an actual operating system, but a collection of tools, libraries, and conventions used to aid in the creation of complex robotics systems across different robotic platforms. Its modular design provides services for heterogeneous computer clusters such as low-level device control, hardware abstraction, and message-passing between

Table 6.1 HamsterV5 specifications obtained from [CogniTeam, 2018].

Dimensions	**Value**
Weight	1.7 kg
Width	0.20 m
Length	0.25 m
Height	0.15 m
Dynamics	**Specifications**
Steering Dynamics	Ackermann
Maximum speed	1.2 m/s
Steering angles	±35°

Figure 6.1 Picture of the HamsterV5, the vehicle platform used for driving experiments.

processes. Programs in ROS are either written in C++ or Python and the general architecture of a ROS system follows a graph-like structure were nodes can receive, post, and multiplex various messages from sensors, actuators, planners, controllers, and such [Quigley et al., 2009].

ROS allows the possibility of dividing the computational load between the different computers through clusters. Messages containing data from sensors, system information, and localization data from the SLAM are sent from the Hamster and can be received by any computer connected to the ROS network. The more computationally demanding motion-planning algorithms can therefore be executed on a different computer and sent to the Hamster via ROS.

Test Tracks

For this work, three miniature test tracks suitable for a Hamster-scaled vehicle have been used to evaluate and verify the performance of the sensor-fusion and motion-planning algorithms. The tracks are referred to as the NASCAR, the Jellybean, and the Intersection test track and they are illustrated in Figure 6.2. The NASCAR track is a super-ellipse shaped miniature track inspired by the racing tracks commonly used in the American auto racing competition NASCAR. The Jellybean track was designed to contain both left and right turns yielding interesting and demanding vehicle dynamics. The Intersection track was created by overlapping two NASCAR tracks of different sizes with a relative rotation of 90°.

All three tracks feature two lanes of width 0.3 meter each. Virtual versions of the tracks were used for simulation experiments in Matlab. However, a full-scale version of the NASCAR track was also measured out in reality in order to conduct real-life driving experiments using the Hamster. Using colored masking tape and laser measurement tools, the NASCAR track was marked out in the RobotLab at Lund University. A picture of the track is shown in Figure 6.3.

6.2 Sensor-Fusion Experiments

The sensor-fusion experiments consisted of both virtual simulations using simulated data as well as real experiments using data from the Hamster. In Figure 6.4, an overview of the repetition is shown that illustrates the general repetition that was followed when implementing the localization system for the Hamster. The first step was to implement an observer in Matlab. We then used simulated data to tune, debug, and reiterate the observer code until the state estiamtes converged fast and were able to track the actual values of the states without major oscillations and drift. Using the Matlab Coder toolbox [Matlab, 2018], the observer code was converted into C++ code that could be incorporated in the ROS system using wrapper code. The code necessary for communication between the observer, motion planner, and the Hamster through ROS was then implemented. Once everything worked as desired in simulations, real-life driving experiments were conducted. Since the characteristics of simulated data differed from the actual sensor data from the Hamster, the observer parameters needed to be re-tuned accordingly. Practical implementation problems that were noticed in real-life experiments required the observer implementation in Matlab to be further developed. The workflow progression was then reiterated until satisfactory performance of the observer was obtained. In the following sections, more detailed explanations are provided for the major steps in the workflow.

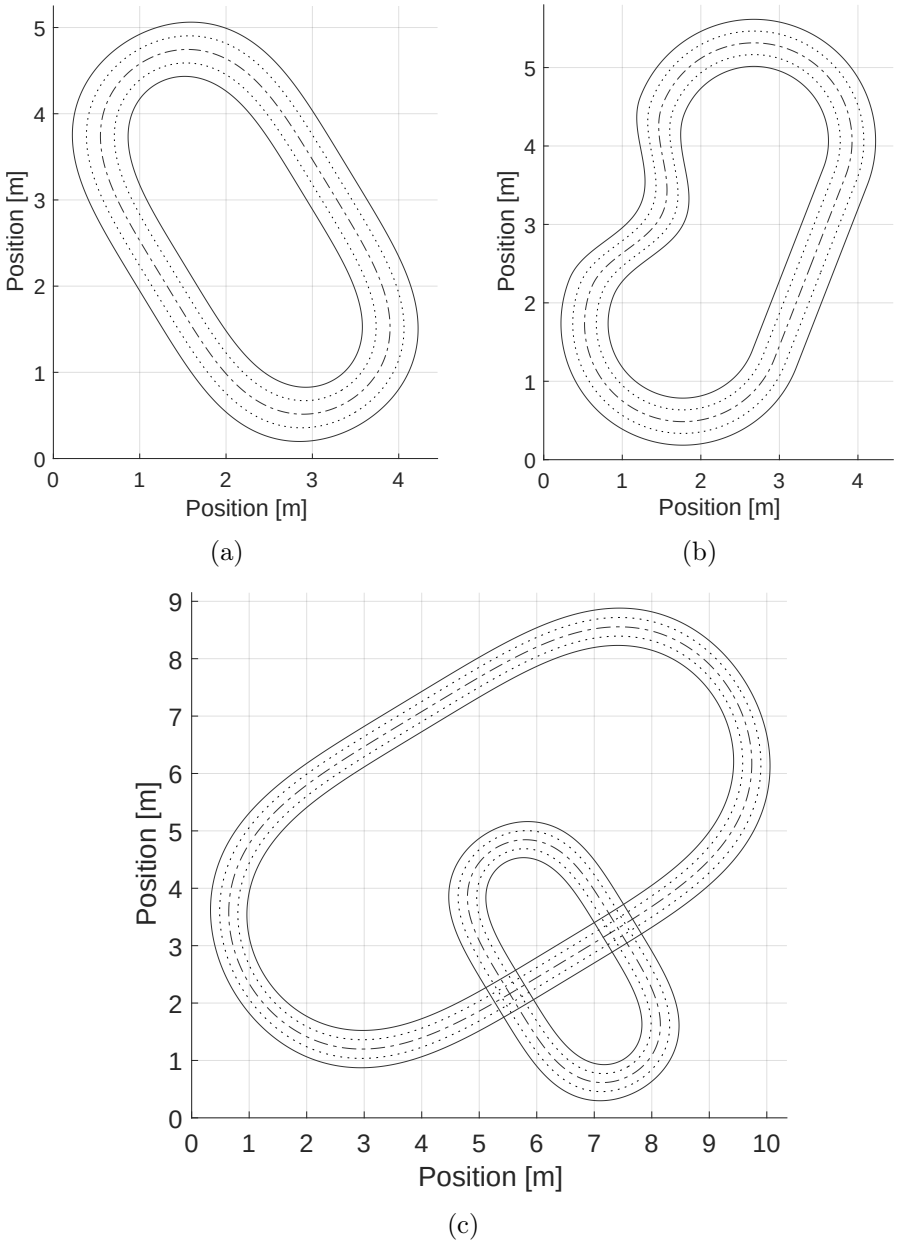


Figure 6.2 Illustration of (a) the NASCAR, (b) the Jellybean and, (c) the Intersection test tracks.

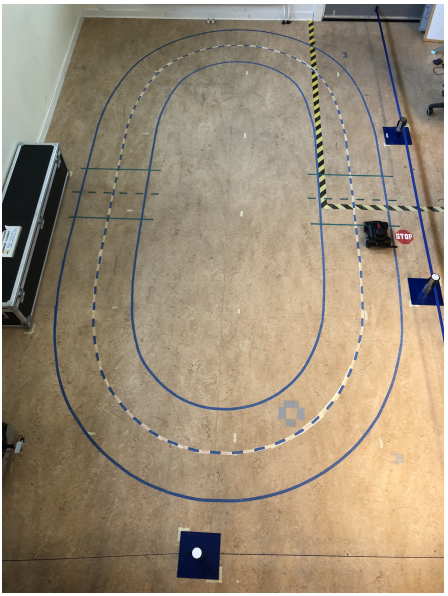


Figure 6.3 Picture of the full-scale NASCAR test track in the RobotLab at Lund University.

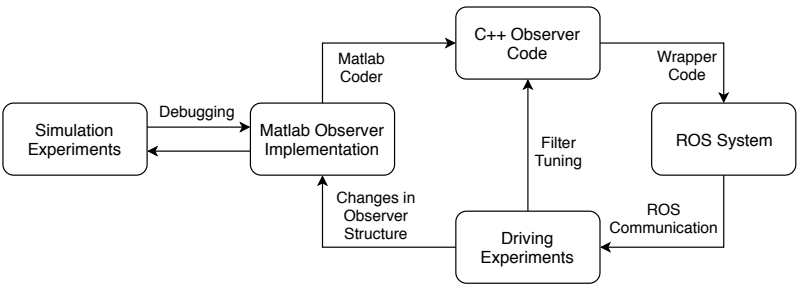


Figure 6.4 Overview of the workflow that was used to implement a localization system for the Hamster.

Simulation Data

To test and validate the Kalman filters, simulation data were required. The simulation data were produced by simulating a virtual vehicle modeled by the kinematic single-track model in Chapter 2. A front-wheel position-based feedback controller developed by Stanford University for the 2005 DARPA

Grand Challenge [Thrun et al., 2006] was implemented to control the virtual vehicle along the Jellybean test track. This track was chosen as it contains both left and right turns, yielding interesting simulation data for the observer.

The approach of the front-wheel position-based feedback controller is to use the front-wheel position of the vehicle as the regulated variable. Let v_f be the front-wheel velocity and e be the shortest distance between the front wheel and the test track. Furthermore, define θ_e as the heading error of the vehicle with respect to the tangent line at the closest point of the track. The steering control law is then given by

$$\delta = \arctan(Ke/v_f) - \theta_e \tag{6.1}$$

for some gain $K > 0$. The term θ_e can be regarded as a combination of a feed-forward and a heading-error term. It can be shown by Lyapunov theory that this controller locally exponetinally stabilizes the vehicle with respect to the reference path of the track [Paden et al., 2016].

The vehicle was simulated with a constant velocity v_0 and thus the input vector to the kinematic single-track model was given by

$$u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} v_0 \\ \delta \end{bmatrix}. \tag{6.2}$$

Since also sensor data from the IMU needed to be simulated, the yaw rate as well as the lateral and the longitudinal accelerations were computed using first and second-order finite differences on the simulated yaw and position data. Furthermore, since the SLAM, the IMU, and the wheel encoders have different update frequencies, the simulated data were modified accordingly.

Driving Experiments

The next step was to test and tune the EKF observer on actual data. With the help of retractable barriers, the area around the NASCAR test track in the RobotLab at Lund University was enclosed. The retractable barriers together with walls provided a static environment that could be used to localize the Hamster using the onboard LIDAR. A 2-D occupancy grid map of the test area was created using the `gmapping` package in ROS [Gerkey, 2018b], which is depicted in Figure 6.5. This map was provided to the `amcl` SLAM system so that pose estimates relative to the map could be computed. Pure estimation experiments were performed by publishing open-loop control inputs in ROS for a pre-designed path with varying velocities and steering-angle inputs. Since the state estimates were not used for feedback, it was possible to tune and evaluate the performance of the localization system without the impact of any other control systems.

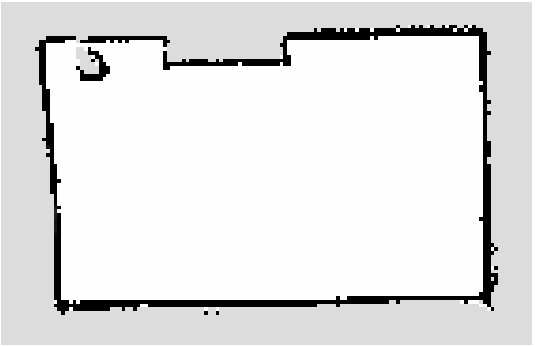


Figure 6.5 2-D occupancy grid map of the test area created using a laser-based SLAM system. The enclosed area corresponds to the walls and the retractable barriers in Figure 6.3.

6.3 Motion-Planning Experiments

The second stage of this thesis aimed to integrate the localization system into an existing ROS framework in order to evaluate the performance and continue the development of the motion-planning algorithm with time-varying velocity that was introduced in Chapter 5. To this purpose, simulated motion-planning experiments were first performed in Matlab using a virtual EV and virtual OVs. Next, further simulations were conducted with ROS nodes acting as EV and OVs. This approach made it possible to integrate and set up all the necessary communications between ROS nodes without having to deal with practical problems related to controlling the actual vehicle. Lastly, the same set of experiments were conducted with the Hamster on the full-scale test track in the RobotLab at Lund University.

The block diagram in Figure 6.6 provides a simplified overview of the ROS network with the integrated localization system. The green area represents the parts of the system that were implemented directly on the Hamster robot platform, whereas the blue area represents that parts that were implemented on an external computer. The motion-planning algorithm uses the estimated vehicle states from the observer to generate safe motion plans that avoids nearby obstacles. The tracking controller is responsible for following the provided motion plan and sends actuator commands to the Hamster Driver node that in turn controls the hardware on the platform. The dashed lines represent the indirect connection to the available sensors on the Hamster. As mentioned in Section 4.2, the onboard Madgwick filter pre-processes the raw IMU data before it is used by the state observer.

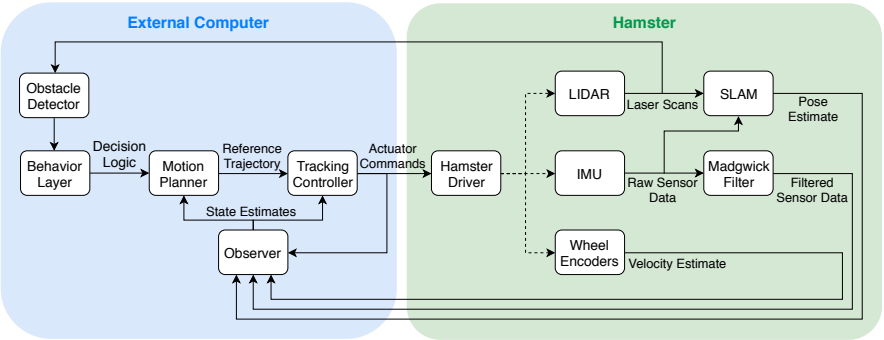


Figure 6.6 Simplified block diagram of the most important parts of the ROS system.

To create more realistic and interesting driving scenarios for the motion planner, an obstacle detection system based on laser scans from the LIDAR has been integrated into the existing system. Furthermore, a behavioral layer has been incorporated that handles higher-level decision logic depending on nearby obstacles and signs. In the following sections, more extensive details are provided for the various motion-planning experiments that have been conducted.

Autonomous Driving Experiments

The first step involved configuring vehicle parameters suitable for a small-scale robot like the Hamster. The motion models, as described in Chapter 2, required specifications of the vehicle mass, vehicle yaw inertia, front and rear tire stiffnesses, and distance to the front and rear wheels from the center of mass. The vehicle dimensions were deduced through careful measurements. The yaw inertia was estimated by approximating the vehicle's shape as a constant density cuboid. The tire stiffnesses were somewhat more difficult to determine. These parameter values are closely related to the vehicle's ability to rotate. Through thorough simulation experiments, we were able to come up with tire stiffness values that seemed to work well in practice, such that the lateral dynamics resembled the motion of the actual Hamster in real-life experiments. A complete list of vehicle parameters is presented in Table 6.2.

The next step of the experimental procedure involved tuning the parameters of the LQR such that the motion plan tracks the lanes smoothly without overshooting. During this step, it was noticed that if the vehicle parameters of the EV were wrongly specified, the EV would have the appearance of driving on an icy surface or in some cases become unstable. Once the LQR and vehicle parameters were correctly tuned, the EV was able to drive along the

Table 6.2 Estimated vehicle parameters of the Hamster.

Parameter	Value	Unit	Description
I_z	0.01	$\text{kg} \cdot \text{m}^2$	Vehicle yaw inertia
$C_{\alpha f}$	0.67	-	Front tire stiffness
$C_{\alpha r}$	0.7	-	Rear tire stiffness
l_f	0.095	m	Distance to front wheel from center of mass
l_r	0.07	m	Distance to rear wheel from center of mass
τ	0.2	s	Time constant of steering delay

test track without any problems in simulations. As such, experiments involving moving obstacles were used to simulate overtaking scenarios. Algorithmic modifications in the motion planner were introduced in order to achieve a more desirable behavior and performance. Some of the most important modifications are presented in Chapter 7.

Adaptations for Hamster Experiments The motion-planning experiments using the Hamster proved to be more complicated. In order for the Hamster to drive smoothly along the track, it is required that the motion planner, the localization system, and the ROS network work flawlessly together. In simulations, the LQR is both used to generate the motion plan and to control the motion of the virtual EV. However, in experiments involving the Hamster, a low-level tracking controller is used to follow the reference trajectory from the motion planner. This allows for a separate controller to be designed for the main purpose of tracking the reference trajectory as good as possible. Otherwise, a trade-off has to be made concerning the shape of the reference trajectory and the tracking performance of the controller. Even though the linear state-feedback controllers have nice stability properties in terms of Lyapunov functions, other controller structures may be better suited for low-level tracking applications. A comprehensive comparison of current state-of-the-art tracking controllers is provided in [Paden et al., 2016].

Theoretically, the safety guarantees of the motion planner do no longer apply when another low-level controller is used. The hope is, however, that the tracking controller has even better stability properties. In this thesis, an NMPC proposed in [Quirynen et al., 2018] was used for this purpose. Since the tracking controller is not the main focus of this thesis, it will not be discussed in detail.

The NMPC was not directly compatible as it required a reference trajectory to be expressed in global fixed-frame coordinates. This was solved by simulating the vehicle model using the input trajectory of the motion plan. This strategy results in a reference trajectory in error coordinates that could then

be converted to global coordinates by kinematic relations. Once the Hamster was able to drive on the test track without major drifts and oscillations, the same overtaking scenarios that had been studied in simulations were studied in reality, but still with virtual OVs.

Intersection Experiments

The motion-planning algorithm was implemented with the intention of working in overtaking scenarios in highway traffic, while traveling with near constant speed. Therefore, modifications were needed in order for the motion planner to function in urban environments. Driving scenarios were created in which the EV needed to react to both crossing vehicles as well as different traffic signs. For this purpose, a stop sign and a give-way sign were implemented for the Intersection test track. In the behavioral layer, a higher decision-making logic was implemented using heuristic rules that would govern the motion planner to act in a traffic-complying manner. Modifications were also needed in the vehicle motion-model as the lateral dynamics have an infinite open-loop gain when the vehicle is standing still. This was solved by defining a minimum velocity threshold below which the velocity was manually set to zero and the motion states were updated accordingly.

Moreover, a sign detection system was implemented. Once a sign was detected, a specific set of contextual behaviors were put into action. For example, when a sign is detected the cost function is altered to prevent the EV from performing overtakes. Behaviors unique to the stop sign involve telling the EV to drive to a specific stopping point next to the stop sign without taking crossing vehicles into consideration. It might seem counterintuitive to disregard the crossing vehicles, but if their predictions were to be taken into account, the EV would in some cases start braking before reaching the stop sign. In the case of the give-way sign, the predictions of the crossing vehicles were instead needed as they determine whether or not the EV should slow down, brake or continue with constant velocity. A lot of tuning was done in simulations before the EV behaved as expected by common driving norms. Once everything worked in the simulations, the same type of experiments were conducted using the actual Hamster. A further discussion regarding the decision logics of the behavioral layer is presented in Section 7.3.

Obstacle Detection with LIDAR

The LIDAR onboard the Hamster not only provides data for the SLAM system, but also information regarding the surroundings of the robot. In this thesis, an obstacle detection ROS package [Przybyla, 2017] was used as a starting point for integrating the obstacle-detector node in Figure 6.6 into the existing ROS framework. The obstacle-detection package was intended

for two 180° LIDARs that are to be merged into a single scan. As such, slight modifications were needed for the detection package to work as intended for the robot platform used in this thesis.

The obstacle detector published information regarding the position, size, and velocity of the detected objects to the ROS system. This data were then parsed in Matlab where some additional noise filtering was performed. The detected obstacles could then be treated by the motion planner just like the virtual OVs, with safety margins specified by the size of the obstacle. Eventually, tests were conducted that involved both static and moving real obstacles.

7

Implementation Aspects

To make the motion planner and the localization system work well together in experiments, there were several implementation aspects that needed to be considered. In this chapter, we highlight some of the practical challenges that have been encountered and the solution methods that have been adopted or developed. Furthermore, algorithmic design choices are presented and discussed.

7.1 Localization System

Steering Offset

The robot platform by Cogniteam that was used to test the motion-planning algorithms required quite a bit of tuning before being fully operational. One of the first things that was noticed was that given a zero steering angle input, the Hamster would still tend toward a certain direction. It was concluded that this was because of a mechanical steering offset of approximately 4 degrees in the wheel suspension. Such a large offset for a small-scale vehicle implies significantly different turning characteristics. Through empirical tests, the steering offset was estimated and could therefore manually be corrected for in the Hamster's startup configurations. However, even after incorporating this steering offset, it was still noticed that the Hamster had a difficult time driving straight and that the severity would increase with different velocities. As discussed in Section 4.2, the solution was to add an explicit bias state for the steering offset in the state observers.

IMU Calibration

Another encountered issue with the Hamster was problems with the SLAM system. In early tests it was noticed that the pose estimates would tend to drift and that the `amcl` package had problems matching the LIDAR scans against the static map. The source of these problems was determined to

be rooted in the IMU measurements that are used internally by the SLAM odometry model. Gyroscopic data obtained from the IMU are often very noisy and magnetometer data could be affected based on the geometric location of the experiments. Also, all IMU sensors are subject to possible misalignment of the coordinate axes. Since, for instance, the gravitational acceleration is very large compared to the longitudinal and lateral accelerations, a slight misalignment could be very significant to the sensor's accuracy. However, after careful measurements, it was deduced that the misalignment was small enough so that it could be estimated as constant bias terms. Since the driving experiments were carried out using rather low velocities, the roll and pitch rotations of the vehicle are negligible. Calibration of the IMU was performed using a built-in program in the Hamster's software that estimates and compensates for constant bias term in all sensor measurements. This procedure together with fine-tuning of the `amcl` configuration parameters significantly reduced the drift and improved the precision of the SLAM estimates.

Sensor Bias Estimation

Even after careful calibration of the IMU, sensor biases will always be present. The IMU calibration has to be performed while the LIDAR is turned off. During experiments, the rotation of the sweeping LIDAR causes small vibrations in the vehicle that can influence the sensors. As discussed in Section 4.2, it is difficult to estimate all sensor biases and the steering offset online by the localization system. A start-up calibration routine for the observer was implemented to estimate the bias term in the sensor models (4.12) and (4.13). While the observer was being launched and the vehicle was standing still, 10 seconds of sensor data were collected from the IMU. The mean values were then used as constant approximations for the sensor bias terms. During driving experiments of only a few minutes, the drifts of the bias terms are very small and this approach was sufficient for the purposes of this thesis.

7.2 Motion Planning

Disturbance Estimation

In the lateral vehicle dynamics, the desired yaw rate caused by the road curvature is considered as a disturbance that is compensated for by a feedforward controller. For this purpose, $\dot{\psi}_{\text{des}}$ needs to be estimated. As described in [Berntorp et al., 2017], it holds that $\dot{\psi}_{\text{des}} = v_{\text{lng}} c(t)$, where $c(t)$ is the curvature of the road which in turn can be estimated by fitting a circle segment to the lane markers of the road. The estimated disturbance d_k was updated in a forward Euler fashion,

$$d_{k+1} = d_k + v_{\text{lng}} \dot{c}(t). \tag{7.1}$$

It was noted that this scheme suffers from drift if there are systematic errors in the estimate of the curvature rate. This can easily happen for tracks like the NASCAR track that solely contains turns in one direction. As a heuristic solution, d_k was reset to zero every time the curvature estimate was sufficiently small, meaning that the road is almost straight.

Decoupling of the Longitudinal and Lateral Dynamics

The motion-planning algorithm in Section 5.2 prevents the EV from entering the critical zones of all nearby obstacles. This was done by predicting the longitudinal motion of the EV, given a constant reference velocity, and comparing this to the predicted positions of the obstacles. Since the velocity of the EV may be different from the reference velocity, a longitudinal motion-model is required. Under the assumption that the longitudinal velocity is slowly time-varying, the longitudinal and lateral dynamics of the EV's motion may be decoupled and controlled individually. This observation significantly simplifies the prediction model.

In this thesis, the simple double integrator described in Section 2.3 was used to model the longitudinal dynamics. A separate LQR was designed for the longitudinal dynamics and the closed-loop system from reference velocity to longitudinal position was used to predict the EV's motion for each velocity level in the motion planner.

The conversion of the motion plan from road-aligned coordinates to global coordinates simulates the motion model using the decoupled controllers. This approach thus also makes it easy to separately tune how aggressively the reference velocity should be tracked, without affecting the lateral motion.

Integral Action

The LQRs guarantee stability but may suffer from stationary errors. This can be resolved by introducing integral action in the controller. A common approach is the Linear Quadratic Integral Regulator (LQI), where the state-space model is augmented with explicit states for the integrated output errors of the system [Young and Willems, 1972]. The LQI is then simply constructed by applying the regular LQR methodology to the augmented state-space model. For the motion planner, it is desirable to remove stationary errors in both the trajectory for the longitudinal velocity and the lateral positioning on the road. This was achieved by augmenting the longitudinal and lateral dynamics with integral states corresponding to these two outputs. For the

lateral dynamics, we get that

$$x_{k+1}^e = \begin{bmatrix} x_{k+1} \\ x_{k+1}^I \end{bmatrix} = \underbrace{\begin{bmatrix} A & 0 \\ -Ch_{MM} & I \end{bmatrix}}_{A^e} \begin{bmatrix} x_k \\ x_k^I \end{bmatrix} + \underbrace{\begin{bmatrix} B \\ 0 \end{bmatrix}}_{B^e} \delta_k + \underbrace{\begin{bmatrix} D \\ 0 \end{bmatrix}}_{D^e} d_k + \underbrace{\begin{bmatrix} 0 \\ Ih_{MM} \end{bmatrix}}_{G} r$$

$$= A^e x_k^e + B^e \delta_k + D^e d_k + Gr, \tag{7.2a}$$

$$y_k = \underbrace{\begin{bmatrix} C & 0 \end{bmatrix}}_{C^e} \begin{bmatrix} x_k \\ x_k^I \end{bmatrix} = C^e x_k^e, \tag{7.2b}$$

where h_{MM} is the sampling period of the motion model and I denotes the identity matrix of appropriate dimensions. The same extension was done analogously for the longitudinal dynamics. Since the lateral controller now has different structure, the gain-scheduling analysis in Section 5.2 needs to be adopted accordingly. The same conclusions still hold, but the state-space matrices have to be exchanged for their augmented counterparts.

Designing the Edge Weights in the Connectivity Graph

As discussed in Section 5.1, the edge weights of the connectivity graph $\mathcal{G}$ can be used as design parameters to ensure that the motion planner follows a desirable driving behavior. The final edge weights that were used for our driving experiments are illustrated in Figure 7.1. In the illustration, a heat map of the cost landscape is shown for the *nodes* of the graph. The value for a certain node is supposed to be interpreted as the weight of *all* edges pointing to this node. In the lateral dimension, it gets more expensive to move to a node that is located between the lanes. This strategy encourages the motion planner to find a path that stays in the center of the lanes and avoids staying between lanes. Along the time dimension, we gradually increase the edge weights. If the EV plans to pursue an overtaking, this will encourage the EV to switch lane as early as possible.

Another idea that was considered and evaluated was to also superpose ramp-like cost landscapes around obstacles in order to make the safety margins softer. This would then encourage the EV to stay further away from the obstacles but not force the EV to drastically reduce its speed if it, for instance, does not manage to turn as fast as the motion planner suggests. This approach requires a lot of careful design choices to be made to balance the relative weights of the edges. Since the weights in Figure 7.1 already make the motion planner avoid retardant overtakings, this turned out to be unnecessary while only adding complexity to the algorithm.

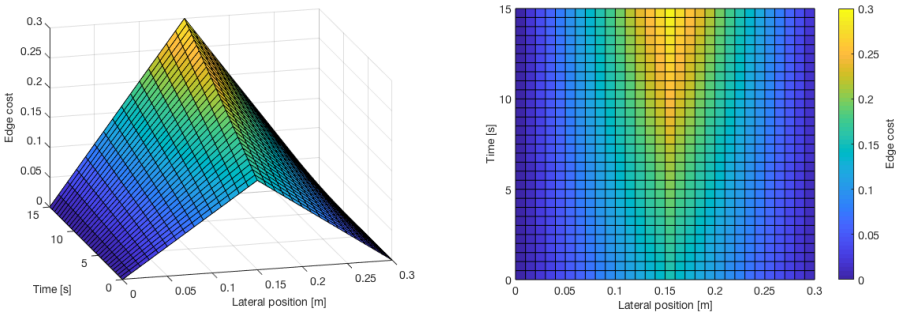


Figure 7.1 Illustration of the edge weights of the connectivity graph.

Graph Search

As mentioned in Chapter 5, the motion-planning problem results in finding a shortest path in the modified connectivity graph $\tilde{\mathcal{G}}$ with designed edge weights according to the previous section. Similarly as in [Berntorp et al., 2017], Dijkstra's algorithm was employed to solve the shortest-path problem in each planning step.

It is also possible to solve the shortest-path problem using dynamic programming by interpreting each time step in the planning horizon as a stage of a corresponding optimization problem. This was an approach that had previously been studied by MERL, and it was also investigated briefly for the motion-planning algorithm presented in this thesis. Dynamic programming is a more general solution framework and naturally is is not as efficient as Dijkstra's algorithm for finding a single shortest path in a graph. However, the more general framework allows for different interpretations and possible modifications that can prove useful in practice.

One extension that was considered was to introduce stochasticity in the shortest-path problem. Since there might be a mismatch in the model of the EV's motion, it is possible that the tracking controller is not able to move the EV to the target setpoint, but rather end up closer to some adjacent setpoint. This could be represented as a small probability of transitioning to the wrong node of the connectivity graph, which can then be accounted for by the expected cost (3.16) in the dynamic programming formulation. The problem of tuning all transition probabilities and balancing them with the edge weights in the connectivity graph would require a lot of careful design choices to be made. Therefore, it was not investigated further in this thesis. The discussion, however, illustrates the advantage of using different solution methods and interpreting the same problem from different perspectives.

Delay Compensation

The motion planner updates the motion plan with a fixed frequency. Since each motion plan requires a certain amount of computational time to compute, it will result in a slight delay between the start of the motion-planning computation and the time when the motion plan is published. As such, there will be a mismatch between the initial position of the motion plan and the actual position of the EV when it receives the motion plan. To account for this mismatch, a delay compensator is desirable. By allocating a well-defined time period for the motion-planning computations, we can ensure that the motion plan is published with a fixed rate. Then, by simulating the EV forward for a period of time that equals the allocated computational time, the motion plan can be computed using the forward-simulated position. As such, the initial positions of the motion plan and the EV are better synchronized. For our experiments, the allocated computational time was 0.07 seconds. The moving obstacles were also delay compensated accordingly.

7.3 Behavioral Layer

The behavioral layer of the decision making hierarchy illustrated in Figure 1.2 is not a main focus of this thesis, but it servers as a way of illustrating the capabilities of the proposed motion-planning algorithm. This section briefly presents the methods that have been used. A illustrative block diagram of the implemented behavioral layer is depicted in Figure 7.2. The red regions represent signs and obstacles in the surrounding of the EV. The obstacles can either be virtual OVs or obstacles detected using LIDAR. We only consider virtual signs that had been positioned at a specific global location on the test track.

Initial Processing

First, the global coordinates of all signs and obstacles were converted to the road-aligned coordinate frame. Next, all objects outside a Region of Interest (ROI) around the EV were discarded so that only relevant information for the current motion-planning problem was processed. This was done to not waste computational time that could result in unnecessary time delays.

Motion Prediction of Obstacles

The motion-planning algorithm is able to handle different types of obstacles, but requires motion predictions in road-aligned error coordinates with respect to the EV. Depending on the type of obstacle that was to be predicted, different prediction methods were used.

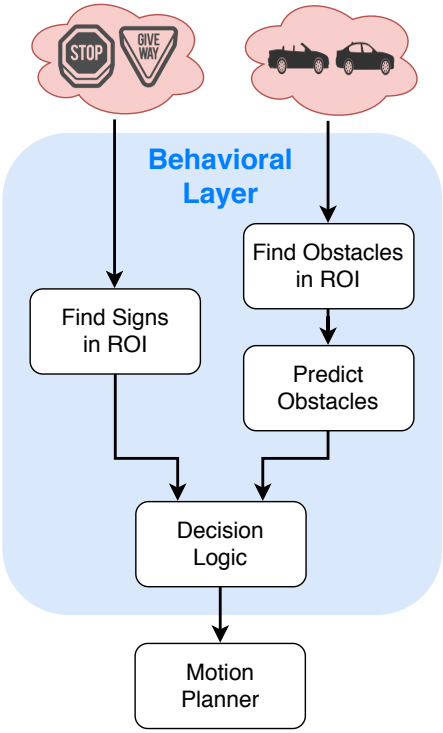


Figure 7.2 Simplified block diagram of the behavioral layer used for the driving experiments.

The obstacles detected using LIDAR contain data regarding position and velocity of the object. In general, there is no information available regarding what type of obstacle that has been detected, and hence a very simplistic prediction model was used. By simulating the obstacle with constant speed in the direction of the velocity vector, predictions in global coordinates were obtained. These predictions could then be converted to the road-aligned coordinate frame by kinematic relations.

If a virtual OV is moving along the same track as the EV, the predictions also need to follow the track accordingly. Therefore, by modeling the moving obstacles using the single-track model as described in Chapter 2, the predictions could be calculated directly in error coordinates by simulating the vehicle along the track with constant speed using state feedback.

For the intersection experiments, some obstacles were traveling along a different track compared to the EV. In this case, the predictions were carried out in global coordinates using the kinematic single-track model and a pure-pursuit controller [Paden et al., 2016]. The control law of a pure-pursuit controller is based on fitting a semi-circle through the vehicle's current pose to a point ahead of the vehicle along the reference path for some lookahead distance. If the path has a nonzero curvature, this type of controller will have a small steady-state error.

Decision Logic

The following heuristic decision rules were used for the signs and predicted obstacles in ROI:

Signs

- If there were any signs in ROI, the EV was put in sign mode and certain overtakings were prevented.
- Counters and the relative position of the signs were used to determine when a certain sign was considered active. For instance, a stop sign was considered active until the EV had come to a complete stop and stood still for at least one planning step.
- For an active stop sign, a stopping position and a velocity threshold was computed. Conceptually, the velocity threshold was chosen so that the EV would not surpass the stopping position within 3 seconds. This ensured that the motion planner would gradually reduce the speed while approaching an intersection.

Obstacles

- The obstacles were categorized as OVs on the main road, crossing OVs or LIDAR detected obstacles. Different safety margins were used depending on the type of obstacle, in which direction it was heading, and the current speed of the EV.
- Overtakings of crossing vehicles were prevented by increasing the lateral safety margin if the predicted position of the OV was on the main road.
- If there was an active stop sign, obstacles behind the stopping position were discarded until the sign became inactive.

8

Results and Discussion

In this chapter, the most important results and discussions from the sensor-fusion and motion-planning experiments are shown. As a result of the vast amount of collected data, not all results are shown in this section. Additional figures and graphs that are of interest are presented in Appendix A.

8.1 Sensor-Fusion Results

Simulation Study

We first provide simulation results for the EKF and UKF using simulated data from the Jellybean test track. The simulated data series is 300 seconds long and includes approximately 5 laps. Artificial measurement noise was generated to create a scenario that would resemble the characteristics of real driving experiments. The artificial noise was uncorrelated and Gaussian distributed with standard deviations as specified in Table 8.1. Additionally, a constant steering offset of -1.15 degrees was added to the true input signal. The filters were tuned separately to achieve as low mean-squared error as possible. The parameters of the process and measurement covariance matrices used in the final versions of the filters are presented in Tables 8.2 and 8.3, respectively.

In Figures 8.1 and 8.2, the state estimates from the EKF and the UKF are plotted for a single experiment. The first subplot depicts a trace of the global position estimates of the vehicle on the test track for the first 1.5 laps. For a selection of evenly spaced points on the first lap, 90% confidence ellipses are plotted centered at the estimated position. In the remaining 7 subplots, the state estimates are plotted as a function of time for the first 100 seconds together with 90% confidence intervals.

Table 8.1 Standard deviation of the artificial white noise that was used to corrupt the simulated data.

Measurement Source	Noise Standard Deviation	Unit
SLAM position	0.05	m
SLAM yaw	0.05	rad
Encoder velocity	0.1	m/s
Gyroscope	0.01	rad/s
Accelerometer	0.003	m/s^2

Table 8.2 The diagonal elements of the covariance matrix Q for the process noise.

Observer	x	y	ψ	v_{lng}	v_{lat}	δ	δ^b
EKF	$1 \cdot 10^{-7}$	$1 \cdot 10^{-7}$	$1 \cdot 10^{-7}$	$5 \cdot 10^{-7}$	$5 \cdot 10^{-7}$	$1 \cdot 10^{-5}$	$1 \cdot 10^{-8}$
UKF	$1 \cdot 10^{-6}$	$1 \cdot 10^{-6}$	$1 \cdot 10^{-6}$	$5 \cdot 10^{-7}$	$5 \cdot 10^{-7}$	$1 \cdot 10^{-5}$	$1 \cdot 10^{-8}$

Table 8.3 The diagonal elements of the covariance matrix R for the measurement noise.

Observer	x	y	ψ	v_{lng}	$\dot{\psi}$
EKF	0.05	0.05	0.05	0.01	0.01
UKF	0.1	0.1	0.1	5	0.01

The overall performances of both observers are good as the blue state estimates seem to be overlapping the true values in red throughout all state plots. The convergence of the state estimates is clearly seen by observing the rapid change in size of the confidence intervals. It is most prominent in the trajectory plots in the form of a rapid decrease in size of the confidence ellipses. The yaw estimates are very good for both the EKF and the UKF as their confidence intervals are very narrow. The confidence intervals of the longitudinal velocities are somewhat larger, but even so the estimates seemed to be rather good for both observers once the initial transients had faded off. The lateral velocity estimates are also reasonable when using the simulated data since the observers are able to follow the true values rather well. Generally, the UKF seemed to have more trouble converging to the true values for both the longitudinal and the lateral velocity. This behavior can probably be mitigated by further tuning of the covariance parameters.

The steering-angle estimates were also satisfactory as they are able to track the true input signal closely with narrow confidence intervals. Both observers were able to converge quickly to the applied steering offset of -1.15 degrees.

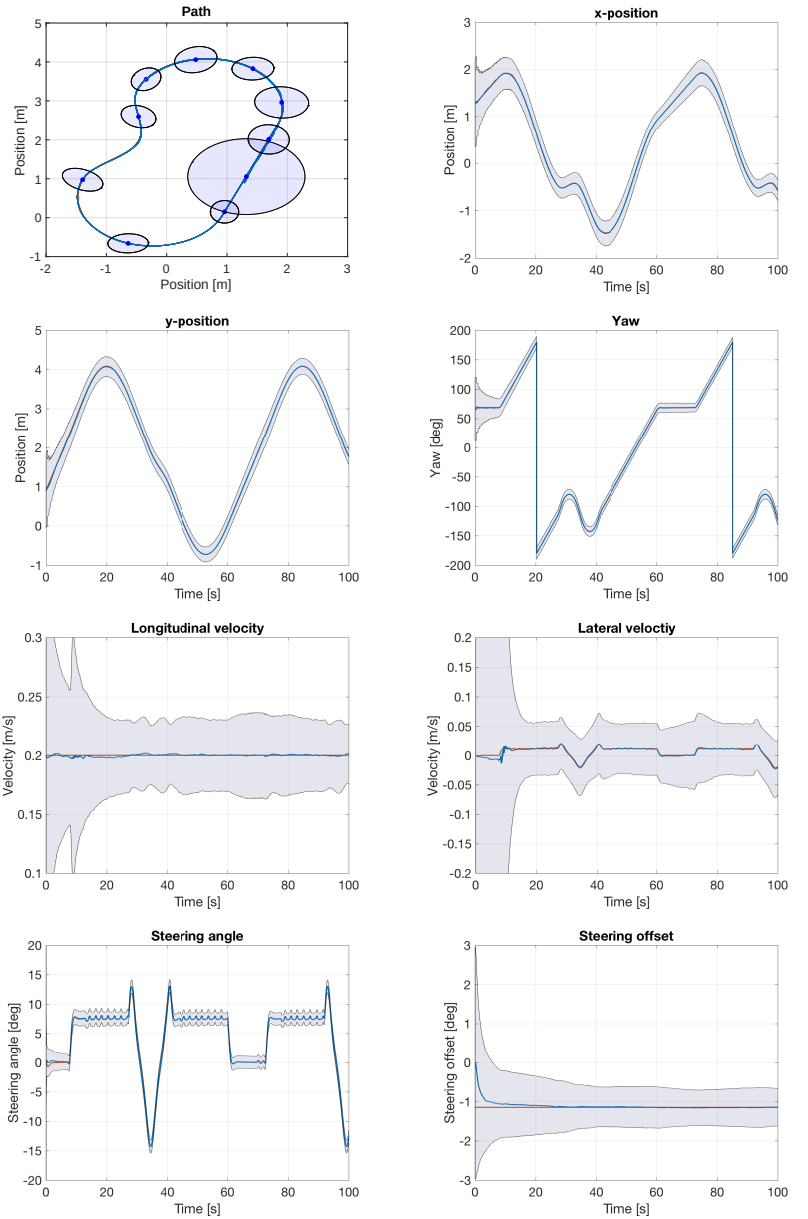


Figure 8.1 The estimated states (in blue) with 90 % confidence intervals for the EKF observer when using simulated data of the position, yaw, steering angle, longitudinal velocity, and longitudinal and lateral acceleration. The red lines represent the true values. Since the estimates are very close to the true values, the lines sometimes overlap and may be difficult to distinguish.

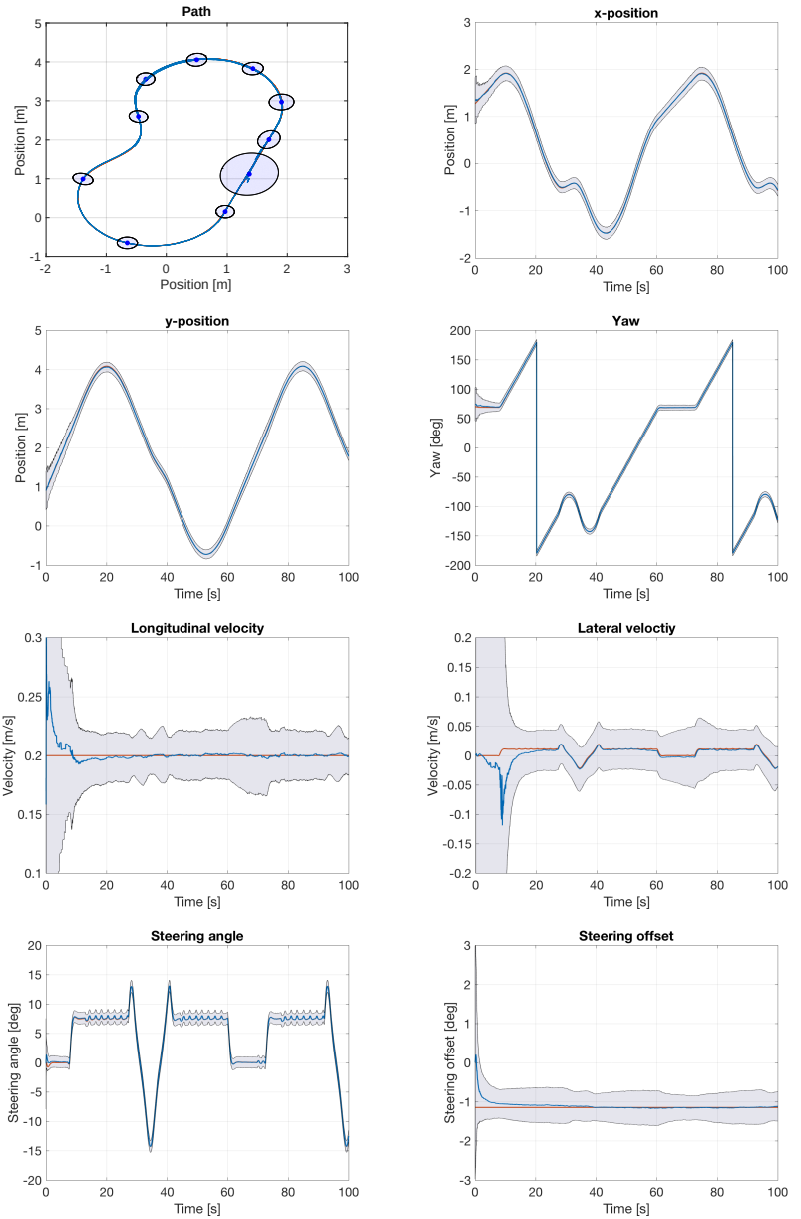


Figure 8.2 The estimated states of the UKF observer when using simulated data of the position, yaw, steering angle, longitudinal velocity, and longitudinal and lateral acceleration. The same notation is used as in Figure 8.1.

Table 8.4 Average MSE and STD for the state estimates of the EKF and UKF for a set of 100 simulation experiments.

	EKF		UKF		
State	MSE	STD	MSE	STD	Unit [$\cdot 10^{-4}$]
x	7.500	2.325	10.620	2.489	m
y	5.545	1.714	10.510	2.182	m
ψ	2.255	0.752	2.634	0.695	rad
v_{lng}	0.027	0.004	0.131	0.028	m/s
v_{lat}	0.113	0.029	1.292	0.333	m/s
δ	0.028	0.002	0.023	0.006	rad
δ^b	0.007	0.002	0.015	0.007	rad/s

In Table 8.4, the average and Standard Deviation (STD) of the Mean Squared Error (MSE) for each state are shown for a set of 100 simulation experiments. New artificial noise was generated for each experiment and the first 10 seconds of the results were discarded to reduce the impact of the initial transients. The average computational time per iteration was 98.6 μs for the EKF and 395.6 μs for the UKF. The results were obtained on a standard 2015 i5 2.7Ghz laptop and were clocked for the observer implementations in Matlab.

The conclusion of the simulation study was that the EKF was computationally faster, had less severe initial transients, and performed better after the filter had converged. Thus, the decision was made to use the EKF for the real-life driving experiments.

Driving Experiments

The parameter settings for the EKF observer used for the real-life driving experiments are presented in Tables 8.5 and 8.6.

Table 8.5 The diagonal elements of the covariance matrix for the process noise used in the real-life driving experiments.

Observer	x	y	ψ	v_{lng}	v_{lat}	δ	δ^b
EKF	$5 \cdot 10^{-3}$	$5 \cdot 10^{-3}$	$5 \cdot 10^{-3}$	$5 \cdot 10^{-3}$	$5 \cdot 10^{-3}$	$1 \cdot 10^{-2}$	$1 \cdot 10^{-7}$

Table 8.6 The diagonal elements of the covariance matrix for the measurement noise used in the real-life driving experiments.

Observer	x	y	ψ	v_{lng}	$\dot{\psi}$
EKF	$5 \cdot 10^{-3}$	$5 \cdot 10^{-3}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	1

Next follows the sensor-fusion results from the real-life driving experiments using open-loop actuator inputs. In Figure 8.3, the estimated path as well as x-position and y-position are plotted in blue. The SLAM measurements are plotted in red. One can spot the low update frequency of the SLAM measurements in the plots. It is most noticeable in the beginning of the x-position and y-position plots in the form of a linear interpolation between the first two points of the SLAM system. Another indication of the scarce SLAM measurement updates is the saw-tooth like behavior in the path plot when the vehicle is in a turn, illustrated in the zoomed-in plot in Figure 8.4. This behavior is owing to the observer adapting itself each time it receives a new SLAM measurement. Since there is a mismatch between the motion model used in the observer and the actual motion of the vehicle, we were not able to fully alleviate this behavior. However, the severity was greatly reduced by properly incorporating the steering offset into the process model of the observer. The deviation to the interpolated SLAM measurements is less than 0.01 m, which proved to be sufficient to achieve reasonable tracking performance for the autonomous driving experiments. The yaw estimates seemed to be following the SLAM measurements very well with a low noise.

The mismatch in the vehicle model is more prominent in the longitudinal velocity plots. The blue line represents the estimated longitudinal velocity and the red line is the recorded data from the wheel encoders. Although the signal is quite noisy, the observer is able to filter out some of the noise but not as much as one would desire. In the lateral velocity, steering angle, and steering offset plots, there are no measurements available to compare the estimated states with. The lateral velocity relies heavily on the accelerometer measurements that are very noisy. Together with the simplified vehicle model, it becomes very difficult to estimate this state. This fact resulted in estimated values that were significantly larger that what is physically possible. Hence, heuristics were introduced to reset the estimated lateral velocity to zero if the yaw rate of the vehicle was small enough. Consequently, this resulted in an estimate that was identically zero over the entire experiment.

The steering-angle estimate follows the rapid changes in control inputs and the observer is able to filter out noise quite well. Interestingly, it can be seen that even though a zero steering-angle control signal is sent between 20 s – 27 s, the estimated steering angle is different from zero. This is because of the inherent steering offset of the Hamster. The steering-offset plot can be seen to converge to a stationary value of around 0.6 degrees. The speed of the convergence is, however, somewhat slow.

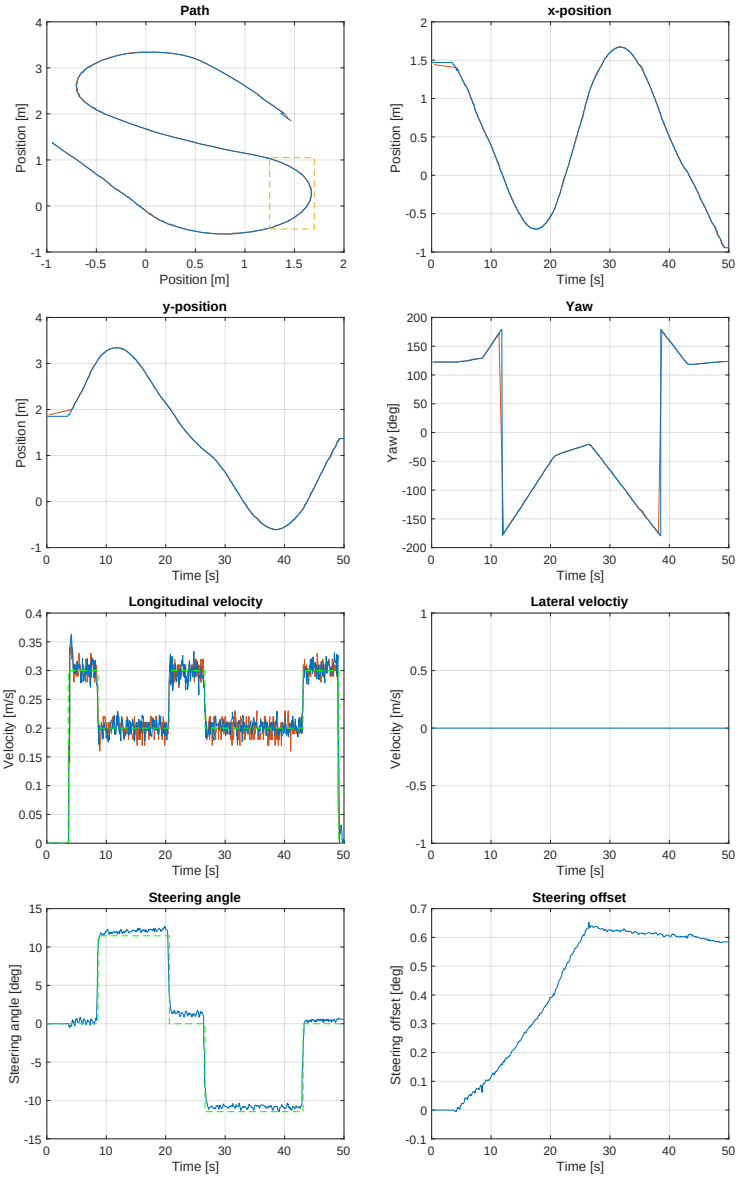


Figure 8.3 The estimated states (in blue) of the observer when using SLAM, IMU, and encoder measurements from the Hamster. The red lines represent the unfiltered sensor data for the states that can be measured directly. The dashed green lines represent the open-loop actuator commands that were used to drive the Hamster. Lastly, the dashed yellow rectangle in the path plot indicates the zoomed region that is depicted in Figure 8.4.

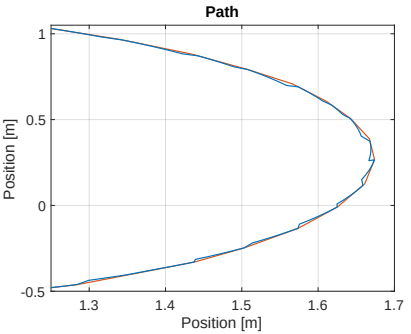


Figure 8.4 Zoomed-in plot of the region marked in yellow in Figure 8.3. The position estimates from the observer are shown in blue, and the interpolated SLAM measurements in red.

8.2 Motion-Planning Results

For the autonomous driving experiments, the implemented motion-planning algorithm presented in Section 5.2 was used. Some of the most important algorithm parameters are presented in Table 8.7.

Table 8.7 Parameter values of the motion planner.

Parameter	Value	Unit	Description
h_{MM}	0.1	s	Sampling time of motion model
h_{MP}	0.5	s	Sampling time of motion planner
T_p	15	s	Planning horizon
N	30	-	Planning horizon in samples (T_p/h_{MP})
M	31	-	Number of lateral discretization points
$v_{\max}$	0.4	m/s	Speed limit of the road
$v_{\min}$	0.12	m/s	Minimum velocity threshold
N_v	8	-	Number of discretized velocity levels[1]
Δv	0.04	m/s	Difference between velocity levels

Both the lateral position and the longitudinal velocity were discretized equidistantly. Since the lanes of the test track are 0.3 m wide and the end points of the discretization grid were chosen as the center lines of the two lanes, the lateral distance between two adjacent setpoints is 0.01 m. These parameter settings together with the tuned LQI parameters for the lateral dynamics result in gain scheduling intervals as presented in Table 8.8.

[1] Not including the zero velocity.

Table 8.8 Gain scheduling intervals of the motion planner.

Gain scheduling level	Value [m/s]
v_1^{GS}	0.4
v_2^{GS}	0.225
v_3^{GS}	0.155
v_4^{GS}	0.118

The number of nonzero elements of the connectivity graph $\mathcal{G}_v$ for the different longitudinal velocity levels are shown in Table 8.9. Here, the velocity levels have been associated with the LQI controller of the corresponding gains scheduling interval as discussed in Section 5.2. The sparsity is the fraction of zero-valued elements of $\mathcal{G}_v$ and the setpoint outdegree is the maximum number of out-neighbors of any node in $\mathcal{G}_v$.

Table 8.9 Properties of the connectivity graph $\mathcal{G}_v$.

v [m/s]	Nonzero elements	Sparsity [%]	Setpoint outdegree
0.4	4493	99.52	5
0.36	4493	99.52	5
0.32	4493	99.52	5
0.28	4493	99.52	5
0.24	4493	99.52	5
0.20	4493	99.52	5
0.16	2753	99.70	3
0.12	2753	99.70	3

The connectivity graphs for the first six and the last two velocity levels, respectively, are in fact identical. For the lower velocity levels, the EV can only reach the two most adjacent setpoints in one planning step, because of the imposed input constraints in $\mathcal{U}$. Since the connectivity graphs are very sparse, the graph-search problem can be computed efficiently in real-time.

In the following sections, a selection of results are provided from the real-life motion-planning experiments with the Hamster. Corresponding results from the simulation study can be found in Appendix A.

Tracking Performance

Figure 8.5 depicts traces of the EV's motion in the global fixed-frame coordinate system for two real-life driving experiments of 2 minutes. In (a), there are no virtual OVs present and thus the EV is driving along the preferred right-lane during the entire experiment. We see that the tracking

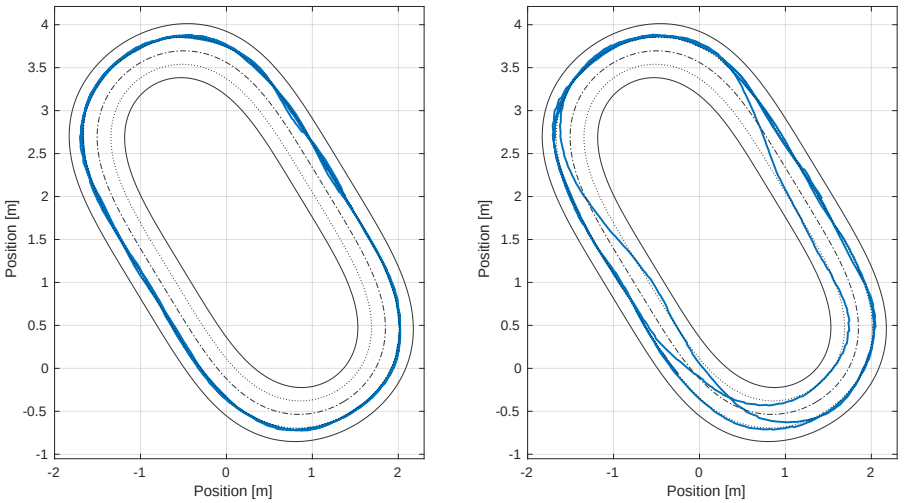
performance of the NMPC is good and that the controller manages to keep the estimated states close to the reference lane. This illustrates that the observer, the motion planner, and the low-level tracking controller successfully communicate over the ROS network.

In subplot (b), there are two virtual OVs present that drive in the right and left lane with speeds of 0.15 m/s and 0.2 m/s, respectively. In this case, the EV has to perform overtakes since it aims to drive at the nominal speed of 0.4 m/s. We see that the NMPC successfully tracks the motion plan during the lane switches, without significant overshoots and oscillations.

In Figure 8.6, the tracking error is depicted for the same two experiments as in Figure 8.5. The tracking error is computed as the minimum Euclidian distance between the EV's position and the path in the x-y-plane of the corresponding motion plan that was used as reference trajectory at the specific time instant. The tracking error has been low-pass filtered through a second order Butterworth filter with a cut-off frequency of 0.5 Hz. This filtering was done to make it easier to see the general trends and remove high-frequency noise. Overall, the tracking error is within a range of ± 1.5 cm in both cases, which is also reasonable if compared to the plots in Figure 8.5.

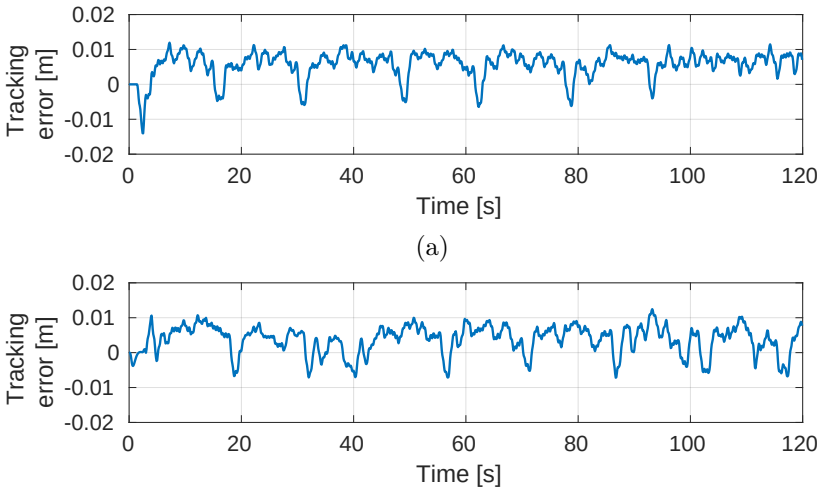
Even though the controller could be tuned to achieve even lower tracking errors, it might not improve the overall driving performance. The SLAM system can only provide pose measurements with a certain degree of accuracy that depends greatly on the tuning of the sensor as well as the quality of the LIDAR measurements.

The traces in Figure 8.5 are close enough to the centerlines of the lanes so that the EV would always stay on the road. In reality, however, the Hamster sometimes drove slightly outside the lane markers. This is because the SLAM system can be systematically off by a few centimeters for up to a couple of seconds. The SLAM serves as the only measurement that provides drift-free information about the position of the EV and these systematic errors are far from white noise. Hence, it was concluded that it is difficult to achieve better localization and tracking performance with the hardware setup used for this thesis. By including, for instance, GPS position data, these issues could possibly be alleviated.



(a) (b)

Figure 8.5 Traces of the EV's motion in the global coordinate system. In (a) there are no OVs present and in (b) there are two OVs present.



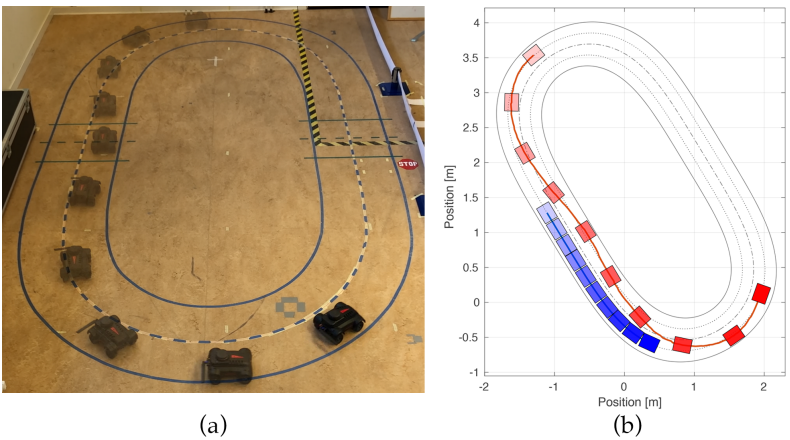
(a)

(b)

Figure 8.6 Tracking error as a function of time for the same experiments as in Figure 8.5.

Overtake with one OV

In Figure 8.7, a side-by-side comparison of an overtaking scenario is illustrated for a real-life driving experiment. Subfigure (a) shows a fused sequence of images from the RobotLab and subfigure (b) depicts the estimated position from the localization system. The blue rectangles show the motion of an OV that is driving with a constant velocity of 0.15 m/s, whereas the red rectangles represent the EV's motion. There is no OV visible in the real photos since it is a virtual OV that was simulated in Matlab. The overtake lasts for 15 seconds and the time evolution is illustrated with an increasing opacity. We see that the motion planner switches lane to overtake the slower driving OV and manages this in spite of the relatively strong curvature of the test track and possible mismatches in the motion model. An interesting observation at the apex of the lower turn is that the real-life EV is slightly outside the lane markers of the tack, which does not coincide with the corresponding position of the red virtual EV. The position of the virtual EV is given by the state estimates from the observer. The difference between the estimated position and actual position is most likely a result of the limitation in accuracy of the SLAM measurements.



(a) (b)

Figure 8.7 Overtaking scenario with one OV. In (a), a fused sequence of images is shown. In (b), the estimated pose of the EV and the motion of the virtual OV are illustrated.

To illustrate how the motion planner operates, the same overtaking scenario is also shown in Figure 8.8. The upper plot shows the reference velocity together with low-pass filtered velocity measurements from the wheel encoders. The motion planner manages to keep a constant reference velocity

during the overtake, owing to the appropriately designed edge weights of the connectivity graph.

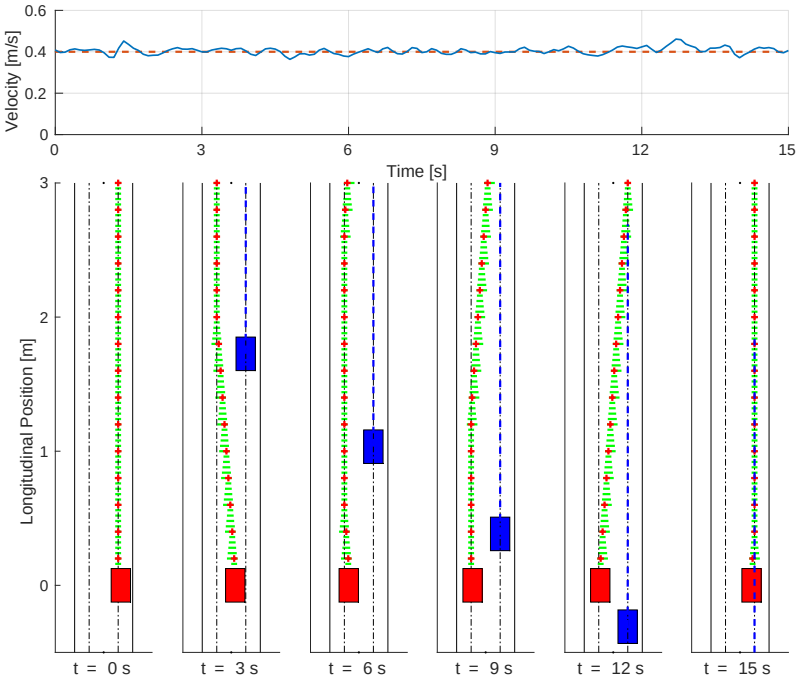


Figure 8.8 Overtaking scenario with one OV from an ROI perspective.

The lower plots show a sequence of six time instants from an ROI perspective around the EV. Since the curvature of the road is considered as a disturbance in the road-aligned coordinate system, the road appears to be straight in this case. The red crosses are the trajectory of setpoints of the current motion plan. The green lines are projections of the positively invariant sets onto the lateral dimension. The increasing width of the green lines between two distinct setpoints illustrate the gradually growing positively invariant sets for the intermediate time steps of the upsampled motion model, as governed by (5.12). The dashed blue line corresponds to the predicted motion of the OV.

When the EV detects the virtual obstacle, the motion planner schedules a lane switch. Eventually, it becomes possible to safely plan a trajectory back to the desired right-lane again within the prediction horizon T_p. By carefully

studying ROI plots for even more time instants, it can be seen that the planned trajectory of lateral setpoints is more or less the same during the overtake. This observation means that the EV is actually able to follow the provided motion plan and that the implicit feedback of the receding horizon approach does not change the shape of the motion plan excessively.

Overtake with two OVs

In Figure 8.9, an ROI plot is shown for an overtaking scenario with two OVs. The OVs in the right and left lane have constant velocities of 0.15 m/s and 0.2 m/s, respectively. In this case, the motion planner has to reduce the reference velocity in order to ensure that the EV does not enter the critical zones around the obstacles. Since the EV aims to drive at the nominal speed of 0.4 m/s and the OV in the left lane has a higher speed, the motion planner switches lane to perform an overtaking of the OV in the right lane. Once it is possible to safely plan a trajectory back to the preferred right lane, this becomes the intended motion plan. In the ROI plot at $t = 5$ s, one can see that the simple pure-pursuit controller has some difficulties keeping the OVs at the center of the lanes. This, however, does not pose any problems for the motion planning of the EV.

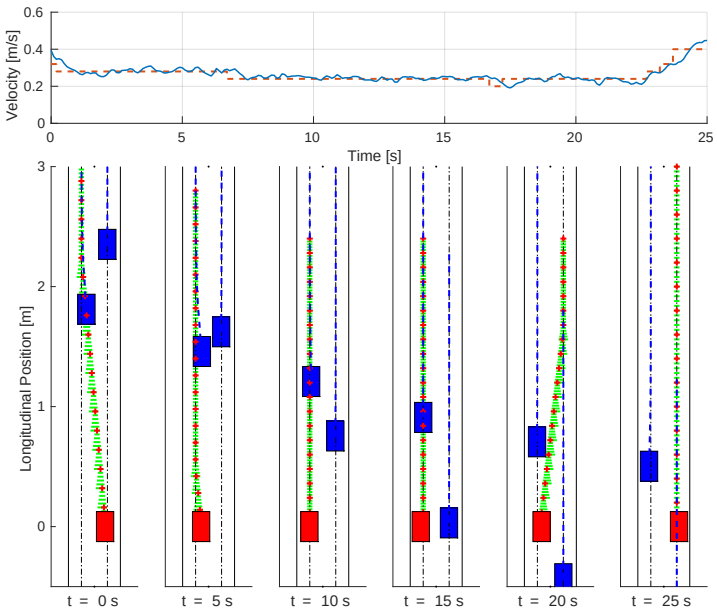


Figure 8.9 Overtaking scenario with two OVs from an ROI perspective.

Intersection with crossing OVs

The following two examples aim to illustrate that the implemented motion-planning algorithm is also capable of handling scenarios in an urban driving environment. In Figure 8.10, the EV approaches one of the intersections of the Intersection test track, where the EV has an obligation to come to a complete stop. In this case, two OVs are driving along the crossing road and the EV also needs to give way to these vehicles.

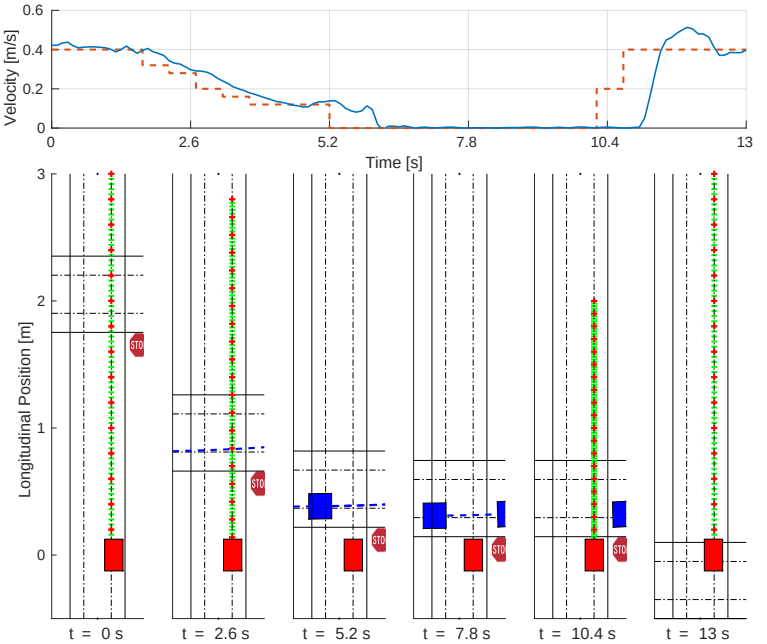


Figure 8.10 Intersection scenario with a stop sign.

Owing to the implemented heuristic decision logic in the behavioral layer, the motion planner gradually reduces the velocity and slowly approaches the stop sign. One can see that it takes some additional time before the EV comes to a complete stop in comparison to the reference velocity profile. A similar type of delay is also seen when the EV starts moving once the intersection is clear. This is because of delays in the actuator system of the Hamster and limitations of the current implementation of the NMPC tracking controller. However, since there are safety margins to the intersecting road, the EV still stops at an appropriate position. The heuristics prevent the EV from performing an overtaking maneuver of the crossing vehicles and hence wait until the safety zones around all OVs have left the area of intersection.

In Figure 8.11, the same intersection experiment as in Figure 8.10 is illustrated with a fused sequence of images. There are no OVs visible since they are simulated virtually. We can see that the EV slowly approaches the stop sign and completely stops by the overlapping EV snapshots in front of the intersection.

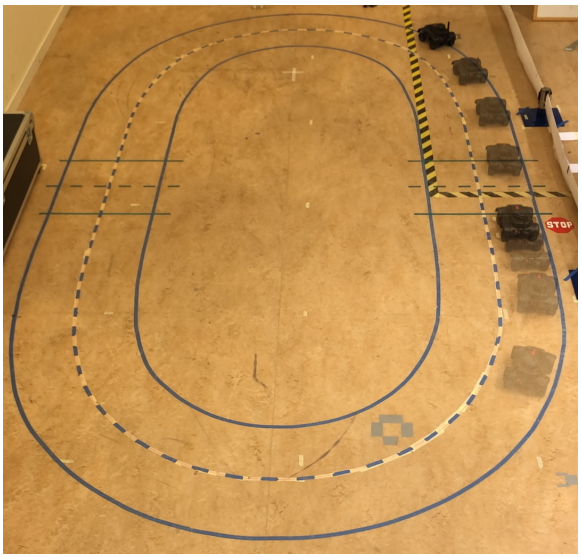


Figure 8.11 A fused sequence of images representing the time progression of the intersection scenario with a stop sign.

Figure 8.12 depicts a similar scenario as in Figure 8.10, but with one OV in the right lane that is standing still. In this case the motion planner schedules a lane switch in order to overtake the OV on the main road, while still respecting the decision logic from the behavioral layer regarding the stop sign. The reason why the EV has to initially reduce its velocity to switch lane is because the OV is detected relatively late owing to the strong curvature of the main road close to the intersection.

The intersection experiments show that the proposed algorithm is capable of computing feasible motion plans over the entire range of reference velocity levels that is considered in this thesis. The results suggest that the motion planner can be used not only for lane changing maneuvers on highways, but also for more complex urban driving scenarios.

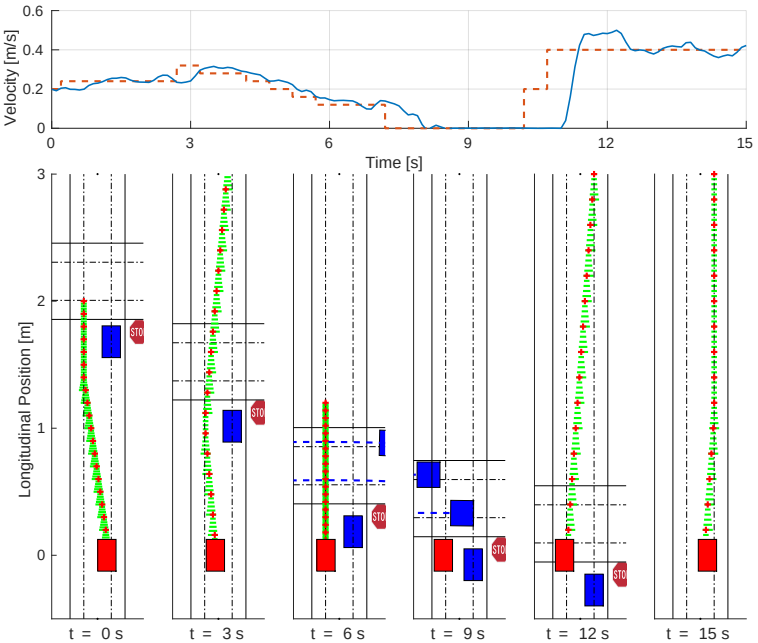
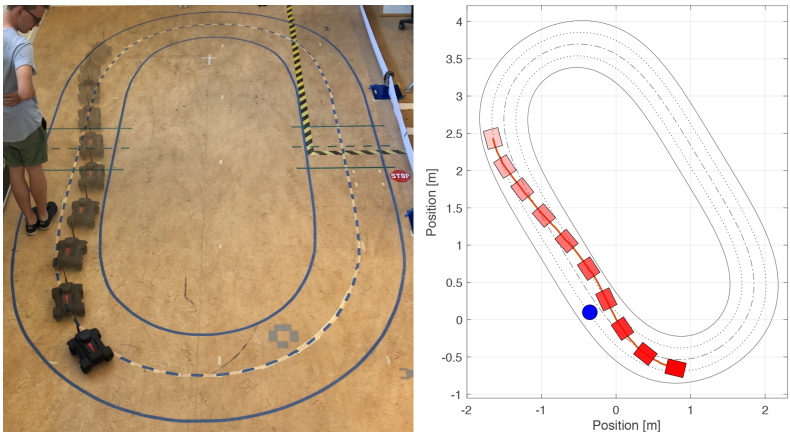


Figure 8.12 Intersection scenario with a stop sign and overtaking of one OV.

Obstacle Detection using LIDAR

In this section, results from two LIDAR obstacle detection experiments are presented. In the first experiment, a static real obstacle is placed in front of the EV. The second experiment is an intersection scenario with a stop sign and a real obstacle moving along the crossing road. The obstacle in both experiments was portrayed by a human.

As seen in Figure 8.13, the LIDAR on the EV is able to detect the static obstacle. Using the visualization tool `rviz` for ROS [Hershberger et al., 2018], the map, the pose of the EV, the detected obstacle, and the laser scan from the LIDAR can be visualized in real-time. A snapshot of the `rviz` window during the static obstacle experiment is shown in Figure 8.14. It can be seen that multiple obstacles were detected, for instance, corners of some nearby objects. All detected obstacles, except for the human, are filtered out when parsed to the motion planner. This is because only the detected obstacles in the interior of the map were considered by the motion planner. The calculated motion plan switches lane to avoid the obstacle, and return back once it has passed the obstacle. This behavior is very similar to results using virtual OVs.



(a) (b)

Figure 8.13 Overtaking scenario with a LIDAR-detected static obstacle. In (a), a fused motion sequence is shown. Subplot (b) illustrates the pose estimates of the EV from the localization system together with the detected obstacle.

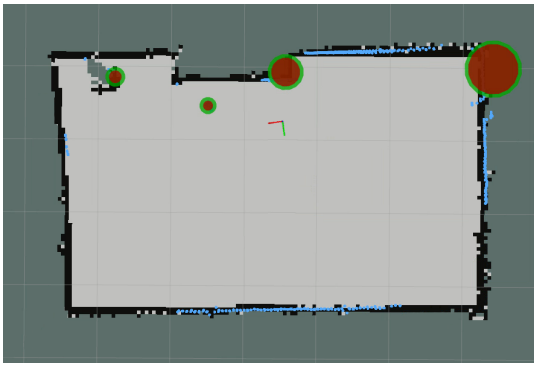


Figure 8.14 Snapshot of the `rviz` window during the LIDAR detection experiment with one static obstacle. The small coordinate frame represents the estimated position of the EV, where the red axis is in the heading direction of the vehicle. The LIDAR-detected obstacles are illustrated by red circles with green borders. The view is rotated 90 ° clockwise with respect to subfigure (a) in Figure 8.13.

The intersection experiment with a real moving obstacle is shown in Figure 8.15. The increase in opacity is once again used to illustrate the time progression during the experiment. One can see that the EV is able to safely stop at the stop sign and wait until the obstacle has passed, before moving on.

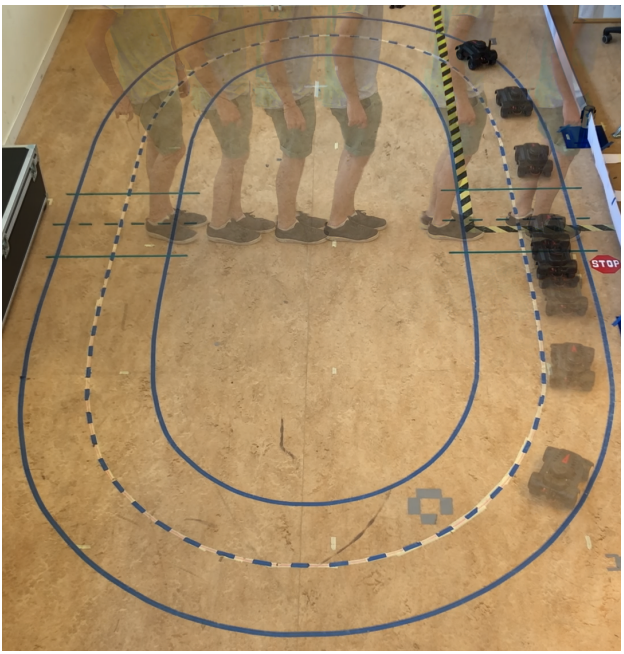


Figure 8.15 Fused sequence of images from the intersection experiment with a LIDAR-detected moving obstacle.

In Figure 8.16, an ROI plot is shown for the same intersection scenario as illustrated in Figure 8.15. From the velocity plot, it is clear that the motion planner is capable planning a smooth deceleration, stop at the stop sign, and continuing on once the intersection is clear. The prediction of the detected obstacle is seen in the ROI plots for the time instants from 2.2 s to 8.8 s. Even though the obstacle is moving along the crossing road in reality, the predicted direction of motion is changing in time. As such, it can be deduced that the velocity estimates are not perfect. There are multiple factors that can affect the velocity estimates; the LIDAR scans, the shape of the moving obstacle, and the fact that the EV itself is moving, to name a few.

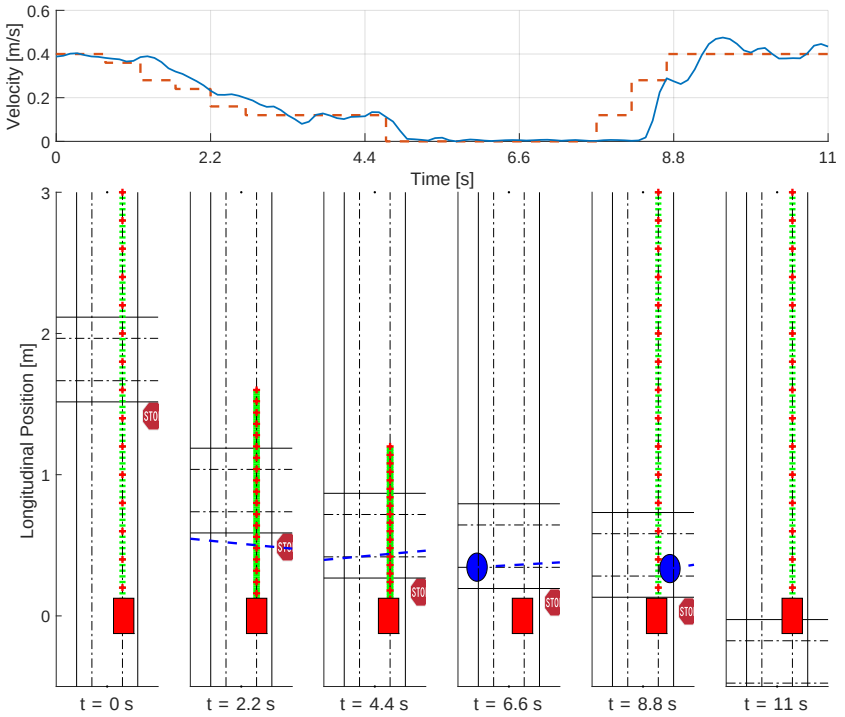


Figure 8.16 Intersection scenario with a stop sign and a LIDAR-detected moving obstacle.

The obstacle detection experiments show that it is possible to detect both static and moving obstacles using LIDAR scans. The velocity estimates are, however, not very reliable. Even so, the detected obstacles can be handled by the motion planner to produce safe reference trajectories.

It was noticed that the position estimates of the detected obstacles were more inaccurate at places with strong curvature along the track. In some cases, it could be seen that even though the obstacle was only blocking one lane, its position estimate would be located in the center of the road, causing the EV to brake. We believe that the sharp turns and the resulting rotation of the EV cause difficulties for the obstacle-detector package to accurately position the obstacles. Since obstacle detection was not a main focus of this thesis, it was not further investigated.

9

Conclusions and Future Work

This chapter summarizes the work that has been performed in this thesis. Conclusions from the sensor-fusion and motion-planning experiments are presented and possible improvements and future work are discussed.

9.1 Conclusions

In this thesis, a motion-planning algorithm based on positively invariant sets has been studied and implemented on a ground-vehicle robot platform. To achieve this objective, a localization system was first designed to estimate the state of the vehicle using available sensor data, which could then be used by the planning and control algorithms. By incorporating a gain-scheduling approach into the motion planner, guaranteed safe reference trajectories that avoid nearby obstacles could be computed for time-varying velocities of the vehicle.

Localization System

To estimate the pose and velocity of the vehicle, an EKF and an UKF were considered. The EKF was shown to perform better in simulations, was computationally faster, and was therefore implemented on the actual robot platform. From the simulation results in Figure 8.1 and real driving results in Figure 8.3, it was seen that the EKF was able to robustly estimate the vehicle's states, whilst detecting and rejecting outliers, and monitoring the filter to prevent divergence issues. Even though the motion model did not fully capture the dynamics of actual vehicle, the difference between the estimates and the much more infrequent SLAM measurement was very small, as illustrated in Figure 8.4.

Motion Planning

The results from the overtaking experiments in Figures 8.8 and 8.9 showed that the motion-planning algorithm was able to calculate feasible motion plans that can handle overtaking scenarios with one and two OVs for time-varying velocities. Through successful intersection experiments involving signs, as illustrated in Figure 8.12, it was shown that the motion-planning algorithm is able to handle more complex urban driving scenarios through the usage of higher-level decision logics. Lastly, it is concluded from the obstacle detection results in Figure 8.13, that an obstacle detection and avoidance system can be implemented using the laser scans from the onboard LIDAR.

To reflect back on the problem formulation stated in Section 1.3, the motion planner, together with the tracking controller and localization system, is able to safely plan, navigate, and maneuver the ground-vehicle robot platform for different driving scenarios. The proposed motion-planning algorithm with the extended gain-scheduling approach allows the vehicle to safely navigate over the entire interval of considered velocities. Through heuristic rules in the behavioral layer, it was shown that the motion-planning algorithm can handle new driving scenarios that had not been tested previously. Hence, we can conclude that the motion-planning algorithm based on positively invariant set is a viable choice of algorithm for calculating safe motion plans, with more driving scenarios to be explored.

9.2 Future Work

Motion planning and autonomous driving in general are difficult control problems and there are many issues that need to be addressed before a fully automated car can enter production stage. In this section, possible areas of future work related to this thesis are suggested.

Localization System

The bias estimation approach adopted in this thesis assumes that the driving experiments are short enough so that the IMU sensor drifts are negligible. For real applications, these bias terms should be estimated online to ensure that the performance does not degrade with time. One solution would be to use more sophisticated sensor models and separate the bias estimation problem into separate, designated, observers. By including sensor data from additional measurement equipment, it could also be possible to use the current observer structure with more bias terms, without having problems with possible unobservability.

For outdoor navigation applications without static environments, it is not feasible to rely on pose estimates from the SLAM system. A natural approach would be to instead include GPS data, which is a straightforward extension of the current observer implementation.

Edge Weights of the Connectivity Graph

The edge weights of the connectivity graphs serve as design parameters that can be used to achieve desirable driving behavior. This question has been addressed briefly in this thesis and the investigation can be continued further. For instance, transition costs that emphasize fuel consumption could be used to encourage energy-efficient maneuvers that are more environmentally friendly.

Gain-Scheduling Analysis

In Section 5.2, a bisection search method was described for finding the gain-scheduling intervals of the motion planner. It has been verified numerically that if the S-matrix is negative definite for some deviated velocity, it is also negative definite for all velocities up to the velocity used to compute the LQR gain. Future work would include to prove this theoretically using the known structure of the matrices of the motion model.

Obstacle Detection Using Camera

The obstacle detection system based on LIDAR data has been used to illustrate that the motion-planning algorithm can be used for real-life obstacle avoidance. As discussed in Section 8.2, poor position estimates of the detected obstacles can give rise to undesirable velocity adjustments. By including data from the onboard camera and utilizing image analysis methods, it should be possible to achieve a higher precision for obstacles that are located in front of the EV. Since the camera is pointed in the heading direction of the EV, this should also have the largest impact on the reference trajectories from the motion planner.

Vehicle Parameters

The cornering stiffness values in the dynamic single-track motion model have been estimated by studying the behavior of a virtual vehicle in simulations. A better alternative would be to estimate these vehicle parameters through empirical experiments on the actual robot, for instance, using methods described in [Berntorp and Di Cairano, 2018; Svendenius, 2007]. This could potentially provide more realistic motion trajectories to the tracking controller, which would result in smaller model errors and hence better driving performance.

Bibliography

Agarwal, P. K., T. Biedl, S. Lazard, S. Robbins, S. Suri, and S. Whitesides (2002). "Curvature-constrained shortest paths in a convex polygon". *SIAM Journal on Computing* **31**:6, pp. 1814–1851.

Amicucci, G. L., S. Monaco, and D. Normand-Cyrot (1997). "Control lyapunov stabilization of affine discrete-time systems". In: *IEEE Conference on Decision and Control*. Vol. 1, 923–924 vol.1. DOI: 10.1109/CDC.1997.650761.

Anderson, B. D. O. and J. B. Moore (1989). *Optimal Control: Linear Quadratic Methods*. 1st ed. Prentice Hall Inc., Engelwood Cliffs, NJ, United States.

Åström, K. J. and B. Wittenmark (1997). *Computer-Controlled Systems: Theory and Design*. Prentice Hall Inc., Upper Saddle River, NJ, United States.

Åström, K. J. and B. Wittenmark (2013). *Adaptive Control*. Dover Publications, Inc., Mineola, NY, United States.

Berntorp, K. and S. Di Cairano (2018). "Tire-stiffness and vehicle-state estimation based on noise-adaptive particle filtering". *IEEE Transactions on Control Systems Technology*, pp. 1–15.

Berntorp, K., A. Weiss, C. Danielson, I. V. Kolmanovsky, and S. D. Cairano (2017). "Automated driving: Safe motion planning using positively invariant sets". In: *IEEE International Conference on Intelligent Transportation Systems*, pp. 1–6. DOI: 10.1109/ITSC.2017.8317672.

Berntorp, K. (2015). "Feedback particle filter: Application and evaluation". In: *IEEE International Conference on Information Fusion*, pp. 1633–1640.

Berntorp, K. and S. Di Cairano (2016). "Particle filtering for online motion planning with task specifications". In: *IEEE American Control Conference*, pp. 2123–2128.

Berntorp, K., T. Hoang, R. Quirynen, and S. Di Cairano (2018a). "Autonomous vehicle control design, implementation, and verification using scaled road vehicles". In: *IEEE Conference on Control Technology and Applications*. Accepted, Invited Paper.

Berntorp, K., C. Danielson, A. Weiss, and S. Di Cairano (2018b). "Positive invariant sets for safe integrated vehicle motion planning and control". In: *IEEE Conference on Decision and Control*. Submitted, Invited Paper.

Bertsekas, D. P. (2005). *Dynamic Programming and Optimal Control*. 3rd ed. Athena Scientific, Belmount, MA, United States.

Betts, J. T. (1998). "Survey of numerical methods for trajectory optimization". *Journal of Guidance, Control, and Dynamics* **21**:2, pp. 193–207.

Boissonnat, J.-D. and S. Lazard (1996). "A polynomial-time algorithm for computing a shortest path of bounded curvature amidst moderate obstacles". In: *ACM Symposium on Computational Geometry*, pp. 242–251.

Borrelli, F., M. Baotic, A. Bemporad, and M. Morari (2001). "Efficient on-line computation of explicit model predictive control". In: *IEEE Conference on Decision and Control*. Vol. 2, pp. 1187–1192.

Buehler, M., K. Iagnemma, and S. Singh (2007). *The 2005 DARPA grand challenge: The great robot race*. Vol. 36. Springer Science & Business Media, New York, NY, United States.

Buehler, M., K. Iagnemma, and S. Singh (2009). *The DARPA urban challenge: Autonomous vehicles in city traffic*. Vol. 56. Springer, Heidelberg, Germany.

Burton, D, A Delaney, S Newstead, D Logan, and B Fildes (2004). "Evaluation of anti-lock braking systems effectiveness". *Research Report 04/01, ARRB Group Ltd*.

CogniTeam (2018). *The hamster*. URL: `http://cogniteam.com/hamster5.html` (visited on 04/29/2018).

Como, G. and F. Fagnani (2018). *Lecture Notes in Network Dynamics*. 1st ed. Department of Automatic Control, Lund University.

Danielson, C., A. Weiss, K. Berntorp, and S. D. Cairano (2016). "Path planning using positive invariant sets". In: *IEEE Conference on Decision and Control*, pp. 5986–5991. DOI: `10.1109/CDC.2016.7799188`.

Dijkstra, E. W. (1959). "A note on two problems in connexion with graphs". *Numerische Mathematik* **1**:1, pp. 269–271.

Doucet, A., N. De Freitas, and N. Gordon (2001). *An introduction to sequential Monte Carlo methods*. Springer, New York, NY, United States, pp. 3–14.

Gerkey, B. P. (2018a). *`amcl` package summary*. URL: `http://wiki.ros.org/amcl` (visited on 04/01/2018).

Gerkey, B. P. (2018b). *gmapping package summary*. URL: http://wiki.ros.org/gmapping (visited on 04/01/2018).

Glad, T. and L. Ljung (2000). *Multivariable and Nonlinear Methods*. 1st ed. Taylor and Francis, New York, NY, United States.

Goldberg, A. V. and C. Harrelson (2005). "Computing the shortest path: A search meets graph theory". In: *ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, pp. 156–165.

Guldner, J., H.-S. Tan, and S. Patwardhan (1996). "Analysis of automatic steering control for highway vehicles with look-down lateral reference systems". *Vehicle System Dynamics* **26**:4, pp. 243–269.

Gustafsson, F. and G. Hendeby (2012). "Some relations between extended and unscented Kalman filters". *IEEE Transactions on Signal Processing* **60**:2, pp. 545–555. ISSN: 1053-587X. DOI: 10.1109/TSP.2011.2172431.

Gustafsson, F. (2010). *Statistical Sensor Fusion*. 1st ed. Studentlitteratur AB, Lund, Sweden.

Hart, P. E., N. J. Nilsson, and B. Raphael (1968). "A formal basis for the heuristic determination of minimum cost paths". *IEEE Transactions on Systems Science and Cybernetics* **4**:2, pp. 100–107.

Hershberger, D., D. Gossow, and J. Faust (2018). *rviz Package Summary*. URL: http://wiki.ros.org/rviz (visited on 05/15/2018).

Holst, A. and V. Ufnarovski (2014). *Matrix Theory*. Studentlitteratur AB, Lund, Sweden.

Iggidr, A. and B. Mohammed (1996). "Stability of Discrete-Time Systems: New Criteria and Applications to Control Problems". *Research Report RR-3003, INRIA*.

Julier, S. J. and J. K. Uhlmann (2004). "Unscented filtering and nonlinear estimation". *Proceedings of the IEEE* **92**:3, pp. 401–422. ISSN: 0018-9219. DOI: 10.1109/JPROC.2003.823141.

Karaman, S. and E. Frazzoli (2010). "Optimal kinodynamic motion planning using incremental sampling-based methods". In: *IEEE Conference on Decision and Control*, pp. 7681–7687.

Karaman, S. and E. Frazzoli (2011). "Sampling-based algorithms for optimal motion planning". *The International Journal of Robotics Research* **30**:7, pp. 846–894.

Khalil, H. K. (2002). *Nonlinear Systems*. 3rd ed. Pearson Education International Inc., Harlow, Essex, United Kingdom.

Lavalle, S. M. (1998). *Rapidly-Exploring Random Trees: A New Tool for Path Planning*. Tech. rep. 98-11, Computer Science Department, Iowa State University.

Lefferts, E. J., F. L. Markley, and M. D. Shuster (1982). "Kalman filtering for spacecraft attitude estimation". *Journal of Guidance, Control, and Dynamics* **5**:5, pp. 417–429.

Lewis, F. L., D. Vrabie, and V. L. Syrmos (2012). *Optimal Control*. 3rd ed. John Wiley & Sons, Inc., Hoboken, NJ, United States.

Liebemann, E., K Meder, J Schuh, and G Nenninger (2004). *Safety and performance enhancement: The Bosch electronic stability control (ESP)*. Tech. rep. No. 2004-21-0060. SAE Technical Paper.

Likhachev, M., G. Gordon, and S. Thrun (2004). "ARA*: Anytime A* with provable bounds on sub-optimality". In: *Advances in Neural Information Processing Systems 16*. MIT Press, pp. 767–774.

Likhachev, M., D. Ferguson, G. Gordon, A. T. Stentz, and S. Thrun (2005). "Anytime dynamic A*: An anytime, replanning algorithm." In: *AAAI International Conference on Automated Planning and Scheduling*, pp. 262–271.

Liu, J. and M. West (2001). "Combined parameter and state estimation in simulation-based filtering". In: *Sequential Monte Carlo Methods in Practice*. Springer, New York, NY, United States, pp. 197–223.

Maciejowski, J. M. (2002). *Predictive Control: With Constraints*. Pearson Education Inc., Harlow, Essex, United Kingdom.

Madgwick, S. (2010). "An efficient orientation filter for inertial and inertial/magnetic sensor arrays". *Report X-io and University of Bristol (UK)*.

Matlab (2018). *Generate C and C++ code from Matlab code*. URL: `https://se.mathworks.com/products/matlab-coder.html` (visited on 03/25/2018).

Merwe, R. V. der and E. A. Wan (2001). "The square-root unscented Kalman filter for state and parameter-estimation". In: *IEEE International Conference on Acoustics, Speech, and Signal Processing*. Vol. 6, pp. 3461–3464. DOI: 10.1109/ICASSP.2001.940586.

NHTSA (2017). *Quick facts 2015*. URL: `https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812348` (visited on 04/29/2018).

NY Times (2016). *Tesla Upgrades Autopilot in Cars on the Road*. URL: `https://www.nytimes.com/2016/09/24/business/tesla-upgrades-autopilot-in-cars-on-the-road.html` (visited on 04/29/2018).

NY Times (2018a). *Fatal Tesla Crash Raises New Questions about Autopilot System*. URL: `https://www.nytimes.com/2018/03/31/business/tesla-crash-autopilot-musk.html` (visited on 04/15/2018).

NY Times (2018b). *Self-Driving Uber Car Kills Pedestrian in Arizona, Where Robots Roam*. URL: https://www.nytimes.com/2018/03/19/technology/uber-driverless-fatality.html (visited on 04/15/2018).

Paden, B., M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli (2016). "A survey of motion planning and control techniques for self-driving urban vehicles". *IEEE Transactions on Intelligent Vehicles* **1**:1, pp. 33–55.

Picard, J. (1991). "Efficiency of the extended Kalman filter for nonlinear systems with small noise". *SIAM Journal on Applied Mathematics* **51**:3, pp. 843–885. ISSN: 00361399. URL: http://www.jstor.org/stable/2102052.

Polak, E. (1973). "An historical survey of computational methods in optimal control". *SIAM Review* **15**:2, pp. 553–584.

Przybyla, M. (2017). *The obstacle_ detector package*. URL: https://github.com/tysik/obstacle_detector (visited on 04/27/2018).

Quigley, M., K. Conley, B. Gerkey, et al. (2009). "ROS: an open-source robot operating system". In: *IEEE International Conference on Robotics and Automation, Open-Source Software Workshop*.

Quirynen, R., K. Berntorp, and S. Di Cairano (2018). "Embedded optimization algorithms for steering in autonomous vehicles based on nonlinear model predictive control". In: *IEEE American Control Conference*. Accepted Paper.

Rajamani, R. (2011). *Vehicle Dynamics and Control*. Springer Science & Business Media, New York, NY, United States.

Rawlings, J. B. and D. Q. Mayne (2009). *Model predictive control: Theory and design*. Nob Hill Pub., Madison, WI, United States.

Reif, J. H. (1979). "Complexity of the mover's problem and generalizations". In: *IEEE Symposium on Foundations of Computer Science*, pp. 421–427. DOI: 10.1109/SFCS.1979.10.

Rosenfeld, A., N. Agmon, O. Maksimov, and S. Kraus (2017). "Intelligent agent supporting human–multi-robot team collaboration". *Artificial Intelligence* **252**, pp. 211–231.

ROS.org (2018). *About ROS: History*. URL: http://www.ros.org/history/ (visited on 05/10/2018).

Sinay, M., N. Agmon, O. Maksimov, S. Kraus, and D. Peleg (2017). "Maintaining communication in multi-robot tree coverage". In: *AAAI International Joint Conference on Artificial Intelligence*, pp. 4515–4522.

SORAVS (2014). "Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems". *SAE J3016*.

Svendenius, J. (2007). *Tire modeling and friction estimation.* PhD Dissertation TFRT–1077–SE, Department of Automatic Control, Lund University.

Thrun, S., M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, et al. (2006). "Stanley: The robot that won the DARPA grand challenge". *Journal of Field Robotics* **23**:9, pp. 661–692.

Tilton, A. K., S. Ghiotto, and P. G. Mehta (2013). "A comparative study of nonlinear filtering techniques," in: *IEEE International Conference on Information Fusion*.

Trafikverket (2017). *Vision zero academy.* URL: `https://www.trafikverket.se/en/startpage/operations/Operations-road/vision-zero-academy/` (visited on 04/29/2018).

Trawny, N. and S. I. Roumeliotis (2005). *Indirect Kalman filter for 3D attitude estimation.* Tech. rep. 2, Department of Computer Science & Engineering, University of Minnesota.

Ungarala, S., E. Dolence, and K. Li (2007). "Constrained extended Kalman filter for nonlinear state estimation". *IFAC Proceedings Volumes* **40**:5, pp. 63–68.

Villagra, J., B. d'Andrea Novel, M. Fliess, and H. Mounier (2008). "Estimation of longitudinal and lateral vehicle velocities: an algebraic approach". In: *IEEE American Control Conference*, pp. 3941–3946.

Volvo (2009). *3-point safety belt from Volvo - The most effective lifesaver in traffic for fifty years.* URL: `https://www.media.volvocars.com/global/en-gb/media/pressreleases/18405` (visited on 04/29/2018).

Wan, E. A. and R. van der Merwe (2000). "The unscented Kalman filter for nonlinear estimation". In: *IEEE Symposium on Adaptive Systems for Signal Processing, Communications, and Control.*

Weiss, A., C. Petersen, M. Baldwin, R. S. Erwin, and I. Kolmanovsky (2014). "Safe positively invariant sets for spacecraft obstacle avoidance". *Journal of Guidance, Control, and Dynamics* **38**:4, pp. 720–732.

Wired (2018). *The wired guide to self-driving cars.* URL: `https://www.wired.com/story/guide-self-driving-cars/` (visited on 04/29/2018).

Yang, T., P. G. Mehta, and S. P. Meyn (2013). "Feedback particle filter". *IEEE Transactions on Automatic Control* **58**:10, pp. 2465–2480.

Young, P. C. and J. Willems (1972). "An approach to the linear multivariable servomechanism problem". *International Journal of Control* **15**:5, pp. 961–979.

Zanelli, A., R. Quirynen, G. Frison, and M. Diehl (2017). "A partially tightened real-time iteration scheme for nonlinear model predictive control". In: *IEEE Conference on Decision and Control*, pp. 4388–4393. DOI: 10.1109/CDC.2017.8264306.

Zenklusen, R. (2015). *Lecture Notes in Introduction to Mathematical Optimization*. 1st ed. Department of Mathematics, ETH Zürich.

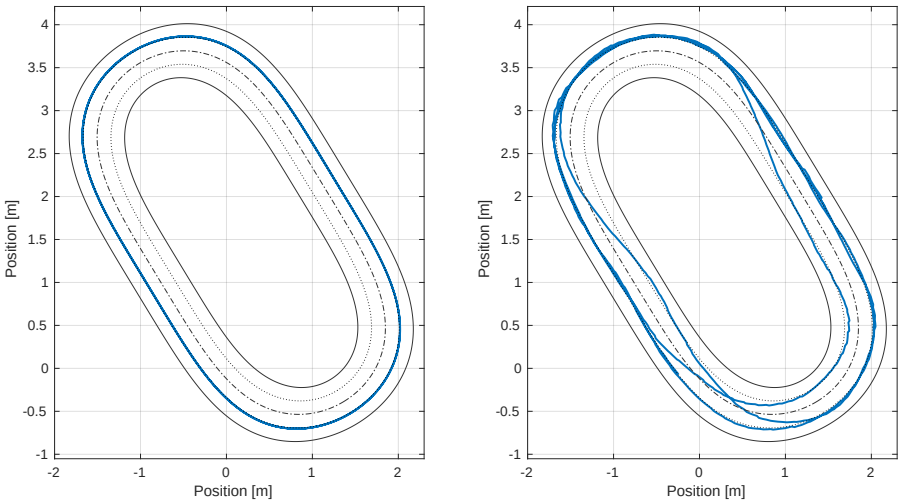
10

Appendix A

In this appendix, the motion-planning results from the simulation study are presented. The plots should be compared with the their counterparts from the real-life driving experiments, which illustrate the impact of communication delays, model mismatches, etc. Since the simulated data are used to compute the motion plans in the global coordinate frame, the tracking error plots are redundant and thus not presented. Additionally, since obstacle detection using LIDAR is solely for practical purposes, no simulation experiments were conducted for this particular case.

Tracking Performance

In Figure 10.1, traces of the EV's motion in global fixed-frame coordinates are shown from the simulation study. In subplot (a), there are no OVs present and in (b) there are two OVs driving with speeds of 0.15 m/s and 0.2 m/s, respectively.



(a) (b)

Figure 10.1 Traces of the EV's motion in the global coordinate system from the simulation study. In (a), there is no OVs present and in (b) there are two OVs present.

Overtake with one OV

Figures 10.2 and 10.3 depict an overtaking scenario with one OV in global and road-aligned coordinates, respectively. The OV is driving with a speed of 0.15 m/s and the constant velocity profile during the overtake can be seen in the upper plot of Figure 10.3.

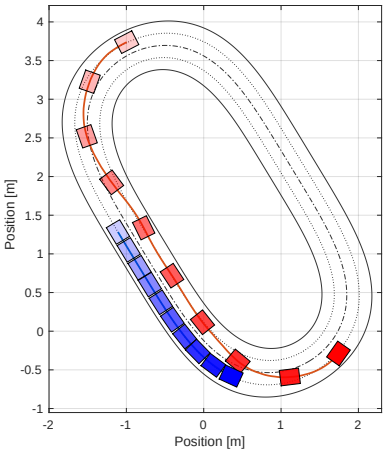


Figure 10.2 Overtaking scenario from the simulation study with one OV shown in the global fixed-frame coordinate system.

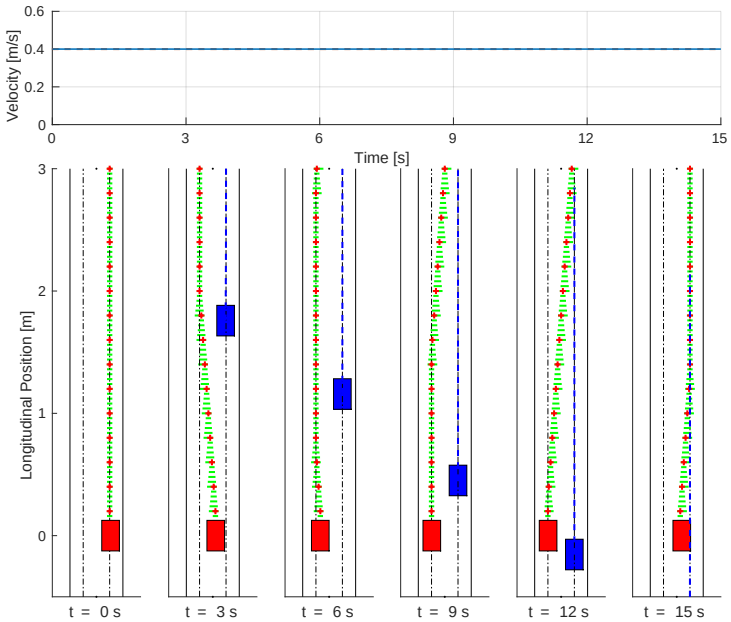


Figure 10.3 Overtaking scenario from the simulation study with one OV from a ROI perspective.

Overtake with two OVs

Figure 10.4 depicts an overtaking scenario with two OVs driving with speeds of 0.15 m/s and 0.2 m/s, respectively.

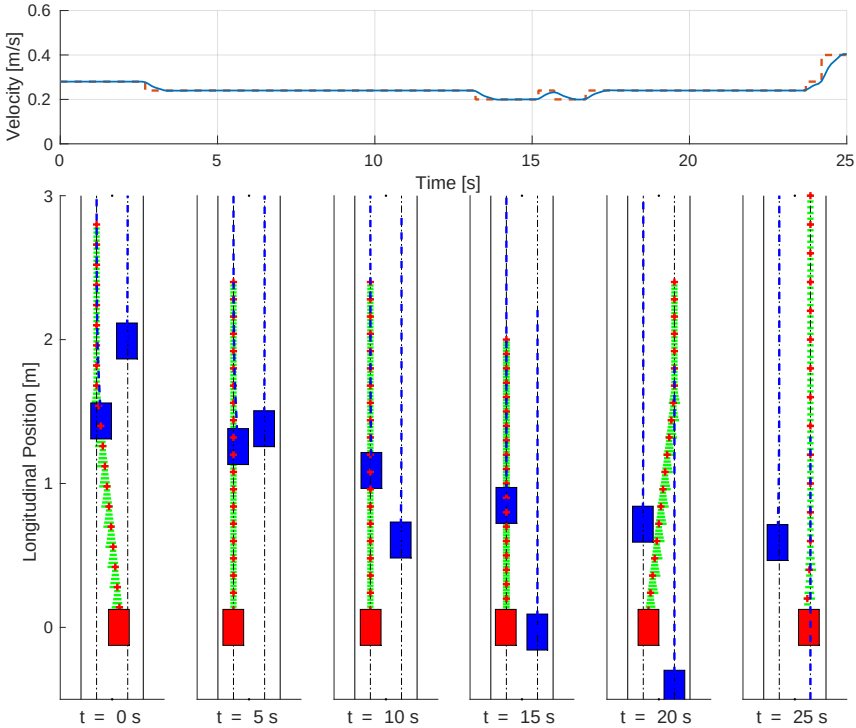


Figure 10.4 Overtaking scenario from the simulation study with two OVs from a ROI perspective.

Intersection with crossing OVs

Figures 10.5 and 10.6 show intersection scenarios with a stop sign and crossing vehicles. In the latter figure, there is one OV standing still on the main road and hence the EV has to switch lane as it approaches the intersection.

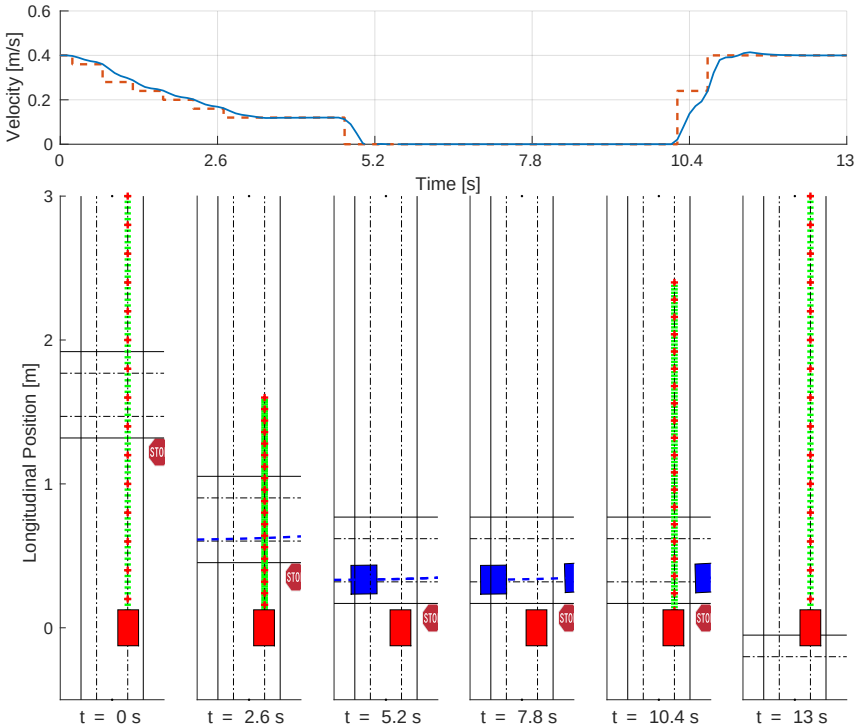


Figure 10.5 Intersection scenario from the simulation study with a stop sign.

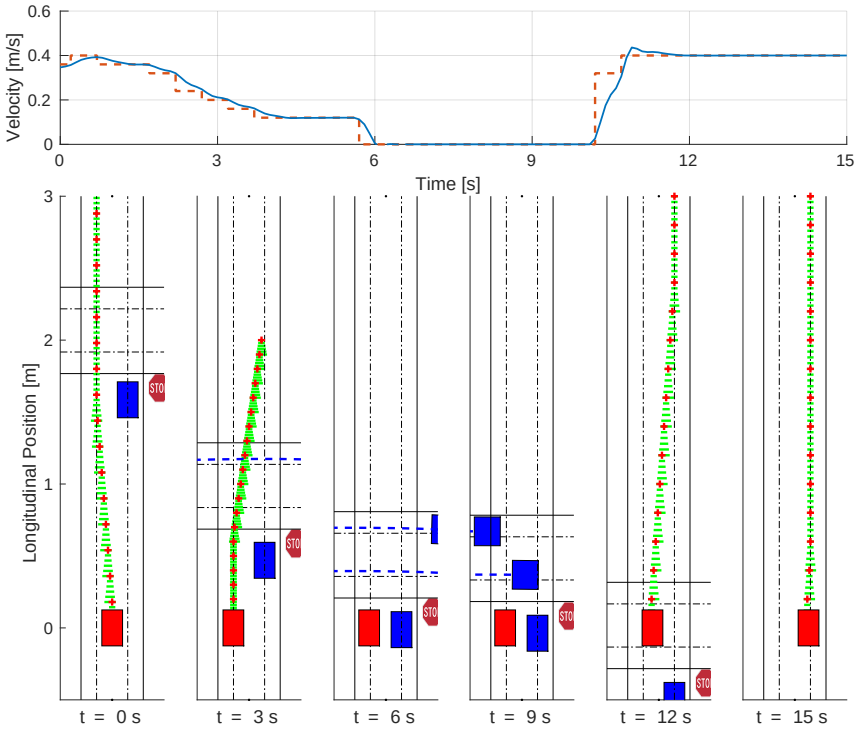


Figure 10.6 Intersection scenario from the simulation study with a stop sign and overtaking of one OV.

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER'S THESIS

Date of issue
August 2018

Document Number
TFRT-6053

Author(s)
Richard Bai
Karl Fredrik Erliksson

Supervisor
Karl Berntorp, AB Berntec
Björn Olofsson, Dept. of Automatic Control, Lund University, Sweden
Anders Robertsson, Dept. of Automatic Control, Lund University, Sweden (examiner)

Title and subtitle
Motion Planning using Positively Invariant Sets on a Small-Scale Autonomous Vehicle

Abstract

Self-driving technology has the opportunity to increase safety in automotive transportation by reducing the impact of human error. Motion planning is a key component of an autonomous system, responsible for providing reference trajectories and paths that the vehicle should follow. This thesis studies a motion-planning algorithm based on positively invariant sets. The focus is on design, implementation, and evaluation of the algorithm on a small-scale ground-vehicle robot platform. By incorporating a gain-scheduling approach into the motion planner, guaranteed safe reference trajectories, capable of navigating the vehicle in a dynamic environment of static and moving obstacles, can be computed for time-varying velocities.

This thesis also deals with sensor-fusion aspects for autonomous vehicles. Through a localization system based on an Extended Kalman Filter (EKF), reliable and robust state estimates can be obtained from inertial sensor data, without the use of an external positioning system. It is shown that the motion-planning algorithm together with the localization system is capable of performing safe overtaking maneuvers for time-varying velocities. Simpler urban driving scenarios involving traffic signs and intersections are used to illustrate the ability of the proposed motion-planning algorithm to also handle more complex driving scenarios. By using laser scans from Light Detection and Ranging (LIDAR) equipment, it is shown that obstacles can be detected and avoided in real-life driving experiments.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-103

Recipient's notes

Security classification

http://www.control.lth.se/publications/