AMZ Driverless: The Full Autonomous Racing System

Juraj Kabzan[†1], **Miguel I. Valls**[1], **Victor J.F. Reijgwart**[1], **Hubertus F.C. Hendrikx**[1], **Claas Ehmke**[1], **Manish Prajapat**[1], **Andreas Bühler**[1], **Nikhil Gosala**[1], **Mehak Gupta**[1], **Ramya Sivanesan**[1], **Ankit Dhall**[1], **Eugenio Chisari**[1], **Napat Karnchanachari**[1], **Sonja Brits**[1], **Manuel Dangel**[1], **Inkyu Sa**[2], **Renaud Dubé**[2], **Abel Gawel**[2], **Mark Pfeiffer**[2], **Alexander Liniger**[3], **John Lygeros**[3] **and Roland Siegwart**[2]
ETH Zurich
Zurich, Switzerland

Abstract

This paper presents the algorithms and system architecture of an autonomous racecar. The introduced vehicle is powered by a software stack designed for robustness, reliability, and extensibility. In order to autonomously race around a previously unknown track, the proposed solution combines state of the art techniques from different fields of robotics. Specifically, perception, estimation, and control are incorporated into one high-performance autonomous racecar. This complex robotic system, developed by AMZ Driverless and ETH Zurich, finished 1st overall at each competition we attended: Formula Student Germany 2017, Formula Student Italy 2018 and Formula Student Germany 2018. We discuss the findings and learnings from these competitions and present an experimental evaluation of each module of our solution.

Supplementary Material

For a supplementary video visit: `https://www.youtube.com/watch?v=hYpToBIMpVQ`
For open-source software and datasets visit: `https://github.com/AMZ-Driverless`

[†] Correspondence: `kabzanj@gmail.com`
[1] Authors are with AMZ Driverless, ETH Zürich.
[2] Authors are with the Autonomous Systems Lab (ASL), ETH Zürich.
[3] Authors are with the Automatic Control Laboratory (IfA), ETH Zürich.

1 Introduction

Self-driving vehicles promise significantly improved safety, universal access, convenience, efficiency, and reduced costs compared to conventional vehicles. In order to fulfill the Society of Automation Engineers (SAE) level 4 autonomy[1], no driver attention must be required, even in emergency situations and under challenging weather conditions. Although a major part of autonomous driving on the public roads will happen in standard situations, a crucial aspect to reach full autonomy is the ability to operate a vehicle close to its limits of handling, i.e. in avoidance maneuvers or in case of slippery surfaces.

Figure 1: *gotthard* driverless at the Formula Student Driverless competition which took place in August 2018 at the famous Hockenheimring, Germany.

In autonomous car racing, vehicles race without a human in the loop as shown in Figure 1. Much alike traditional motorsports, autonomous racing provides a platform to develop and validate new technologies under challenging conditions. Self-driving racecars provide a unique opportunity to test software required in autonomous transport, such as redundant perception, failure detection, and control in challenging conditions. Testing such systems on closed tracks mitigates the risks of human injury.

This paper describes an entire autonomous racing platform, covering all required software modules reaching from environment perception to vehicle dynamics control. Starting with the perception pipeline, the developed system works using either LiDAR, vision or both. Next, the motion estimation subsystem fuses measurements from five different sensors, while rejecting outliers in the sensor readings. For localization and mapping, we utilize a particle filter-based SLAM system, facilitating the detection and association of landmarks. Subsequently, we propose a custom approach to plan paths which takes the most likely track boundaries into account, given a map and/or on-board perception. Lastly, we present a control framework that directly minimizes lap time while obeying the vehicle's traction limits and track boundary constraints.

The presented system has been thoroughly tested in context of the world's largest autonomous racing competition, Formula Student Driverless (described in Section 2). Our team, AMZ Driverless, secured the overall first place at each competition we participated in (see Table 1). This demonstrates that the system can reliably race and finish on previously unknown tracks.

This paper presents a revision of our previous works (de la Iglesia Valls et al., 2018; Gosala et al., 2018; Dhall et al., 2019) and extends these with a description of the full autonomous system, including path planning

[1]Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles (J3016_201609), `https://www.sae.org/standards/content/j3016_201609/`

FS Germany 2017			FS Italy 2018			FS Germany 2018		
Team	Fastest	Total	Team	Fastest	Total	Team	Fastest	Total
Zurich ETH	55.53	557.89	**Zurich ETH**	-	776.28	**Zurich ETH**	28.48	285.93
Hamburg TU	105.44	DNF	Munchen UAS	-	874.51	Hamburg TU	48.93	581.24
			Karlsruhe KIT	-	DNF	Darmstadt TU	233.31	2336.60
						Karlsruhe KIT	124.79	DNF
						Augsburg UAS	146.64	DNF
						Munchen UAS	81.14	DNF
						Trondheim NTNU	138.46	DNF

Table 1: Overview of the fastest lap and total time in seconds of each team that finished at least one lap, at each competition attended by AMZ Driverless. DNF is used when a team did not finish all ten laps. No official fastest lap times for FS Italy 2018 are available. The aim of every team is to achieve the lowest total time.

(Section 4.3), control (Section 5) and the testing framework (Section 6). Furthermore, this paper provides a holistic evaluation of the system (Section 7).

In summary, the contributions of this paper are;

- A complete autonomous racing system, covering all aspects of environment perception, state estimation, SLAM, planning, and vehicle dynamics control[2]
- A demonstration of the system's capabilities on a full scale autonomous racecar, by reaching lateral accelerations of up to 1.5g ($14.7\,\mathrm{m\,s^{-2}}$)
- A performance evaluation of each subsystem, after thorough testing in context of real-world autonomous racing competitions
- An open-source implementation of our automated simulation framework as well as an extensive data set for development and testing of autonomous racing applications [3]

The paper is organized as follows; Section 1 presents related studies for autonomous racing. The problem statement and overall concept are addressed in Section 2. Perception, motion estimation and mapping are described in Section 3 and 4 respectively and are followed by the vehicle control strategy in Section 5. Each of these 3 sections directly presents its associated real-world experimental results. Our simulation testing framework is presented in Section 6. In Section 7, we provide an overview of the performance of the system as a whole and share our insights and suggestions for designing an autonomous racecar. Finally, this paper is concluded in Section 8.

Related Work

In 2004, 2005 and 2007, DARPA organized a series of Grand Challenges designed to radically speed up autonomous driving development. The first two challenges consisted of autonomous races and were extensively covered in the literature (Iagnemma and Buehler, 2006a; Iagnemma and Buehler, 2006b; Buehler et al., 2008). The system presented in this paper differs in that the car is designed to travel over a narrow unknown race track, whose boundaries are indicated only by cones.

After the DARPA challenges ended, the bulk of autonomous driving research shifted toward the industry. Yet several systems papers on autonomous racing are still being published, such as in (Goldfain et al., 2018)

[2]For a video showing the entire pipeline in action visit `https://www.youtube.com/watch?v=hYpToBIMpVQ`

[3]`https://github.com/AMZ-Driverless`

describing the AutoRally platform. The autonomous system presented in this paper differs in that it runs on a full size, road racecar and has been tested at speeds exceeding $15\,\mathrm{m\,s^{-1}}$.

In the remaining paragraphs, a concise overview is provided of the state of the art for each sub-system presented in this paper.

Starting with perception, a system has been developed that can detect cones, estimate their 3D poses and colour using either LiDAR, vision or both. This follows from the fact that the tracks used to validate this paper's autonomous racecar are demarcated solely by colored cones and redundancy against single sensor failure was identified as a key requirement for robustness. This is explained in more detail in Section 2.

The LiDAR based perception system breaks the problem into a ground removal step similar to (Himmelsbach et al., 2010). It then uses Euclidean clustering and classification based on geometric priors to detect the cones and their poses, as presented in our first paper (de la Iglesia Valls et al., 2018). As a last step, the color patterns on sufficiently nearby cones can be identified using LiDAR intensity returns processed within a convolutional neural network (CNN). We presented this full pipeline in our previous paper (Gosala et al., 2018). This paper therefore only includes a short summary for completeness and to provide context for further experimental results.

To be able to drive through a track of cones, first they need to be detected on the image. Before the advent of large datasets and general purpose GPUs, classical computer vision techniques such as (Viola and Jones, 2001) for human face detection using smart, hand-crafted features and pedestrian detector (Dalal and Triggs, 2005) using histogram of oriented gradients (HoG) and support vector machines (SVM) were commonly used. Recently, CNNs such as the Single-shot detector (Liu et al., 2016) and YOLO (Redmon et al., 2016) have shown great promise in detecting objects more efficiently in a single pass of the image, unlike previous methods. In our work, we use a modified version of YOLOv2 (Redmon and Farhadi, 2016) for the purpose of cone detection in our pipeline. Further, to apply these detections for robotics based applications, it is essential to know the 3D position of these objects. CNNs have been proposed to predict view-points (Tulsiani and Malik, 2015) for different objects; their work aims to capture the relation between keypoints on an object and their respective view-points. More recently, end-to-end pose estimation has been proposed where a 6 DoF pose is the output. In our work, instead of directly producing the pose of an object from a CNN model, we use a CNN for locating the keypoints in the image (Dhall et al., 2019) and subsequently use information about the cone's appearance and geometry to estimate it's pose via 2D-3D correspondences.

The Motion Estimation subsystem estimates the linear and angular velocities by fusing measurements from six different sensors. Several filter families exist, including the Extended Kalman Filter (EKF), Unscented Kalman Filter and Particle Filter (Thrun et al., 2005). An EKF was chosen given it's computational efficiency and accuracy when dealing with mildly non-linear systems perturbed by Gaussian noise (Xue and Schwartz, 2013). Outliers are rejected using the chi-squared test and a variance based drift detector. We previously discussed these outlier rejection approaches in (de la Iglesia Valls et al., 2018) and (Gosala et al., 2018). Consequently, this paper focuses on a more detailed description of the process and measurements models that we used, followed by more detailed experimental results.

A broad range of techniques exist to concurrently map an environment and localize a robot within it (Cadena et al., 2016). Since the race tracks in our experiments are initially unknown, the use of an online SLAM method is required. Many promising online graph-based SLAM methods have been proposed in the literature, such as (Engel et al., 2014; Hess et al., 2016; Mur-Artal and Tardos, 2017; Labb and Michaud, 2019). However, the tracks we race on consist almost solely of colored cones placed atop wide open surfaces. The environment is thus sparse and comprised of landmarks that are distinguishable but that cannot be identified uniquely. This problem lends itself well to Particle Filter based SLAM systems such as FastSLAM (Montemerlo et al., 2002). This was illustrated in (de la Iglesia Valls et al., 2018), where we applied FastSLAM to autonomous racing and showed that it efficiently and reliably provides sufficiently accurate maps. We extended this work by switching to FastSLAM 2.0 (Montemerlo et al., 2003) and by estimating

cone colors in addition to their poses in (Gosala et al., 2018). In this paper we therefore focus on providing more detailed experimental results and only briefly discusses the SLAM pipeline itself.

The planning problem statement for our autonomous racing use case differs from the general planning problem presented in the literature. Because the track is unknown when driving the first lap and the perception range is limited, there is no known goal position. Many classical search techniques such as those presented in (Paden et al., 2016) can therefore not be directly used. In (Tanzmeister et al., 2013) a search technique is presented that does not require knowledge of a goal position and instead picks the best trajectory out of a set of feasible candidates. Since the environment in our problem is entirely built up of cones, we propose to discretize the space into a Delauney graph (Delaunay, 1934) by connecting only the middle points of each cone pair. Paths are sampled from this graph and the most likely track middle line is chosen based on a quadratic cost function that takes the path's geometry, the rules of the competition, and the uncertainty of cone colors and positions into account. This middle line is then tracked with a pure pursuit controller (Coulter, 1992) to complete the first lap.

Once the first lap is completed, the SLAM subsystem provides a map including the boundaries of the entire track. Therefore, our focus lies in control techniques that are able to use this knowledge to push a car to its performance limits. In the literature, several control techniques for autonomous racing are known. They can be roughly categorized into two categories, predictive (Liniger et al., 2015; Verschueren et al., 2014; Funke et al., 2017; Rosolia et al., 2017; Liniger and Lygeros, 2019) and non-predictive controllers (Zhang et al., 2001; Kritayakirana and Gerdes, 2012; Klomp et al., 2014). Since non-predictive racing controllers are not able to plan the motion of the car, it is necessary to compute the racing line beforehand. An approach which was shown to work well is (Funke et al., 2012), where a set of clothoid curves was used to build a path. Alternatively, also methods from lap time optimization could be used (Casanova et al., 2000; Kelly and Sharp, 2010). On the other hand, predictive racing controllers, are able to plan the optimal motion of the car by re-planning in a receding horizon fashion. Due to not requiring any pre-optimization and thereby being flexible with respect to the track we decided to use a predictive motion planning technique. More precisely we extended the real-time Model Predictive Control (MPC) method proposed in (Liniger et al., 2015), which is able to directly find an optimal trajectory based on the performance envelope of the car. In this paper, we extend the method extensively to work with full-sized cars. The main difference is a novel vehicle model which is able to deal with zero velocity and includes the behavior of low-level controllers.

2 Main Concept

The presented work was developed for the Formula Student Driverless competition. Therefore, the competition's rules, goals, and constraints lay the basis for the problem formulation and thus are detailed in the present section.

2.1 Problem Statement – The Race

Recognizing the interest in autonomous driving and the associated technological challenges, Formula Student Germany organized the first driverless competition in 2017 followed by other countries in 2018. Formula Student (FS) is an international engineering competition, in which multidisciplinary student teams compete with a self-developed racecar every year.

The main race, *trackdrive*, consists of completing ten laps, as fast as possible, around an unknown track defined by small $228 \times 335\,\text{mm}$ cones. Blue and yellow cones are used to distinguish the left and the right boundary respectively. The track is an up to 500m long closed circuit, with a minimum track width of 3m and the cones in the same boundary can be up to 5m apart. By the rule book, this track contains straights, hairpins, chicanes, multiple turns, and decreasing radius turns among others. An exemplary track layout schematic is provided in Figure 2.

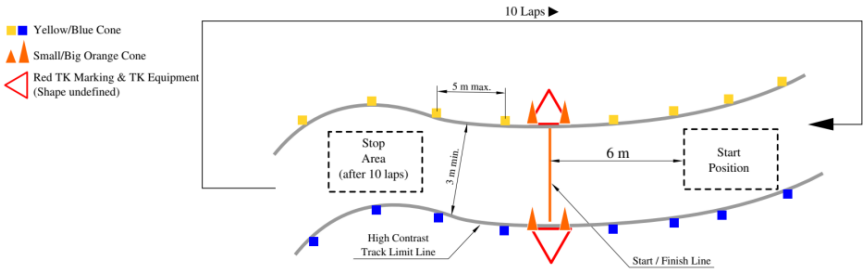


Figure 2: The track layout of a trackdrive discipline (FSG, 2018). Blue and yellow cones mark the track boundaries.

It is important to note that the track is completely unknown before starting the race which increases the challenge considerably. In addition, all computations and sensing are required to happen on-board.

2.2 Hardware Concept – The Racecar

The AMZ driverless racecar is an electric 4WD racecar with a full aerodynamic package, lightweight design and high drivetrain efficiency designed by AMZ[4] in 2016, see Figure 1. The car weighs only 190 kg and is able to accelerate from $0 - 100\text{km}\,\text{h}^{-1}$ in 1.9s, faster than any road car or Formula1 car.

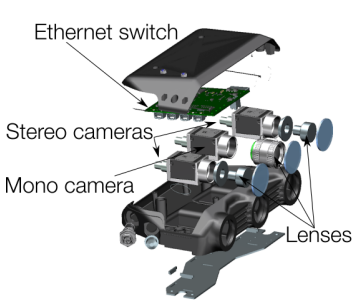
In order to make the vehicle race fully autonomously, sensors allowing for environment perception need to be added. As all the perception algorithms, decision making and control will have to be executed on board, also additional computational units have to be added. The main system is required to be robust and reliable, which calls for redundancy in the pipeline. Therefore, two independent perception pipelines working with multiple sensor modalities were developed, allowing a robust operation of object detection (namely cones) and state estimation.

The first sensor modality is a 3D LiDAR sensor which is placed on the front wing. This maximizes the number of point returns per cone, and thus allows to perceive cones which are further away. A LiDAR was chosen over radars because of its ability to detect small non-metallic objects (such as plastic cones) and its high accuracy in depth. The choice of the LiDAR sensor is based on its physical parameters like the horizontal and vertical resolution and fields-of-view (FoV). In our application, the vertical resolution turns out to be the most important parameter since it limits the number of returns per cone which directly relates to the distance at which cones can be perceived.

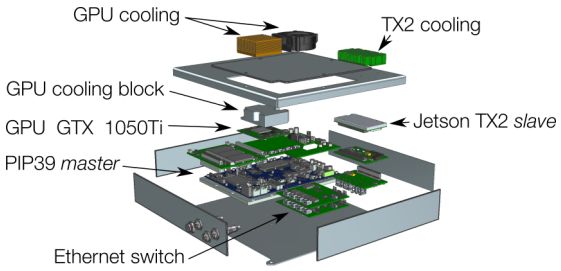
The second sensor modality is image-based. Three CMOS (Allen and Holberg, 1987) cameras with global shutter are used in a stereo and mono setup, see Figure 3a. The stereo setup is intended to deal with cones in close proximity whereas the mono camera is used for detecting cones further away. To ensure that the position estimates of cones close to the car is highly accurate, the stereo setup uses lenses with short focal lengths (focal length of 5.5mm and horizontal FoV of 64.5° each). For accurately estimating the cone positions in the range of $7 - 15\text{m}$, a lens with focal length of 12mm and horizontal FoV of 54.5° was chosen for the monocular camera. Considering the vantage point, the cameras were placed on the main roll hoop, above the driver's seat in the car, see Figure 4. This offers the advantage that the occlusion among cones is reduced to a minimum and even the cones placed one behind the other (in line of sight) can be perceived sufficiently well.

Besides cameras and a LiDAR, which are used to determine the position of cones, the AMZ driverless racecar is equipped with additional sensors to estimate the car's velocity. The sensor setup is shown in Figure 4. Each of the four hub mounted motors incorporates a resolver, which provides smooth angular speed measurements. The motor controllers provide motor torques and the car is equipped with an industrial grade, laser based

[4] www.amzracing.ch



(a)



(b) Our computing system consists of a *master* computer, a *slave* computer and an industrial ethernet switch.

Figure 3: Self-developed camera and computing system housing.

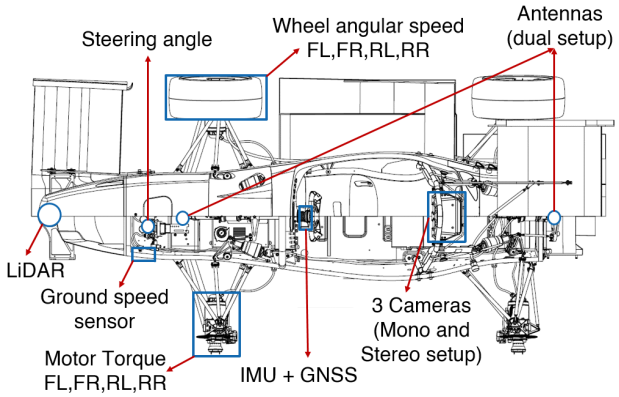


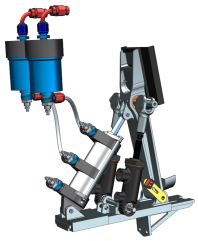
Figure 4: Location of all sensors providing data for motion estimation and perception.

non-contact optical ground speed sensor (GSS). The steering angle is measured using an encoder placed in the steering rack. An accurate heading is crucial for localization and mapping of an autonomous vehicle, therefore, a dual antenna Inertial Navigation System (INS) is used which is capable of providing heading measurements with an accuracy of 0.08°. The INS also includes an Inertial Measurement Unit (IMU).

Similar to the LiDAR-camera setup, this redundant sensor setup can cope with single sensor failures. The need for such redundancy was confirmed throughout the testing seasons where many failures where observed.

As a steering actuator, a motor was added under the front axle. This motor is able to steer from full left to full right in less than 0.9s. The used autonomous racecar is capable of braking by recuperation through the motors. This generates a sufficient deceleration in order to race without a driver while not using the mechanical brakes actively. An Emergency Braking System (EBS) was installed behind the pedals as shown in Figure 5a. This system is only able to either fully brake or release the mechanical brakes and is used only in case of emergency. It can be triggered from either a Remote Emergency System (RES) or from an on-board computer.

To execute the required computations, two computing units were installed (see Figure 3b). Both units have a CPU and a GPU. One is a rugged industrial computer — from here on called *master* computer — and the other is an efficient embedded unit called *slave*. The hardware and software were designed such that in case of *slave* failure (running only non-critical parts of the software pipeline), the system can still race. An Electronic Control Unit (ECU) is the only system that interacts with the actuators, also handling the low level controls and safety checks. In Figure 6 one can see which parts of the pipeline run in which unit.



(a) An Emergency Braking System (EBS) was added behind the braking pedal.

(b) A steering motor was added below the axle.

Figure 5: Actuators were installed to reproduce a human driver capabilities.

2.3 Software Concept

To reach the full potential of the car, it is estimated that the track must be known for at least 2s ahead of the vehicle's current postion. This horizon, which at $80\text{km}\,\text{h}^{-1}$ is above 40m is not achievable with the on-board sensor setup. Thus, a map of the track is built, such that once the track is known, its shape ahead can also be anticipated by estimating the car's pose in the map. This defines our two modes of operation, *SLAM* mode and *Localization* mode.

In *SLAM* mode, the vehicle path has to be computed from the local perception and a map of the track has to be generated. The main challenges are the limited perception horizon as the track has to be inferred from the perceived cones only. In *Localization* mode, the goal is to race as fast as possible in the previously mapped track. Given the above requirements, due to the need to detect cones, map the track, and control the car to its full potential, the software stack is divided in three main modules, *Perception*, *Motion Estimation and Mapping* and *Control*.

Perception: Its goal is to perceive cones and estimate their color and position as well as the associated uncertainties. This will be the input to the *Motion Estimation and Mapping* module. To increase robustness, the perception system is designed to withstand any single failure and thus two fully independent pipelines are implemented at a hardware and software level, namely the LiDAR and camera pipelines. More details are provided in Section 3.

Motion Estimation and Mapping: Its goal is to map the track, detect the boundaries and estimate the physical state of the car. The inputs are the observed cone positions from *Perception* (camera and LiDAR pipeline) and the remaining sensors' measurements to estimate the velocity, see Section 4. For this purpose, three elements are needed. First, a velocity estimation that produces a fault tolerant estimate. Second, a SLAM algorithm, that uses cone position estimates from *Perception* as landmarks, and the integrated velocity estimate as a motion model. Finally, and only for the *SLAM* mode, a boundary estimation algorithm is used to classify cones as right or left boundary despite colour misclassification. Further insights are provided in Section 4.

Control: Its goal is to safely operate the car within the limits of handling while minimizing lap time. Taking the track boundaries and the state estimates as inputs, the control actions (steering, throttle and brake) are computed. Stability and robustness to external disturbances are important factors for safe operation. The low-level control distributes the torque among the four wheels, the safety system checks for consistency and unsafe states and is able to trigger the EBS. See Section 5 for further details.

The software-hardware architecture of the system can be seen in Figure 6. Going from sensors to actuators passing through all software modules and the computing system they run on.

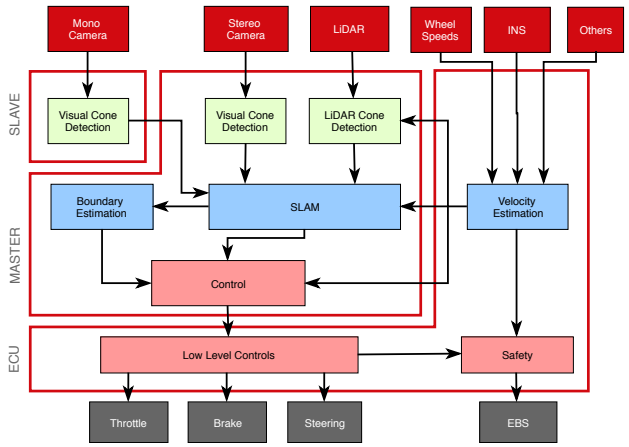


Figure 6: Software-hardware architecture of the autonomous system. From top to bottom the sensors are depicted in red, perception submodules in green, motion estimation and mapping submodules in blue, control submodules in light red and actuators in grey. The three red bounding boxes show the computing platform where the submodule is executed.

3 Perception

The goal of the perception pipeline is to efficiently provide accurate cone position and color estimates as well as their uncertainties in real-time. Using these cone positions as landmarks, the *SLAM* module builds a map aiding the car to navigate autonomously. In order to perceive the environment reliably, a combination of sensors is used for redundancy and robustness. A LiDAR based pipeline is presented to detect cones based on its geometric information and a pattern recognition method based on LiDAR intensity returns to estimate color of the cones with a high confidence. In parallel, a camera-based multi-object detection algorithm is implemented alongside algorithms to accurately estimate the 3D positions of the cones, both from stereo and monocular images.

3.1 LiDAR-based Sensing & Estimation

We capitalize on the LiDAR information in two ways. First, we use the 3D information in the point cloud to detect cones on the track and find their position with respect to the car. Furthermore, we use the intensity data to differentiate between the various colored cones.

3.1.1 The Cone Detection Algorithm

The cones demarcating the race track are detected and classified using the pipeline shown in Figure 7. The three major phases, pre-processing, cone detection, and color estimation make up the cone detection algorithm, which are explained next.

Pre-processing

Due to the placement of the LiDAR sensor on the car, only cones in front of the car can be perceived, while the rear-view is occluded by the racecar. Thus the points behind the sensor are filtered out. The LiDAR sensor cannot inherently estimate motion which can lead to large distortions in the point cloud of a single scan and the appearance of *ghost cones* if not accounted for. The scanned fractions are thus undistorted by using the velocity estimates of the car.

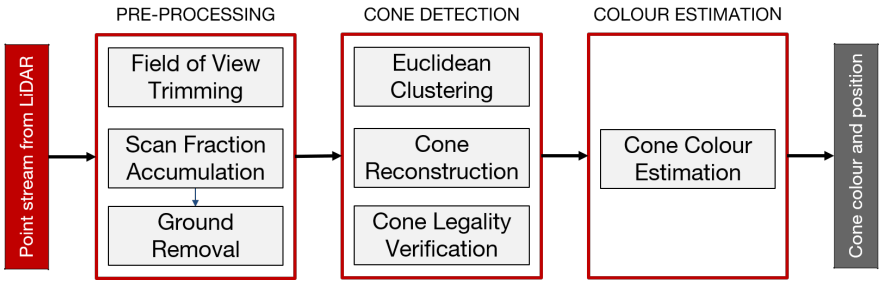
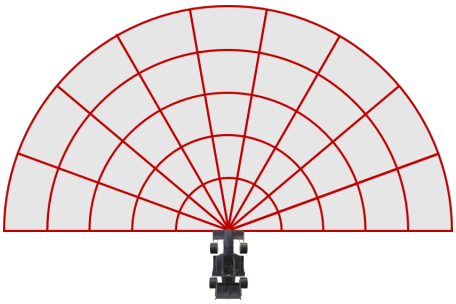
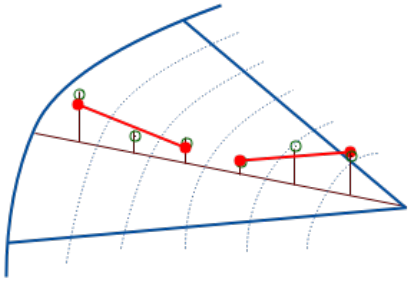


Figure 7: The LiDAR pipeline used to detect cones and estimate their color. The pipeline is composed of three major phases, namely *Pre-processing*, *Cone Detection*, and *Color Estimation*. The pipeline accepts raw point clouds from the LiDAR sensor and outputs the location and color of the cones with respect to the reference frame of the car.

An adaptive ground removal algorithm (Himmelsbach et al., 2010) that adapts to changes in the inclination of the ground using a regression based approach is used to estimate the ground plane and distinguish between the ground and non-ground points, after which the ground points are discarded (Gosala et al., 2018). The ground removal algorithm works by dividing the FoV of the LiDAR into angular segments and splitting each segment into radial bins (see Figure 8a). A line is then regressed through the lowermost points of all bins in a segment. All the points that are within a threshold distance to this line are classified as ground points and are removed (see Figure 8b).



(a) The FoV of the LiDAR is divided into multiple segments and bins.



(b) Isometric view of the adaptive ground removal algorithm where the green circles represent the lowest LiDAR point in each bin and the red lines represent the estimated ground plane. The ground plane is regressed through the lowest points of all the bins in each segment and all points in the vicinity of these regressed ground lines are classified as ground and are subsequently removed.

Figure 8: View of LiDAR which is placed on top of the front wing.

Cone Detection

The aforementioned ground removal algorithm removes a substantial amount of cone points in addition to those of the ground. This significantly reduces the already small number of return points that can be used to detect and identify cones. This is addressed by first clustering the point cloud after ground removal using Euclidean-distance based clustering algorithm and then reconstructing a cylindrical area around each cluster center using points from the point cloud before ground removal. This recovers most of the falsely removed points improving cone detection and color estimation (see Figure 9). The reconstructed clusters are then passed through a rule-based filter that checks whether the number of points in that cluster is in accordance

with the expected number of points in a cone at that distance computed with

$$E(d) = \left(\frac{1}{2} \frac{h_c}{2d \tan(\frac{r_v}{2})} \frac{w_c}{2d \tan(\frac{r_h}{2})} \right), \tag{1}$$

where d represent distance, h_c and w_c are the height and width of the cone respectively, and r_v and r_h are the vertical and horizontal angular resolutions of the LiDAR respectively. The clusters that successfully pass through this filter are considered to be cones and are forwarded to the color estimation module.

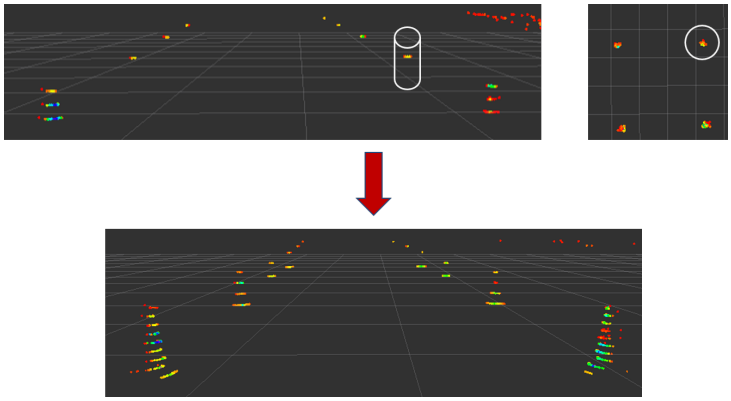


Figure 9: An illustration of the cone reconstruction process. The top-left image shows the LiDAR point cloud after executing the adaptive ground removal algorithm. One can note the sparsity of the point cloud and the dearth of points in a cone. The cones are reconstructed by retrieving a cylindrical area around each cluster center (top-right) resulting in the bottom image wherein the presence of cones is more evident.

Cone Pattern Estimation

According to the rule book (FSG, 2018), a yellow cone has a *yellow-black-yellow* pattern whereas a blue cone has a *blue-white-blue* pattern which results in differing LiDAR intensity pattern as one moves along the vertical axis of the cone as shown in Figure 10. This intensity gradient pattern is leveraged in the cone color estimation using a CNN architecture. The CNN consists of four convolutional and five fully connected layers as shown in Figure 11. The network accepts a 32×32 grayscale image of the cone as input and outputs the probability of the cone being *blue*, *yellow*, and *unknown*. The input image is created by mapping the 3D bounding box of the cone cluster to a 32×32 image where the cluster center is mapped to the center of the image and all the other points are appropriately scaled to fit in it. The pixel values store the intensities of points in the point cloud which are then scaled by a constant factor to increase the disparity between the intensities of various layers and make the difference more apparent. The CNN uses an asymmetric cross-entropy loss function that penalizes misclassifications (eg. blue cone classified as yellow cone) much more severely than incorrect classifications (eg. blue cone classified as unknown) which results in the network being more cautious towards ambiguous inputs and reduces the overall probability of misclassifying the color of cones. Furthermore, dropout and batch-normalization are used to prevent complex co-adaptations between neurons, control over-fitting, and improve the generalization of the network. However, the sparsity of the point cloud combined with the small size of the cones limits the color estimation range to 4.5 m, as cone returns beyond this distance do not allow for a distinction between the differing intensity patterns.

3.2 Camera-based Sensing & Estimation

To make the perception pipeline robust to sensor failure, a camera based sensing and estimation pipeline is deployed in parallel and independent of the LiDAR one. Considering that the vision pipeline is developed for a real time system, it becomes crucial to detect and estimate color and positions of multiple cones with as little latency as possible and with minimum utilization of computational resources. Hence, this section proposes

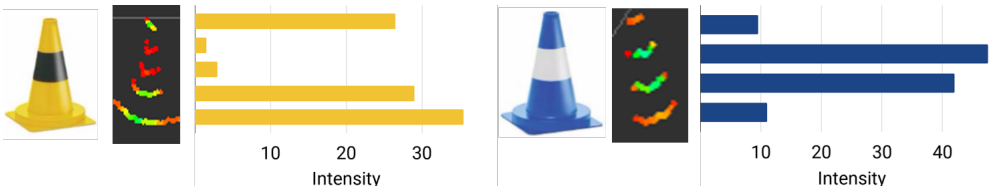


Figure 10: The intensity gradients for the pre-defined yellow and blue cones along with their point cloud returns. When reading the intensity values along the vertical axis, one denotes *high-low-high* and *low-high-low* patterns for the yellow and blue cones respectively. These differing intensity patterns are used to differentiate between yellow and blue cones.

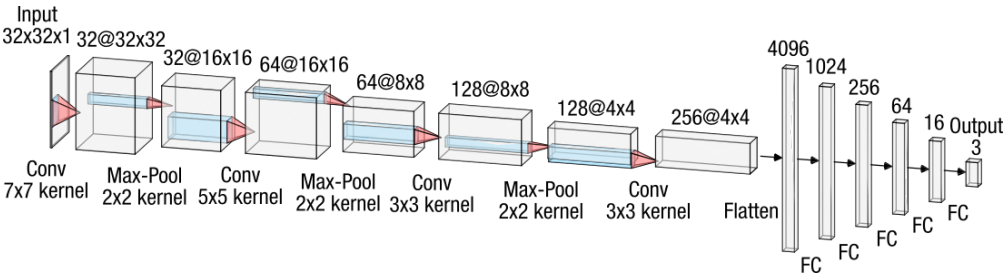


Figure 11: An illustration of our CNN architecture for estimating the color of the cones which is composed of four convolutional and five fully-connected layers. The networks accepts a 32x32 grayscale image and outputs the probability of the cone belonging to each of the three classes, namely *blue*, *yellow*, and *unknown*.

a real-time computer vision pipeline to detect and estimate 3D position using a single image (monocular pipeline) and a pair of images (stereo pipeline).

3.2.1 The Cone Detection Algorithm

Cones on the track are detected and their positions are estimated using the pipeline shown in Figure 12. The two pipelines have three major phases, multiple object detection Neural Network (NN), key-point regression and pose estimation. For the monocular pipeline, keypoints are detected in a bounding box using "keypoint regression". The 3D pose of a cone is then estimated via the Perspective-n-Point (PnP) algorithm. For the stereo pipeline, features are matched in corresponding bounding boxes and triangulated to get an 3D position estimates of cones.

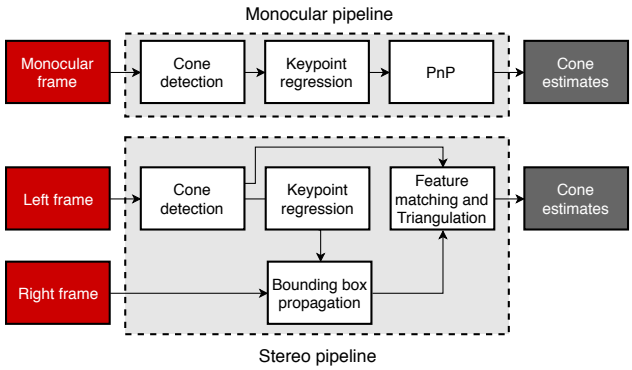


Figure 12: Pipeline of cone detection and position estimation using monocular and stereo cameras.

3.2.2 Multiple Object Detection

To estimate the 3D positions of multiple cones from a single image, it is necessary to detect these cones and classify them as blue, yellow or orange. A real-time and powerful object detector in the form of YOLOv2(Redmon and Farhadi, 2016) was trained on these three different types of cones. On being fed with the acquired images, this YOLOv2 returns the bounding box positions around the detected cones along with the confidence scores for each detection. This object detector network was chosen due to its robust outputs along with the ability to be fine-tuned with less data (using the ImageNet dataset (Deng et al., 2009)) and due to the existence of pre-trained weights which act as good priors during training.

3.2.3 Keypoint Regression

Next, one needs to retrieve the 3D positions from objects on the image. This in itself is not solvable using a single image, because of ambiguities due to scale and limited information. The ill-posed problem can be solved by leveraging additional prior geometric information of the objects along with the 2D information obtained from the image. A "keypoint regression" scheme is used, that exploits this prior information about the object's shape and size to regress and find specific feature points on the image that match their 3D correspondences whose locations can be measured from a frame of reference $\mathcal{F}_w$ (Dhall et al., 2019) as shown in Figure 13.

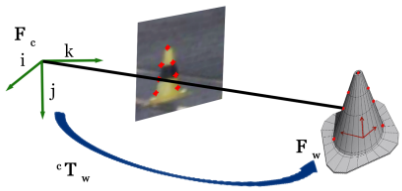


Figure 13: Schematic illustrating the matching of a 2D-3D correspondence and estimation of a transformation between the camera and world frames.

From Patches to Features: Classical Computer Vision Techniques

Initially, classical computer vision approaches were explored to extract these keypoints. Firstly, the bounding box is converted from the RGB color space to the LAB color space. In the LAB color space, the "L" channel represents the lightness whereas the "a" and "b" channel represent the green-red and blue-yellow components of the image respectively. In this color space, the "b" channel of the bounding box is extracted. An "Otsu" threshold is then applied on this transformed bounding box to obtain a binary image. Three vertices of the triangle were identified by performing contour fitting on the upper triangle of the binary image. Exploiting the 3D prior information of the cones, these three vertices were extended to obtain further four more keypoints. Figure 14 shows the extracted seven keypoints. The last image in Figure 14 shows the erroneous keypoints obtained in an edge case scenario obtained using this approach. Therefore, to make the keypoint extraction robust in such cases, the neural network described below is used.

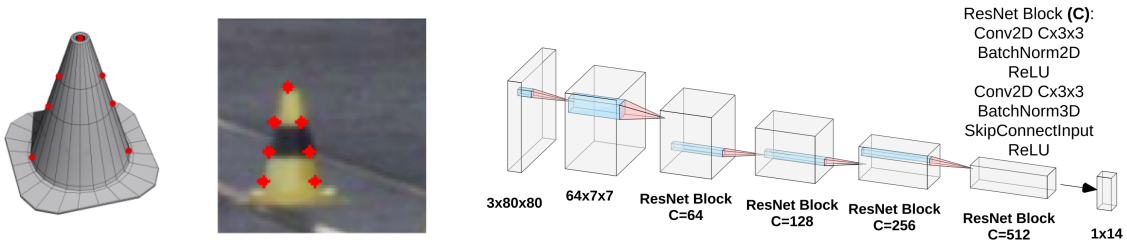
(a) (b) (c) (d) (e)

Figure 14: Classical computer vision approaches for keypoint extraction. (a) "b" channel of LAB color space (b) Contour fitting on upper triangle (c) 3 defining vertices of cone (d) Extracted keypoints (e) Erroneous output when multiple cones are present.

From Patches to Features: Keypoint Regression

The customized CNN (Dhall et al., 2019) detects "corner-like" features given a 80×80 image patch input. Figure 15a, shows the keypoints in 3D and 2D. We measure their 3D location relative to the base of the cone ($\mathcal{F}_w$). The keypoints are chosen such that they have a similar flavor of "corners-like" features from classical computer vision. The patches passed for the keypoint regression are bounding boxes detected by the object detector in the previous sub-module. The output vector is interpreted as the (X, Y) coordinates of the keypoints for a given input patch. A part of ResNet (He et al., 2016) is used as the backbone. The architecture applies *same* convolutions using a 3×3 kernel via a residual block (see PyTorch (Paszke et al., 2017) for more details). Using residual blocks reduces the chance of overfitting as they are capable of learning an identity transformation. The input tensor is passed through a convolution layer (with batch-norm) and non-linearity in the form of rectified linear units (ReLU). Batch-normalization makes the network robust to bad initialization. Following the batch-norm block, the signal passes through the following residual blocks $C = 64$, $C = 128$, $C = 256$ and $C = 512$ (refer to Figure 15b). A final fully-connected linear layer predicts the location of the keypoints.

With more convolution layers, the output feature tensor has more channels and smaller spatial dimensions. The final feature maps contain more global information than local information (Ronneberger et al., 2015). For regression, however, location specific information is essential. Using a ResNet-based architecture prevents drastic reduction of spatial resolution as the input volume is processed deeper into the network.



(a) Regressed keypoints are represented using red markers.

(b) The "keypoint network" (Dhall et al., 2019) takes a sub-image patch of $80 \times 80 \times 3$ as input and maps it to coordinates of the keypoints.

Figure 15: Keypoints are regressed from a detected box with a custom CNN and used to estimate the 3D position of a cone.

3.2.4 Monocular 3D Position Estimation

The camera frame is defined as $\mathcal{F}_c$ and the world frame as $\mathcal{F}_w$. We choose $\mathcal{F}_w$ to be at the base of the cone, for convenience of calculation of the 3D location of the keypoints.

Detected keypoints are used as 2D points on the image to make correspondence with the respective points on the 3D model of a cone. Using the correspondences and the camera intrinsics, PnP is used to estimate the pose of every detected cone. This works by estimating the transformation ${}^{c}\mathcal{T}_{w}$ between the camera coordinate system and the world coordinate system. Since, the world coordinate system is located at the base of the cone, lying at an arbitrary location in $\mathbb{R}^3$ this transformation is exactly the pose that needs to be estimated.

To estimate the position of the cone accurately, the non-linear version of PnP is used to obtain the transformation. RANSAC PnP is used instead of vanilla PnP, for the occasional incorrect or noisy correspondences. PnP is performed on the keypoint regressor output and the pre-computed 3D correspondences. Using the above pipeline, the pose transformation of multiple objects can be estimated using just a single image.

3.2.5 Stereo 3D Position Estimation

The pipeline for cone detection and pose estimation using stereo cameras is illustrated in Figure 12. Unlike a conventional stereo pipeline where object detection is done on both the images of the stereo pair, here it is done only on the left frame of the stereo pair using the YOLOv2 network as mentioned in Section 3.2.2. The bounding boxes in the left camera frame are then propagated to the right camera frame using stereo geometry. Finally SIFT features are matched between corresponding bounding boxes and triangulated for position estimates.

Bounding Box Propagation

With the knowledge of the location of the cone in the left camera frame, an approximation of the position of the same cone in the right camera frame has to be estimated. Therefore, the extracted keypoints are projected into the 3D space using the PnP algorithm as described in Section 3.2.4. Consequently, this 3D position is projected into the image plane of the right camera using the epipolar constraints and an approximate bounding box is drawn on the right camera frame. Therefore, using this approach, a bounding box detected in the left image frame is propagated spatially to the right camera frame which is depicted in Figure 16.

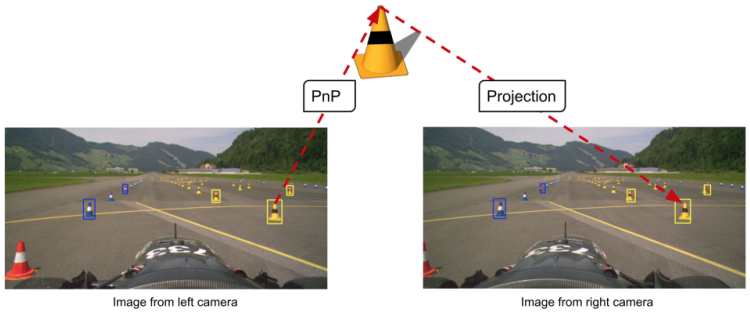


Figure 16: Propagation of a bounding box from the left frame to the right frame.

Feature Matching and Triangulation

Although the bounding box has been approximated in the right camera frame, it is challenging to extract the position of the exact seven keypoints just from propagation of the bounding box. This is attributed to the offset caused by the 3D-2D projection errors. Therefore, there is a need of feature matching between the bounding box pair obtained from the left and the right image frames. BRISK and SIFT feature extraction techniques were explored and the SIFT feature extractor was selected as it was observed to perform better at admissible computational load. Fewer mismatched features were found in SIFT feature matching as compared to BRISK feature matching as illustrated in Figure 17. As a result, the median of positions estimated by triangulating SIFT features gives more accurate estimates of cone positions than BRISK features.

Figure 17: Comparison of feature matching obtained from BRISK and SIFT feature descriptors.

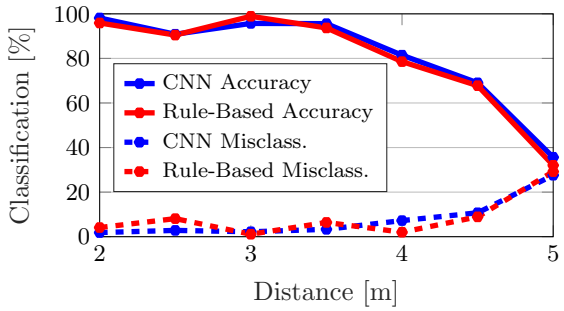
Given the feature descriptors of the bounding box of the left and right image frames, a "KNN" feature matching technique is applied to obtain the closest matched descriptors. Consequently, triangulation is performed on each of these matched descriptor pairs to obtain the cone's accurate 3D position estimate. For robustness against outliers, median filtering is applied on the obtained 3D position estimates and a single 3D position estimate is calculated.

3.3 Perception Validation & Results

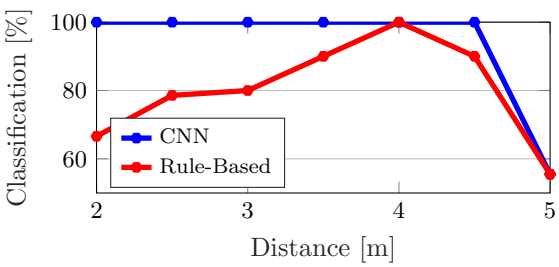
3.3.1 LiDAR-based Results

The color estimation module is validated by comparing the performance of the CNN and a rule-based approaches. In addition to that, we show a map created with our SLAM technique (see Section 4.2) with only LiDAR cone and color observations in order to validate the color estimation over the period of one lap.

Figure 18 compares the performance of the CNN and a rule-based approaches on cones of the same type (data which are similar to one used for training) as well as different type as those used to train the CNN respectively. The rule-based approach works by computing the direction of change in the intensity gradient along the vertical axis. A positive change followed by a negative one implies that the cone is blue in color whereas a negative change followed by a positive one represents a yellow cone. For test data with cones of the same type, the CNN and the rule-based approaches yield comparable results with both achieving a prediction accuracy of 96% for cones that are close-by, as illustrated in Figure 18a. However, the superior performance of the CNN is evident when both approaches are evaluated using test data that consists of cones of a different type. These cones have different point cloud intensities than usual due to the presence of an additional *FSG* sticker on the cone. The rule based approach reports a large number of misclassifications whereas the CNN is hardly affected by it. Furthermore, due to the use of an asymmetric cross-entropy loss function, despite the fact that classification accuracy is low at large distances, the miss-classification rate is small.



(a) Case 1: The network is evaluated using test data which are similar to the images that the model was trained on.



(b) Case 2: The network is evaluated using test data which are different from the images the model was trained on.

Figure 18: Comparison of classification performance of the CNN and Rule-Based approaches.

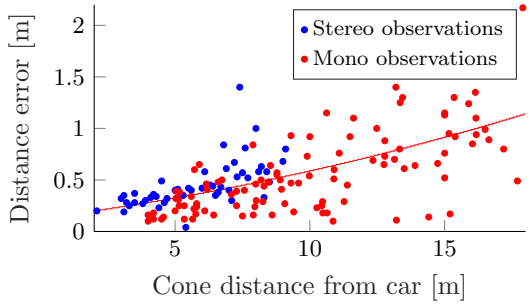
Figure 20a shows a map created by SLAM (see Section 4.2) using the cone position and color estimates from LiDAR alone. All the cones in the map are accurately colored, thus proving the accuracy and reliability of the color estimation module. The cones near the start line are deliberately not colored in order to prevent any potential misclassifications, caused by orange cones, during the start.

3.3.2 Camera-based Results

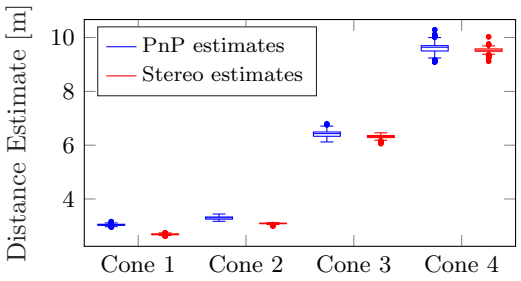
We compare the 3D positions estimates from the vision pipeline with those from an accurate range sensor such as the LiDAR, as can be seen in Figure 19a. As expected, the error increases with distance to the cone.

However, it is less than 1.0 m at a distance of 15 m.

An experiment was conducted to compare the 3D positions of cones estimated via PnP and those estimated via triangulation. For this experiment setup, cones were placed in front of the car at various distances and then images were actively acquired. The estimated variance of each cone was observed over time and plotted as a box plot in Figure 19b. This figure shows that the variance in the position estimates is higher for positions estimated via PnP (mono pipeline) and this is mitigated through triangulation (stereo pipeline).

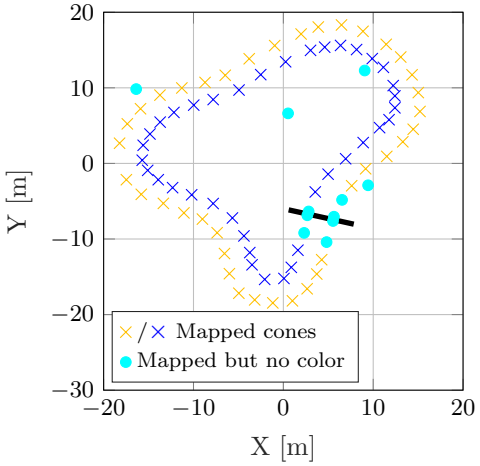


(a) The distance errors between LiDAR and vision estimated cone position.

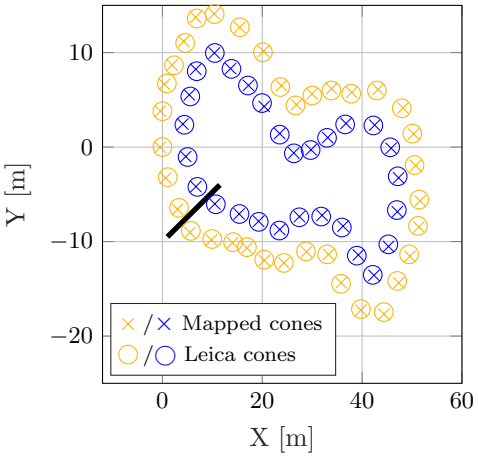


(b) The variance of 3D depth estimation obtained through PnP and through triangulation.

Figure 19: Comparison of different 3D estimation techniques and comparison of lidar and vision cone observation pipeline.



(a) A map output from SLAM using position and color information from LiDAR alone.



(b) Comparison of stereo 3D estimate of stereo system and the ground truth.

Figure 20: Maps created with SLAM using inputs from either only vision or LiDAR based pipelines. In both figures, the black line indicates the starting line.

Further, we illustrate the quality of a map generated using cones from the vision pipeline. We measured a ground truth of cone positions with the Leica Totalstation, shown in Figure 20b, as well as created a map with our SLAM algorithm (see Section 4.2) using only vision-pipeline cone estimates. These two maps were then aligned using the Iterative Closest Point (ICP) (Chetverikov et al., 2002). The Root Mean Square Error (RMSE) between cone estimates and its corresponding ground-truth is estimated to be 0.25 m.

4 Motion Estimation and Mapping

4.1 Velocity Estimation

Robust and accurate velocity estimation plays a critical role in an autonomous racecar. The estimated velocity is used to compensate the motion distortion in the LiDAR pipeline, propagate the state in the SLAM algorithm, as well as for the *control* module. Velocity estimation needs to combine data from various sensors with a vehicle model in order to be robust against sensor failure and to compensate for model mismatch and sensor inaccuracies. This problem is addressed with a 9 state Extended Kalman Filter (EKF), which fuses data from six different sensors.

4.1.1 Estimator Design

State estimation is a common task known in robotics. The main challenge is to fuse data from different measurements, each having different bias, noise and failure characteristics, together with the prior of the robot motion model. We propose to use an Extended Kalman Filter (EKF), the well-known estimator for mildly non-linear systems with Gaussian process and sensor noise, due to its computational efficiency, and accurate estimates (Thrun et al., 2005). An overview of the velocity estimation algorithm is shown in Figure 21.

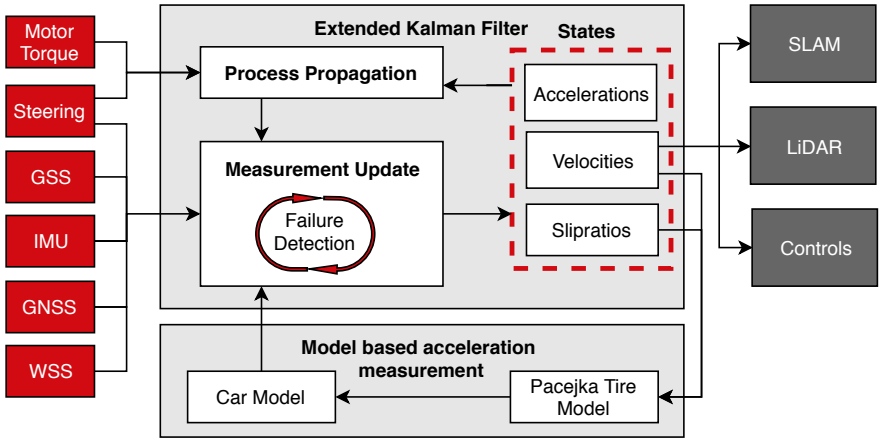


Figure 21: A simplified velocity estimation architecture.

Process Model

The process model represents a prior distribution over the state vector. In our case, a constant acceleration process model is used since jerk is close to a zero mean Gaussian distribution. Hence, the velocities are propagated using the acceleration whereas slip ratios are propagated using the dynamics derived by time differentiation of the slip ratio (Appendix A.2), resulting in the following process model,

$$
\begin{aligned}
\dot{\mathbf{v}} &= \mathbf{a} + r\left[v_y, -v_x\right]^T + \mathbf{n_v}\,, \\
\dot{r} &= f_M(\mathbf{sr}, \mathbf{v}, r, \delta) + n_r\,, \\
\dot{\mathbf{a}} &= \mathbf{n_a}\,, \\
\dot{\mathbf{sr}} &= R\left(\mathrm{diag}(I_\omega \mathbf{v}_{wx})^{-1}\mathbf{t}_M\right) + \left(\frac{C_\sigma R}{I_\omega} - a_x\right)\left(\mathrm{diag}(\mathbf{v}_{wx})^{-1}\mathbf{sr}\right) - a_x \mathrm{diag}(\mathbf{v}_{wx})^{-1}\mathbf{1}_{4\times 1} + \mathbf{n_{sr}}\,,
\end{aligned}
\tag{2}
$$

where $f_M(\cdot)$ describes the yaw moment generated from tire forces which are in turn estimated using a linear function of longitudinal and lateral slip (see Chapter I (Doumiati et al., 2013)). $\mathbf{t}_M \in \mathbb{R}^4$ and δ are the motor

torques and steering angle respectively, and are the inputs to the process model. C_σ is the longitudinal tire stiffness (Doumiati et al., 2013) providing a linear relationship between slip ratio and longitudinal force and is estimated from experiments. R is the radius of the wheel. $\mathbf{v}_{wx} \in \mathbb{R}^{4\times1}$ are the longitudinal velocity of the wheel hubs, I_ω is the moment of inertia of the wheel, and $\mathbf{n}_{\{.\}}$ is independent and identically distributed (i.i.d.) Gaussian noise. All these build a state vector $\mathbf{x} = [\mathbf{v}^T, r, \mathbf{a}^T, \mathbf{sr}^T]^T \in \mathbb{R}^9$ with $\mathbf{v} = [v_x, v_y]^T$ and r being the linear and angular velocities, respectively and $\mathbf{a} = [a_x, a_y]^T$ represent the linear accelerations. Finally, the four wheel slip ratios are denoted by $\mathbf{sr} = [\mathrm{sr}_{FL}, \mathrm{sr}_{FR}, \mathrm{sr}_{RL}, \mathrm{sr}_{RR}]^T$ where subscript F/R means front or rear axle of the car respectively and R/L denotes the right or left side of the car respectively. $\mathrm{diag}(\mathbf{x})$ represents a diagonal matrix consisting from elements of vector $\mathbf{x}$ and $\mathbf{1}_{4\times1} \in \mathbb{R}^{4\times1}$ is a vector of ones. This process model captures the fact that the probability of slippage increases with increased motor torques.

Note that acceleration is part of the state vector. This is chosen in order to fuse it from multiple measurements which makes it robust and redundant with respect to the acceleration measurement alone. To reduce the uncertainty due to the change in surface inclination and time varying bias, the acceleration is fused from two sources, the tire slip model and the Inertia Measurement Unit (IMU).

Measurement Model

Since every sensor is running at their own frequency, the measurements arrive asynchronously. In addition, we assume that the sensor noise is uncorrelated among different sensors. This allows us to run the correction steps for each sensor independently. As shown in Figure 21, the EKF is updated using velocity measurements $\mathbf{h_v}(\mathbf{x})$ of GSS and GNSS measurements, acceleration measurements $\mathbf{h_a}(\mathbf{x})$ from IMU as well as the tire slip model(Pacejka and Bakker, 1992), yaw rate measurements $h_r(\mathbf{x})$ from gyroscope and wheel speed measurements $\mathbf{h}_\omega(\mathbf{x})$ from the resolvers. This sums up to the following measurement model

$$\begin{aligned}
\mathbf{z_v} &= \mathbf{h_v}(\mathbf{x}) = \mathbf{R}(\theta_s)(\mathbf{v} + r[-p_{s,y}, p_{s,x}]^T) + \mathbf{n_{z_v}}\,, \\
z_r &= h_r(\mathbf{x}) = r + n_{z_r}\,, \\
\mathbf{z_a} &= \mathbf{h_a}(\mathbf{x}) = \mathbf{a} + \mathbf{n_{z_a}}\,, \\
\mathbf{z}_\omega &= \mathbf{h}_\omega(\mathbf{x}) = \frac{1}{R}\mathrm{diag}(\mathbf{v}_{wx})(\mathbf{sr} + \mathbf{1}_{4\times1}) + \mathbf{n_{z_\omega}}\,,
\end{aligned} \tag{3}$$

where $\mathbf{R}(\theta_s)$ denotes the rotation matrix with θ_s being the orientation of the sensor in the car frame and $[p_{s,x}, p_{s,y}]$ are the coordinates of the sensor also in the car frame.

To incorporate the slipratio into the wheelspeed measurement update, they are split into two components, linear hub velocity and slipratio in the measurement model explained in Appendix A.2. Both the components are updated simultaneously using wheelspeed measurement.

The Non-linear Observability analysis (Appendix A.1) shows that the system is still observable if either one of GSS or GNSS fails. All possible combinations of sensor configuration can be found in Table 2. If the IMU fails along with one of the velocity sensors (GSS or GNSS), then the yaw rate becomes unobservable. To avoid this, the model is converted from a full dynamic model to a partial kinematic one using a *zero slip ratio measurement update* (ZSMU). High changes in wheel speeds are still captured as slip by the process model and the ZSMU later shrinks the slip ratio in the update step of the EKF, which is reliable at low car speeds ($< 6\,\mathrm{m\,s^{-1}}$).

4.1.2 Failure Detection

Inaccurate and faulty sensor measurements result in temporarily or even permanently disturbed state estimates, given the recursive nature of the algorithm. It is therefore important to detect such sensor failures. The sensor faults can be classified as *outlier* (e.g., spikes in measurements), *drift* and *null*. A null measurement is defined as not receiving an input from the sensor, which is addressed by updating the measurement using the respective callback. A Chi-square-based approach for outlier detection, and a variance based sensor

Sensors				Result
GSS	IMU	GNSS	WSS	Observability
✓	✓	✓	✓	✓
×	✓	✓	✓	✓
×	×	✓	✓	✓*
×	✓	×	✓	✓*
×	✓	✓	×	✓*
×	×	×	✓	✓*
×	×	×	×	×

Table 2: Results are obtained by removing sensors successively. ✓ denotes that a sensor is available, × denotes that a sensor is not available, and ✓* denotes that the full state of the original system is not observable and the presented ZSMU is applied, to observe the reduced system.

isolation for drift detection are implemented. The details for the Chi-square-based approach can be found in (de la Iglesia Valls et al., 2018).

The Chi-square test works flawlessly for outliers. But sensor drift is only detected after a long time when the error is large. Such failure cases are identified using a variance based sensor drift detection given by

$$\sum_{i=1}^{n}(z_i - \mu_z)^2 < k\,, \tag{4}$$

where μ_z represents the mean of the sensor measurement. For each measurement, the variance is calculated using n number of sensors which were used to measure that state variable. If the variance is higher than the parameter k, the sensors are removed progressively, until the sensor with the highest contribution to the variance is rejected in the given time instant. The drift detection can also detect outliers but it requires measurement of a state variable from more than 2 sensors to work effectively. Hence, both outlier and drift detection are implemented alongside each other.

4.2 Simultaneous Localization and Mapping

The Formula Student competition poses multiple distinct challenges for mapping. Since the track is demarcated by cones only, a feature-based map representation is preferred over a grid-based one. Landmarks cannot be distinguished using unique descriptors, as the cones are described only by their position and color. As a result, the algorithm needs to handle uncertain data association. The SLAM algorithm must operate in real-time and the run-time should be predictable and even adjustable. Testing time is a major limiting factor for the whole system performance. Hence, it is desired to have an easily tunable algorithm that quickly achieves its full potential. Since environment conditions and sensor failures are unforeseeable, the algorithm should be able to detect malfunctioning sensors and proceed accordingly. Bearing the above criteria in mind, the fastSLAM 2.0 (Montemerlo et al., 2003) algorithm was chosen. Its particle filter nature inherently provides multi-hypothesis data associations. Additionally, it is easy to trade-off its performance versus run-time by adjusting the number of particles. FastSLAM scales linearly in the number of landmarks and therefore provides superior computational performance compared to the quadratic complexity of an EKF-SLAM algorithm (Montemerlo et al., 2002).

As illustrated in Figure 22, the implementation is divided into two parts, a localizer which processes velocities and continuously integrates them giving a pose update at 200Hz and a mapping algorithm. Processing the landmark observations of both perception pipelines and the cars velocity estimate result in a high frequency update of the corresponding map. The algorithm details can be found in Appendix A.3.

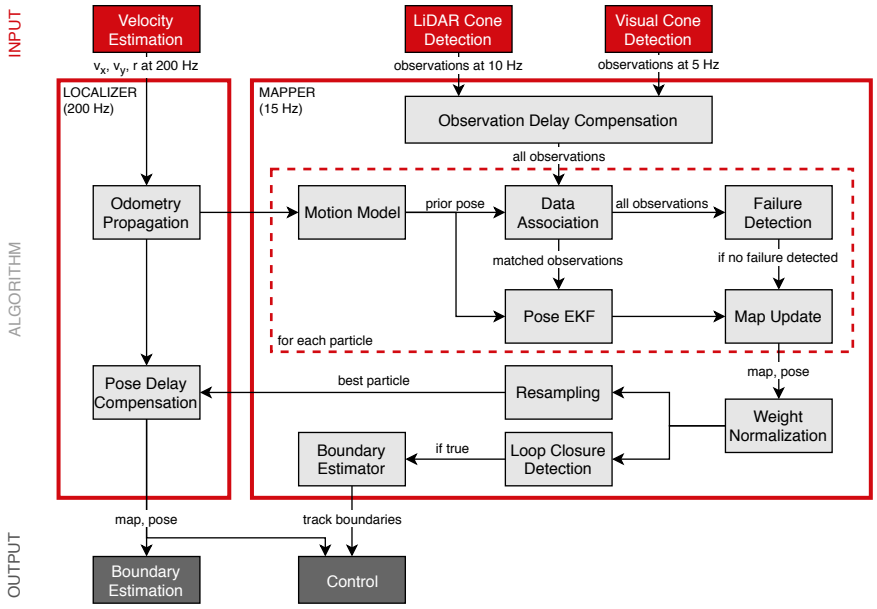


Figure 22: Detailed SLAM architecture used to fuse landmark observations from camera and LiDAR with velocity estimates into a coherent map and pose within the map. The dashed line visually demonstrates which parts of the algorithm are computed on a particle basis.

Failure Detection

To benefit most from the redundant sensor setup, a two-stage sensor failure detection is implemented. The goal is to avoid an (irreversible) map update due to observations coming from a malfunctioning sensor, penalizing measurements that do not match previous observations. In a first stage each landmark holds two counters, a recall counter n_s, which counts the number of times a landmark has been seen, and the second counter n_m, which is increased when a landmark is not seen, despite being in the sensors' FoV. If the confidence of a landmark as relation between both counters drops below a threshold, the landmark is assumed to be dubious. The second stage of the failure detection is on a sensor level. A set of observations from a sensor at a given time (after delay compensation) is only accepted if enough observations match with landmarks that have a landmark confidence above 80%, given there are any in the FoV.

Lap Closure Detection and Post-processing

After one lap, sufficient information is collected, a static map is assumed and further observations will not increase the quality of the map significantly. The car actively detects the completion of this first lap, a process we call lap closure detection. Lap closure is detected with three basic rules. First, the spread of all particles, measured by the sample standard deviation of the position, has to be below a threshold. Second, the cars current heading has to be roughly equal to the heading at the beginning of the race and third, all particles have to be close to the starting point.

Localization

After finishing the first lap, a lap closure is detected and the operational mode switches from *SLAM* to *Localization* mode. The track boundaries are then identified and the map is not longer updated. The map of the most likely particle (highest weight) is taken as the map for subsequent laps. In order to localize the car in the created map, Monte Carlo Localization (Dellaert et al., 1999) is used. A smooth pose update is given through the computation of the mean over all particles.

4.3 Boundary Estimation

The *SLAM* mode has two main goals, first to map the environment using the SLAM algorithm described in Section 4.2, and second to discover the track layout while remaining within the track boundaries. In this section, we discuss the second goal where we have to decide upon a path only given local cone observations. Note that the cones observations are probabilistic in both color and position. Additionally, false positive cones could be detected and the color may not be present at all. This makes this task challenging and means that our path planning algorithm needs to be able to deal with a high level of uncertainty.

In this section, we propose a path planning algorithm that uses the cone observations within the sensor range as guidance. Since the cones describe the track limits, it follows that the center line of the real track is in between two cones, which gives us a finite but potentially very large set of possible paths. This also means that there is no given goal position, which makes this a non-standard path planning problem. However, the cones naturally give rise to a graph search problem if an appropriate discretization is used.

4.3.1 Path Planning Algorithm

Our graph search path planning algorithm consists of three steps: first the search space is discretized using a triangulation algorithm, second, a tree of possible paths is grown through the discretization. Finally, all paths are ranked by a cost function and the path with the lowest cost is picked, which gives us an estimate of the boundary and the center line. Due to the limited sensor range, we run the path planning algorithm in a receding horizon fashion, re-estimating a new center line every 50 ms given new cone observations. In the following, these three algorithm steps are explained in detail.

The cones naturally form a discrete space, if the goal is to drive in between them. Thus, we discretize the X-Y space by performing a Delaunay triangulation (Delaunay, 1934) (Lee and Schachter, 1980), which subdivides the X-Y space into connected triangles. The vertices of the triangulation are the cone observations and the triangles are chosen such that the minimum internal angle of each triangle is maximized. Note that Delaunay triangulation is performed using the CGAL Library (The CGAL Project, 2018).

The second step of the algorithm consists of growing a tree of possible paths through the discretized space. Starting from the position of the car, the paths always connect to the next center points of the surrounding edges. The resulting paths describe all possible center lines given the observed cones and the current car positions. In each step, the tree is grown in a breath first manor. Furthermore, in order to avoid wasting memory and computation power, the worst paths (the ones with the highest cost) are pruned in a beam search fashion (Com, 2018). Thus, we only keep a fixed number of nodes. Finally, we only run a finite number of iteration before the algorithm is stopped. The number of iteration is determined experimentally to keep the computation time within the given limits and still guarantee a good coverage of all possible paths.

The final step of the algorithm computes a cost for each path generated in the second step and selects the one with the lowest cost as the most likely track center line. We choose a cost function which exploits information about the rules of the competition (see Section 2.1) including implications of these rules, as well as sensor knowledge (see Section 3). The cost function uses five cost terms, that are explained in Table 3 and always apply to a specific path.

To get the cost of a path, the five costs are first computed, then normalized, squared, weighted and finally added together. The best path is then selected as the estimate of the center line as well as the corresponding boundaries. In a final step, the estimated center line path is then given to a pure pursuit path following controller (Coulter, 1992), which generates the input to follow the center line.

Cost	Reason
Maximum angle change	Large angle changes from one path segment to the next are unlikely because even sharp corners are constructed using multiple cones
Standard deviation of the track width	The width is regulated by the rules and is unlikely to change a lot
Standard deviation of the distance between left as well as the right cones	Cones corresponding to the track are normally roughly space equal. This is not necessarily true if both tight corners and straights are present in the observed cones, however, this is unlikely given the limited sensor range and the cost helps to remove outliers
Maximal wrong color probability	If color information is given we know by the rules that left cones should be blue and right cones should be yellow, this cost uses the color probability given by the perception layer (see Section 3) to penalize paths that do not respect this rules. By using the color probability the cost becomes automatically zero if there is no color information
Squared difference between path length and sensor range	The cost penalizes too short and too long paths and gives an emphasize to paths that are the same length as the sensor range which is around 10 m

Table 3: Overview of cost structure used to find the track boundaries.

4.4 Motion Estimation and Mapping Validation & Results

4.4.1 Velocity Estimation Validation & Results

Figure 23 shows the estimated velocity, $v_{\text{estimated}}$, without the GSS compared to ground truth from GSS, v_{GSS}. Also, it shows the estimated slip ratio of rear left wheel, sr_{RL}, compared with slip ratio obtained from GSS and WSS, $\text{sr}_{RL,\text{GT}}$. It can be seen that the velocity estimate is accurate even when the wheels substantially slip. The very accurate velocity estimate is also shown when we compare the distance between the position of the car obtained by integrating the estimated velocity and the GPS position. The difference is less than 1.5 m over a 310 m long track, which results in a drift of less than 0.5 %.

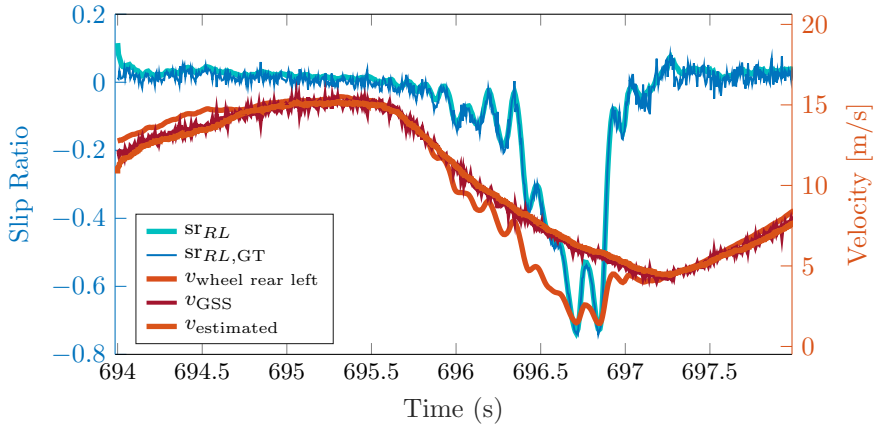


Figure 23: Estimated longitudinal velocity and slip ratio compared to ground truth data.

The ability to detect and further reject sensor failures is depicted in Figure 24. It can be observed that the chi-square based failure detection is able to reject the signal only when the failure is short-lived, whereas the drift failure detection is able to also discard continuous sensor failures. Using both techniques in conjunction ensures removal of most of the sensor failures.

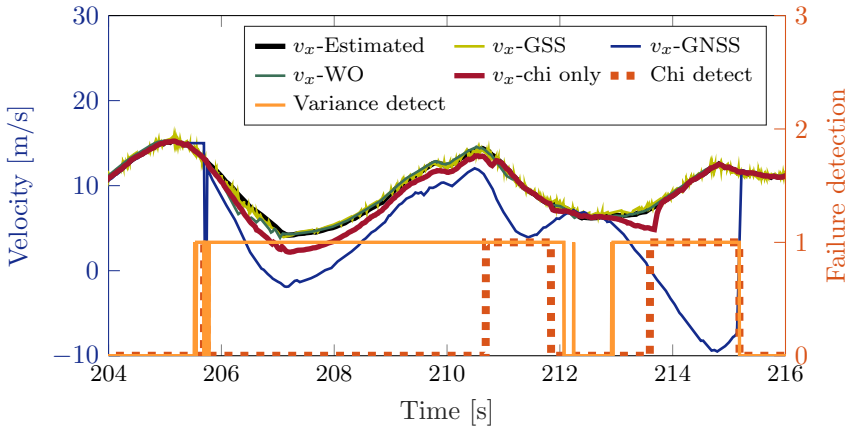


Figure 24: Failure of the GNSS (Blue) is detected by the drift detection (Orange). v_x-Estimated (chi only) (Dark Brown) is the estimated velocity using the chi test (Brown dashed) which shows that the chi test without drift detection is unable to detect the full sensor failure.

4.4.2 SLAM Validation & Results

Multiple results were already presented in Section 3.1 as well as Section 3.2 which show the estimated map with comparison to ground-truth measurements. By purely vision-based mapping, a RMSE of 0.25 m between estimated map and GT is achieved, whereas maps made with only LiDAR measurements gives a RMSE of 0.23 m.

The 230 m long track shown in Section 4.4.2 is mapped with both LiDAR and vision pipeline. In addition, the cars position is measured in order to verify the vehicle's particulate filter-based location. Therefore, a Totalstations measurement prism was attached to the car and measured for positional ground-truth. The estimated position differs only by a RMSE of 0.2 m from the tracked ground-truth.

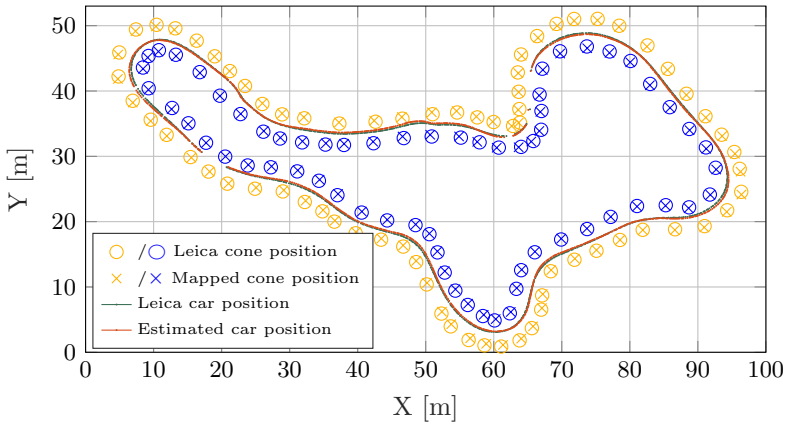
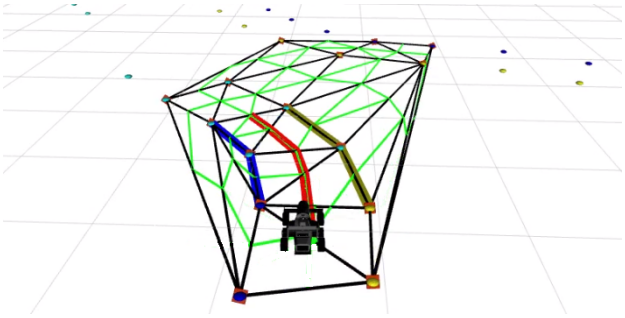


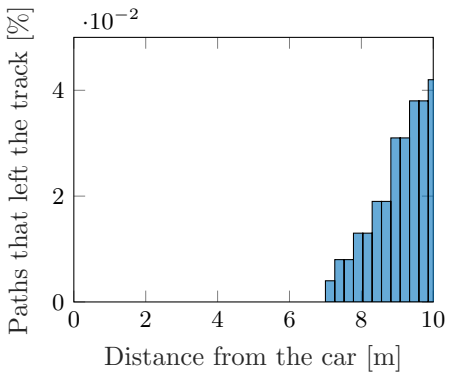
Figure 25: Mapping and localization comparison against ground-truth measurements.

4.4.3 Boundary Estimation Validation & Results

The result of all three steps of our proposed path planning algorithm are visualized in Figure 26a, where the car is able to find the proper path and track boundaries. The Delaunay triangulation given the observed cones as vertices is shown in black, and the tree of possible trajectories through the mid-points is shown in green. It is clearly visible that an exhaustive library of possible paths is generated using this approach, and that the cost function picked a path (shown in red) that is regularly spaced and of the correct length given the sensor range of the car.



(a) Depicted in black is the Delaunay triangulation, in green the tree of possible paths, in red the chosen path, and in blue and yellow the resulting boundaries for the chosen path.



(b) Percentage of paths outside of the real track at a certain distance from the car.

Figure 26: Evaluation of boundary estimation.

The boundary estimation algorithm is tested extensively under various track layouts as well as different road conditions. The system is robust when driving through rule compliant tracks. However, in case of distances larger than 5 m between two consecutive cones, we observed an increased error in predicting the track. In Figure 26b we show the classification success rate from the FSG trackdrive competition. For this track, only 4.2% of all iterations yield a path which is not inside the real track. Furthermore, the miss predicted paths only leave the real track at more than 7 m away from the car. At these distances, a miss-classified path does not cause the car to leave the track, since we run the algorithm in a receding horizon fashion, and only driving with $3\,\mathrm{m\,s^{-1}}$ in the *SLAM* mode. These results clearly show the robustness of the algorithm in the competing environment.

5 Control

After the first successfully finished lap, the complete track is mapped using the SLAM algorithm (see Section 4.2). Thus, the car now knows the track layout and can localize itself within the environment. Given this capability, we can race the car around a known track. Which brings us to our motion planning problem where the goal is to drive around the track as fast as possible. For this task, we extend the nonlinear MPC formulation of (Liniger et al., 2015). The formulation aims to maximize the progress along a given reference path (in our case the center line) while respecting the vehicle model and track constraints. The two main advantages of the proposed MPC is the direct consideration of the vehicle limits when computing the command and that the algorithm does not need any pre-determined logic, only the track layout, and the vehicle model. Note, that we extend the formulation of (Liniger et al., 2015) in several ways, first, by using a novel vehicle model that is well behaved and accurate also when driving slow. Second, by including simple actuator dynamics and finally by including tire friction constraints. We also directly solve the nonlinear program instead of relying on a convex approximation scheme.

In this section we first present our novel vehicle model which is suited for slow and fast driving while being tractable in a real-time MPC framework, furthermore, we explain the MPC formulation introduced in

(Liniger et al., 2015), with a focus on parts which are novel for this paper.

5.1 Vehicle Model

The task of driving a vehicle at its operational limits is challenging due to the highly nonlinear behavior in this operation range. Therefore, we model the dynamics of our racecar as a dynamic bicycle model with nonlinear tire force laws. The model is able to match the performance of the car even in racing conditions, while at the same time being simple enough to allow the MPC problem to be solved in real-time. The used vehicle model is derived under the following assumptions, (i) the vehicle drives on a flat surface, (ii) load transfer can be neglected, (iii) combined slip can be neglected, and (iv) the longitudinal drive-train forces act on the center of gravity. The last three assumptions are valid since the used low-level controllers are designed to deal with these effects. The resulting model is visualized in Figure 27. The equation of motion is given by,

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\varphi} \\ \dot{v}_x \\ \dot{v}_y \\ \dot{r} \end{bmatrix} = \begin{bmatrix} v_x \cos\varphi - v_y \sin\varphi \\ v_x \sin\varphi + v_y \cos\varphi \\ r \\ \frac{1}{m}(F_{R,x} - F_{F,y} \sin\delta + m v_y r) \\ \frac{1}{m}(F_{R,y} + F_{F,y} \cos\delta - m v_x r) \\ \frac{1}{I_z}(F_{F,y} l_F \cos\delta - F_{R,y} l_R + \tau_{\mathrm{TV}}) \end{bmatrix}, \tag{5}$$

where the car has a mass m and an inertia I_z, l_R and l_F represent the distance form the center of gravity to the rear and the front wheel respectively, $F_{R,y}$ and $F_{F,y}$ are the lateral tire forces of the rear/front wheel, F_x is the combined force produced by the drive-train and τ_{TV} the additional moment produced by the torque vectoring system. The state of the model $\tilde{\mathbf{x}} = [X, Y, \varphi, v_x, v_y, r]^T$, consists of (X, Y) and φ the position and heading in a global coordinate system, as well as the (v_x, v_y) the longitudinal and lateral velocities, and finally the yaw rate r. The control inputs $\tilde{\mathbf{u}} = [\delta, D]^T$ are the steering angle δ and driving command D. The driving command replicates the pedals of a driver and $D = 1$ corresponds to full throttle and $D = -1$ to full braking. We denote this model as $\dot{\tilde{\mathbf{x}}} = \tilde{f}_{\mathrm{dyn}}(\tilde{\mathbf{x}}, \tilde{\mathbf{u}})$.

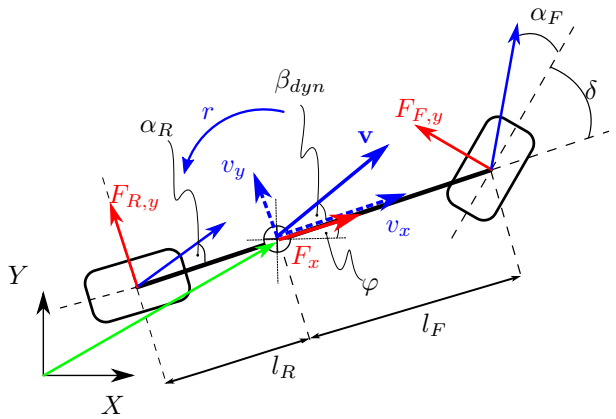


Figure 27: Bicycle Model: Position vectors are in depicted in green, velocities in blue, and forces in red.

The forces $F_{a,y}$ represent the interaction between tires and track surface where the subscript a refers to the front or rear tires, $a \in \{F, R\}$. A simplified Pacejka tire model (Pacejka and Bakker, 1992) was chosen, which meets the trade-off between precision and computational requirements,

$$\alpha_R = \arctan\left(\frac{v_y - l_R r}{v_x}\right), \qquad \alpha_F = \arctan\left(\frac{v_y + l_F r}{v_x}\right) - \delta, \tag{6}$$
$$F_{R,y} = D_R \sin\Big(C_R \arctan\Big(B_R \alpha_R\Big)\Big), \qquad F_{R,y} = D_F \sin\Big(C_F \arctan\Big(B_F \alpha_F\Big)\Big),$$

where $B_a, C_a, D_a, a \in \{R, F\}$ are experimentally identified coefficients of the model and $\alpha_a, a \in \{R, F\}$ are the rear and front slip angles.

In addition to the lateral tire models, a drive-train model (7) describes the longitudinal force F_x acting on the car depending on the applied driver command, $D \in [-1, 1]$,

$$F_x = C_m D - C_{r0} - C_{r2} v_x^2 \,, \tag{7}$$

which consists of a motor model, $C_m D$, rolling resistance, C_{r0}, and drag, $C_{r2} v_x^2$. The parameters $C_a, a \in \{m, r0, r2\}$ are identified from experiments. Note that this simplistic model is valid since the low-level controller takes care of the motor torque distribution to each wheel.

The final modelling component is the torque vectoring moment τ_{TV}. The moment is generated by the low-level controller distributing the requested force independently to the four motors. By doing so, an additional torque acting on the car can be generated. To include this effect the logic of the low-level controller is included in the model (5). The low-level controller, based on (Milliken et al., 1995), was developed for human drivers and is designed to mimic a kinematic car in order to achieve a more predictable behaviour. More precisely, the yaw moment is determined by a proportional controller, which acts on the error between the kinematic yaw rate r_{target} and the measured yaw rate r. Thus the torque vectoring yaw moment τ_{TV} is given by,

$$r_{\text{target}} = \delta \frac{v_x}{l_F + l_R} \,, \tag{8}$$

$$\tau_{\text{TV}} = (r_{\text{target}} - r) P_{\text{TV}} \,, \tag{9}$$

where a small angle assumption on δ is used, and P_{TV} is the proportional gain of the low level torque vectoring controller.

One problem of the dynamical bicycle model (5) is that the model is ill-defined for slow velocities due to the slip angles (6). However, slow velocities are important for race start and in sharp corners. For slow driving normally kinematic models are used which do not depend on slip angles. However, kinematic models are not suited for fast driving as they neglect the interaction of the tires and the ground. Therefore, to get the best of both models within one formulation we propose a novel vehicle model combining a dynamic and a kinematic vehicle model. To this end, we first formulate the kinematic model using the state of the dynamic model $\tilde{\mathbf{x}}$. The accelerations of the kinematic model are obtained through the differentiation of $v_{y,\text{kin}} = l_R r_{\text{target}}$ and $r_{\text{kin}} = \tan(\delta) \frac{v_x}{l_F + l_R}$, resulting in the following dynamics,

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\varphi} \\ \dot{v}_x \\ \dot{v}_y \\ \dot{r} \end{bmatrix} = \begin{bmatrix} v_x \cos\varphi - v_y \sin\varphi \\ v_x \sin\varphi + v_y \cos\varphi \\ r \\ \frac{F_x}{m} \\ (\dot{\delta} v_x + \delta \dot{v}_x) \frac{l_R}{l_R + l_F} \\ (\dot{\delta} v_x + \delta \dot{v}_x) \frac{1}{l_R + l_F} \end{bmatrix} \,, \tag{10}$$

where we assumed δ to be small, what simplifies the following terms, $\cos(\delta)^2 \approx 1$ and $\tan(\delta) \approx \delta$. The resulting model (10) is denoted as $\dot{\tilde{\mathbf{x}}} = \tilde{f}_{\text{kin}}(\tilde{\mathbf{x}}, \tilde{\mathbf{u}}, \dot{\tilde{\mathbf{u}}})$. Thus, both models are formulated using the same states which allows us to combine them. The resulting vehicle model is generated by linearly blended the two models, as follows

$$\begin{aligned} \dot{\tilde{\mathbf{x}}} &= \lambda \tilde{f}_{\text{dyn}}(\tilde{\mathbf{x}}, \tilde{\mathbf{u}}) + (1 - \lambda) \tilde{f}_{\text{kin}}(\tilde{\mathbf{x}}, \tilde{\mathbf{u}}, \dot{\tilde{\mathbf{u}}}) = \tilde{f}(\tilde{\mathbf{x}}, \tilde{\mathbf{u}}, \dot{\tilde{\mathbf{u}}}) \,, \\ \lambda &= \min\left(\max\left(\frac{v_x - v_{x,\text{blend min}}}{v_{x,\text{blend max}} - v_{x,\text{blend min}}}, 0 \right), 1 \right) . \end{aligned} \tag{11}$$

The models are only combined in the velocity range of $v_x \in [v_{x,\text{blend min}}, v_{x,\text{blend max}}]$; below we use purely the kinematic model while for velocities above $v_{x,\text{blend max}}$ we use purely the dynamic model. For our car we determined that this velocity range is from $v_{x,\text{blend min}} = 3$ to $v_{x,\text{blend max}} = 5\,\text{m/s}$.

Note that, the model now depends on the derivatives of the inputs $\dot{\tilde{\mathbf{u}}}$. We propose to use input dynamics to incorporate the input derivatives in the model. More precisely, we adopt the so-called Δ formulation in the MPC and assume that the inputs cannot be controlled directly but only their derivatives. Thus, the input dynamics are given by $\dot{D} = \Delta D$ and $\dot{\delta} = \Delta\delta$ which we summarize as $\dot{\tilde{\mathbf{u}}} = \Delta\tilde{\mathbf{u}}$, where $\Delta\tilde{\mathbf{u}} = [\Delta D, \Delta\delta]^T$ are the new control inputs. Thus the new model (11) is now properly defined. Furthermore, the input dynamics approximately model the real behavior of the steering system and the drivetrain, as well as allow to naturally constrain and penalize the input rates, among others implementing the rate limit of the steering system described in Section 2.

Tire Constraints

One of the assumptions in the dynamic bicycle model (5), is that the combined slip can be neglected. Combined slip occurs if the tire has to transform lateral as well as longitudinal forces. In such a case a real tire cannot produce the same maximum grip as in pure conditions. Simply said, a tire has a certain elliptic force budget it can transfer to the ground, often called the friction ellipse. This budget can be introduced as a constraint without explicitly modeling combined slip, which would require additional states. Thus, an ellipsoidal constraint on motor force at each tire $F_{R,x} = F_{F,x} = 0.5 C_m D$ (assuming equal front-rear force splitting) and $F_{a,y}$ is enforced,

$$\begin{aligned} F_{R,y}^2 + (p_{\text{long}} F_{R,x})^2 &\leq (p_{\text{ellipse}} D_R)^2\,, \\ F_{F,y}^2 + (p_{\text{long}} F_{F,x})^2 &\leq (p_{\text{ellipse}} D_F)^2\,, \end{aligned} \tag{12}$$

where p_{long} and p_{ellipse} are the tire specific ellipse parameters. The shape of the ellipse can greatly influence the driving style, e.g., with lower p_{long} the vehicle is allowed to corner more while accelerating, whereas with higher p_{ellipse} the tires are allowed to go closer to the limit.

These tire friction constraints are meant to ensure that the tire budget is not violated. Additionally, the low-level traction controller, which is based on (Bohl et al., 2014), ensures the motor torques are correctly distributed also considering load changes. Therefore, the combination of the tire constraints and the low-level controller make sure that the model assumptions are valid.

5.2 Contouring Formulation

The goal of the contouring formulation is to follow a reference path as fast as possible, in our case the center line of a track parametrized by the arc length. A third order spline is used to describe the path, as it offers a fast way to evaluate any point along the contour $(X_{\text{ref}}(\theta), Y_{\text{ref}}(\theta))$. To follow the path, the position of the car (X, Y) has to be linked to the position on the path, or in other words the arc-length. We call this arc-length $\theta_{\mathcal{P}}$ and it can be computed by projecting the cars position (X, Y) onto the reference path. However, computing the projection inside the MPC is computationally too expensive. We introduce θ to the model to approximates the true arc-length $\theta_{\mathcal{P}}$. To keep track of θ as well as the velocity and acceleration of the car relative to the reference path, a double integrator model is introduced, where $\dot{\theta} = v_\theta$ and $\dot{v}_\theta = \Delta v_\theta$. Note that the notation is used to highlight the similarity to the input dynamics of the vehicle model.

To ensure that θ is a good approximation of the true arc-length, a cost function is introduced which minimizes the approximation error along the path called the lag error $\hat{e}_l$ and the error perpendicular to the reference path called the contouring error $\hat{e}_c$ (see Figure 28). Based on the geometry of the reference path the costs

are given by,

$$\begin{aligned}\hat{e}_c(X,Y,\theta) &= \sin(\Phi(\theta))(X - X_{\text{ref}}(\theta)) - \cos(\Phi(\theta)(Y - Y_{\text{ref}}(\theta))\,,\\ \hat{e}_l(X,Y,\theta) &= -\cos(\Phi(\theta))(X - X_{\text{ref}}(\theta)) - \sin(\Phi(\theta)(Y - Y_{\text{ref}}(\theta))\,,\end{aligned} \tag{13}$$

with $\Phi(\theta)$ being the angle of the tangent to the reference path. Note that, if both costs are small θ is a good approximation of $\theta_{\mathcal{P}}$.

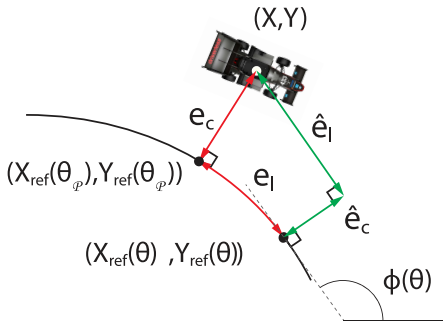


Figure 28: The approximate lead and lag errors with respect to the vehicle's position.

Finally, to follow the reference path as fast as possible the velocity along the path v_θ is maximized, which corresponds to maximizing the progress over the horizon. Combining these three costs, the cost function of the contouring formulation is defined as,

$$J(\mathbf{x}_k) = q_c \hat{e}_{c,k}^2 + q_l \hat{e}_{l,k}^2 - \gamma v_{\theta,k}\,, \tag{14}$$

where the individual objectives are weighted with $\gamma \geq 0$, $q_c \geq 0$ and $q_l \geq 0$. This allows to find the desired trade off between path following (minimize $\hat{e}_c$ and $\hat{e}_l$, defined in (13)) and lap time (maximizing v_θ).

Track Constraints

Finally, we consider track constraints, which limit the MPC to stay within the track boundaries. The constraint is formulated by locally approximating the track by circles centered on the middle line,

$$(X - X_{\text{cen}}(\theta))^2 + (Y - Y_{\text{cen}}(\theta))^2 \leq R_{\text{Track}}(\theta)^2. \tag{15}$$

Compared to the linear constraints used in (Liniger et al., 2015), these circular constraints are a convex inner approximation of the track, which increases the safety of the formulation. To reduce the computation time the position of the circles $(X_{\text{cen}}, Y_{\text{cen}})$ is determined given the previous solution of the MPC problem, by fixing θ.

5.3 MPC Problem Formulation

In summary, the dynamics of the problem are given by the vehicle model (11), the input dynamics, and the arc-length dynamics. To summarize them in one system, we define $\mathbf{u} = [\delta, D, v_\theta]^T$ and $\Delta\mathbf{u} = [\Delta\delta, \Delta D, \Delta v_\theta]^T$. This allows to formulate the full model used in the MPC as,

$$\dot{\mathbf{x}} = \begin{bmatrix}\dot{\tilde{\mathbf{x}}}\\ \dot{\theta}\\ \dot{\mathbf{u}}\end{bmatrix} = \begin{bmatrix}\tilde{f}(\tilde{\mathbf{x}}, \tilde{\mathbf{u}}, \Delta\tilde{\mathbf{u}})\\ v_\theta\\ \Delta\mathbf{u}\end{bmatrix} = f(\mathbf{x}, \Delta\mathbf{u})\,, \tag{16}$$

giving rise to a model with 10 states and 3 inputs. It is important to note, that in this formulation the control input is $\Delta\mathbf{u}$ and the steering, driving commands as well as v_θ are states. This final system of differential

equations (16) is discretized with Runge-Kutta 4th order integrator, resulting in the discrete time system $\mathbf{x}_{k+1} = f(\mathbf{x}_k, \Delta\mathbf{u}_k)$.

The MPC cost function consists of the previously defined contouring cost, $J_k(\mathbf{x}_k)$, as well as a control input penalization

$$R(\mathbf{u}_k, \Delta\mathbf{u}_k) = \mathbf{u}_k^T \mathbf{R}_\mathbf{u} \mathbf{u}_k + \Delta\mathbf{u}_k^T \mathbf{R}_{\Delta\mathbf{u}} \Delta\mathbf{u}_k.$$

In addition to that we introduce a side slip angle cost which controls the aggressiveness of the MPC, which is defined as,

$$L(\mathbf{x}_k) = q_\beta (\beta_{\text{kin},k} - \beta_{\text{dyn,k}})^2 \,,$$

where the squared error between the dynamic side slip angle, $\beta_{\text{dyn}} = \arctan(v_y/v_x)$, and the kinematic side slip angle $\beta_{\text{kin}} = \arctan(\tan(\delta) l_R/(l_R + l_F))$ is penalized with the weight $q_\beta \geq 0$, .

The inequality constraints consist of track (15), tire (12), input, and input rate constraints. The track constraints (15) are softened with a slack variable, S_c, and the corresponding cost $C(S_{c,k}) = q_s S_{c,k} + q_{ss} S_{c,k}^2$ is added, to guarantee feasibility at all times.

The final MPC problem with a prediction horizon of N looks as follows,

$$\begin{aligned}
\min_{\Delta\mathbf{u}} \quad & \sum_{k=0}^{N} J(\mathbf{x}_k) + R(\mathbf{u}_k, \Delta\mathbf{u}_k) + L(\mathbf{x}_k) + C(S_{c,k}) \\
s.t. \quad & \mathbf{x}_0 = \mathbf{x}(0), \\
& \mathbf{x}_{k+1} = f(\mathbf{x}_k, \Delta\mathbf{u}_k), \\
& (X_k - X_{\text{cen},k})^2 + (Y_k - Y_{\text{cen},k})^2 \leq R_{\text{Track},k}^2 + S_{c,k}, \\
& F_{R,y,k}^2 + (p_{\text{long}} F_{R,x,k})^2 \leq (p_{\text{ellipse}} D_R)^2, \\
& F_{F,y,k}^2 + (p_{\text{long}} F_{F,x,k})^2 \leq (p_{\text{ellipse}} D_F)^2, \\
& \mathbf{u}_{\min} \leq \mathbf{u}_k \leq \mathbf{u}_{\max}, \\
& \Delta\mathbf{u}_{\min} \leq \Delta\mathbf{u}_k \leq \Delta\mathbf{u}_{\max}.
\end{aligned} \tag{17}$$

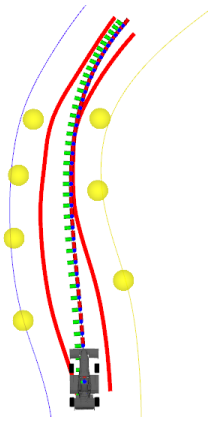
The MPC problem (17), is a nonlinear optimization problem which is solved using the commercial solver ForcesPro NLP (Zanelli et al., 2017; Domahidi and Jerez, 2014) in a receding horizon fashion. For our experiments we use a sampling time of 50 ms and a prediction horizon of 40 time steps, which corresponds to a look-ahead of 2 s. The predicted horizon of the MPC with corresponding track constraints can be seen in Figure 29a. One can notice that the predicted horizon is longer than the observed cones, which validates our need of a SLAM map. Furthermore, it can be seen that the MPC is driving a race line and does not only follow a middle line. More results of the achieved driving performance are given in Section 7.

5.4 Runtime Analysis

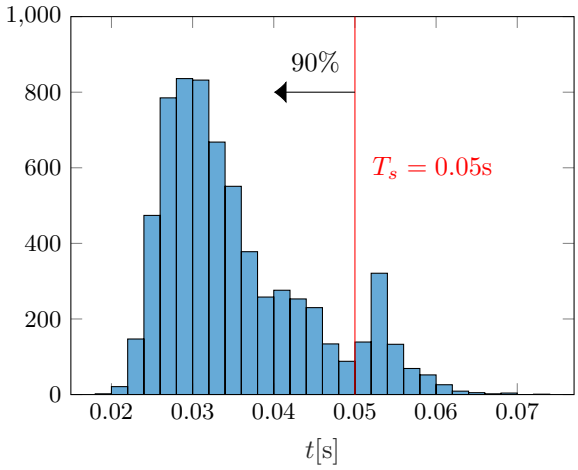
Since the used control technique is optimization based, one of the important factor is the solve time. The histogram of solve times achieved during the FSG trackdrive competition can be seen in Figure 29b. We can see that 90% of the solve times are below the sampling time, and at most, it takes 0.065 s to solve the nonlinear optimization problem. Due to the soft real-time structure of ROS, computation times above the sampling time do not cause problems, and no performance impact could be noticed in case of over times.

6 Testing Framework

To ensure that all modules work as designed, we formalized our testing procedures and created several tools to verify the correctness of our implementations. We made extensive use of automated simulations to test



(a) The predicted horizon of the optimization based controller shown with axes. The red lines represent maximal lateral constraints, blue and yellow line the right and left boundary respectively, and yellow larger circles the current observed cones.



(b) Solve time histogram of the used optimization problem.

Figure 29: Evaluation of used Model Predictive Control technique.

our code, as described in Section 6.1. In addition, both the simulation and testing generates considerable amounts of data. Our approach to handle the large number of logs and maximize its value is described in Section 6.2. We call the framework Automated Testing System[5] (ATS).

6.1 Simulation

Real-world testing with the racecar requires time and manpower what drastically limits the number of tests that can be performed. Simulation is used to catch problems before they happen on the race track and thus increase the efficiency of the actual testing. Simulations are ran automatically, triggered by pull requests into our main active code branch and every night several tests are run in simulation.

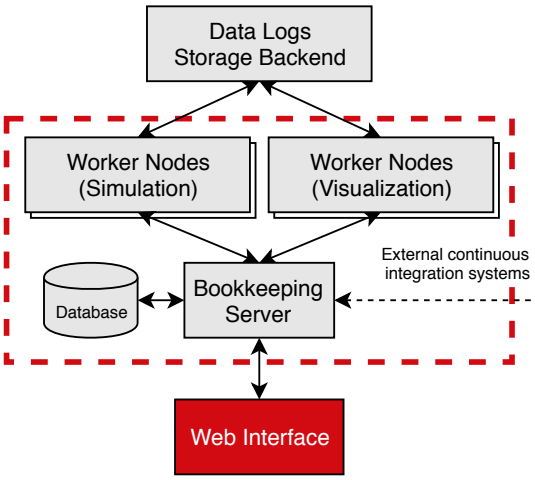
The used simulator is FSSIM[6] which is built upon the popular Gazebo software package (Koenig and Howard, 2004). FSSIM however does not make use of the Gazebo's included physics engine. Instead, an integrated first principle model is used. This allows us to match the simulation to the actual vehicle performance. The simulated race track surface is approximated as a flat plane with constant friction and can be automatically generated from log data. Perception sensors are not simulated because of large computational requirements. Instead, direct cone observations with various noise models are simulated. Simulation results are post processed to give a binary pass/fail and further lap statistics, i.e. lap times. Logged data is uploaded to the data analysis system and immediately processed into a set of visualizations (see Section 6.2). In case of a failure, notifications are sent out. This reduces the iteration time and increases the software quality.

6.2 Data Analysis & Visualization

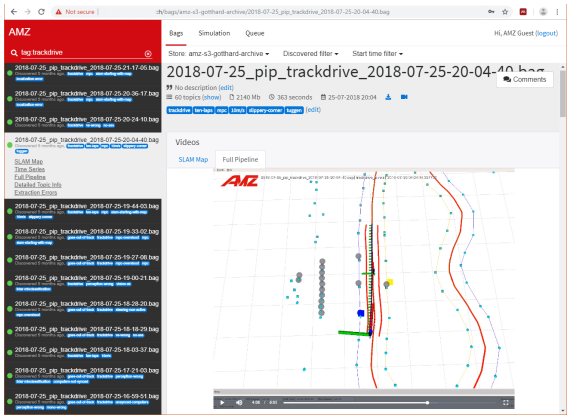
During the simulation and testing of the car, large amounts of data are recorded. The archiving, browsing and distribution of these data becomes increasingly difficult due to their volume and size. We designed a framework that automatically visualizes and indexes the logs. The framework is accessed through a web interface, shown in Figure 30b, which enables everyone to check results quickly. Since the software stack is built around ROS, the popular rosbag format is used for the data logs.

[5] https://github.com/AMZ-Driverless/rbb_core
[6] https://github.com/AMZ-Driverless/fssim

The framework is built in a distributed fashion with architecture depicted in Figure 30a. At the heart is a service that does the bookkeeping for all data logs and stores this in a PostgreSQL database keeping a queue of all tasks that need to be performed. These tasks are further distributed to worker nodes. Worker nodes poll the service to request scheduled tasks. This is preferred over a push based system to keep the system simple and stateless. The queue is made for a small number of long running tasks and not optimized for many small short running operations, so polling delay is negligible. The main tasks are simulation (as described in Section 6.1) indexing and visualization.



(a) Overview of the architecture of the automated testing framework. Communication is marked with arrows.



(b) Web-based visualization is easily accessible from everywhere.

Figure 30: Automated Testing System (ATS) overview.

Visualizations are either manually queued or scheduled upon discovery of new data logs. Each data log can contain different sets of data depending on where it was recorded and which tests were performed. To accommodate this, a matching system is used where sets of data topics are matched against pre-configured plugin setups. For the most important 3D visualization plugin we heavily rely on RViz [7]. The output on a virtual screen is recorded while the logs are being played back. This is a powerful primitive that can be used with any existing visualization software that needs to be automated. Each plugin can output a data object visible in the web interface (e.g., a set of images with plots or a recorded video).

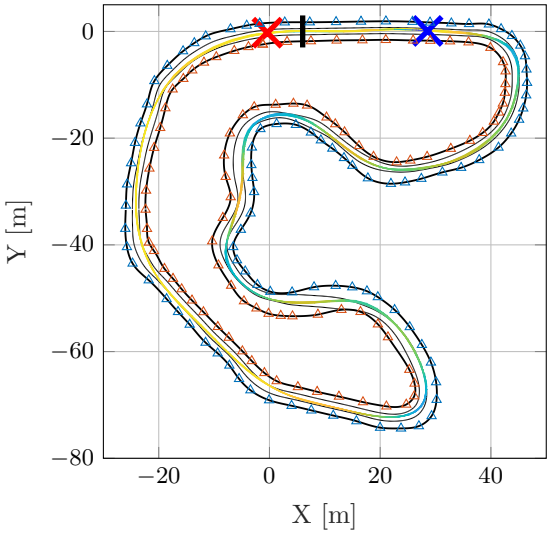
7 The Race Results

As final results, we show the performance achieved during 10 consecutive laps at the FSG 2018 competition. These can be seen in the Figure 31a. The starting position is indicated by the red cross, located 6 m before the start line. The vehicle autonomously drove 10 laps and successfully completed the event by coming to a safe stop (blue cross) within the allowed 20 m after the starting line. One can see in the shape of the driven path that the car slows down in sharper turns and accelerates at the exit of the turns, similarly to a human driver. Over the course of this run, the vehicle achieved 1.5g lateral acceleration which is close to the tire limit. The GG plot of the ten laps can be seen in Figure 31b. The squared shape is caused by conservative torque limits, which emphasis safety over pure performance. As illustrated in Table 4, the used optimization-based control technique results in fast and consistent lap times.

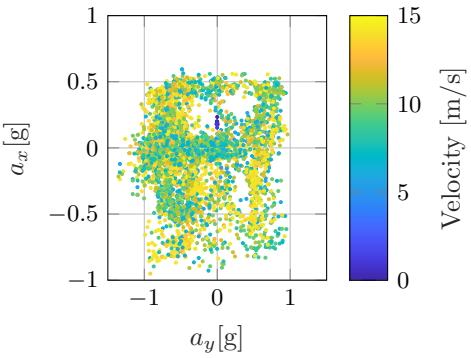
[7] `http://wiki.ros.org/rviz`

Lap Number	1	2	3	4	5	6	7	8	9	10
Lap Time [s]	28.913	28.608	28.558	28.483	28.625	28.542	28.523	28.571	28.512	28.593

Table 4: Achieved lap times on the FSG 2018 competition. Thanks to the optimization nature of the controller the lap times are very consistent.



(a) The driven line from the FSG 2018 competition with corresponding cone positions. The red cross represents the starting position, the blue cross the ending position, and the black thick line the starting line. From the driven path one can see that the vehicle cuts corners while driving slower in corners and faster on straights, as a human driver. Color of the trajectory indicates the velocity of the racecar.



(b) Lateral versus longitudinal accelerations measured at the CoG (GG plot) over ten laps.

Figure 31: The results achieved during FSG 2018 trackdrive discipline.

7.1 Lessons Learned

Over the course of the past two years, the design of our autonomous system grew significantly. However, our design principles largely stayed the same. Parts of these were brought in from previous projects, others resulted from failures. We would therefore like to share these with the reader.

First, we learned that hardware is important, and only real-world testing can thoroughly evaluate the whole system. Thus, testing is the most vital factor for success and it is therefore paramount to keep the development and testing cycles short in order to complete and evaluate features well in time for the competition. Simulation testing is also invaluable, as it allows to make real-world testing more efficient. This realization led to the development of the ATS (Section 6). Its ability to simulate multiple trials automatically made possible to find good parameters in simulation and only fine tune these at the testing spot. Moreover, ATS' powerful analysis tool helped us to quickly identify problems in both real-world and simulation testing.

Finally, when testing one needs to pay attention to extreme cases. For a racecar this means pushing the car to the limit, using aggressive driving, challenging track layouts, all during good and poor weather conditions. This, helps to better understand the car and allows us to successfully compete in competitions where it is key to go to the limit but not beyond.

Having a software architecture with fixed, clearly defined interfaces makes it possible to develop, simulate, and test each subsystem on datasets independently. However, the full pipeline should still be tested regularly

to verify that it works as a whole and that overall computational constraints are met.

Since the racecar can accelerate from 0 to 100km/h in under 2s and drives at significant speeds, safety is our primary concern when testing. We therefore developed a *safety calculator* to estimate the necessary distance between the track and obstacles, considering the racecar's maximum speed, human reaction time, and even communication delays. The calculated safety margins are used when building test tracks during our testing.

Finally, one of the biggest bottlenecks in our current architecture are delays, as shown in Figure 32. The time it takes from a cone detection until a corresponding control command reaches the car sums up to about 300 ms. When driving at the limits of handling, this can lead to severe instabilities e.g., the car missing a braking point what might result in leaving the track. Our current workaround is to limit our top speed and wheel torques, which causes us to fall behind human performance.

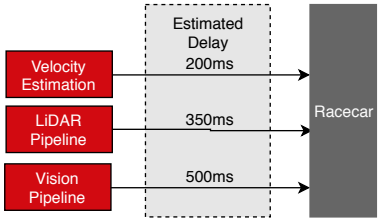


Figure 32: Total delay diagram of the proposed software-hardware architecture.

8 Conclusion

In this paper we showed that our autonomous racecar design enables autonomous driving on unknown tracks, create a map of them, and further drive faster after loop closure detection. The system was extensively tested and then used in FSD competition. In 2017, out of 15 teams attending the first FSG competition, ours was the only one able to finish all the disciplines. Furthermore, in 2018 we competed in FSI as well as FSG where our system outperformed our previous results and won both competitions beating numerous score records. The results showed the capabilities and scalability of the proposed software stack as well as the potential of the created tools.

At the time of writing, a new AMZ Driverless racecar is being designed and prepared for the 2019 season. The overall goal of the new team is to match the performance of a human driver. Therefore, the general focus lies on reducing and handling delays in a consistent way throughout the software stack. Furthermore, the perception group's primary aim is to increase the observation radius. The estimation group is working on detecting the track boundaries more reliably and over a longer horizon, creating globally consistent maps, and adding the ability to localize globally. In controls, the focus lies on solving the problem more efficiently, such that more accurate models can be used online, and on rigorous system identification. The software architecture group is working on detecting and recovering from subsystem failures online, as well as automating the testing process further. Formula Student Driverless is a unique opportunity for engineering students to learn about complex robotic systems and to deploy one under pressure in a competitive environment, just like in industry.

Acknowledgments

In the name of AMZ Driverless, we would like to thank all our sponsors as we could not have realized this project without their support. Furthermore, we would like to thank Marc Pollefeys, Andrea Cohen, and Ian Cherabier from ETH Zürich's CVG Group, Dengxin Dai from ETH Zürich's CVL Group and Mathias Bürki from ETH Zürich's ASL Group for the knowledge and helpful ideas that they contributed to the project.

References

(2018). FOLDOC - Computing Dictionary. http://foldoc.org/beam%2bsearch. Accessed: 2018-12-03.

(2018). Fsg competition handbook 2018. https://www.formulastudent.de/fileadmin/user_upload/all/2018/rules/FSG2018_Competition_Handbook_V1.1.pdf. Accessed: 2019-02-09.

Allen, P. E. and Holberg, D. R. (1987). *CMOS analog circuit design*. Oxford university press.

Bohl, D., Kariotoglou, N., Hempel, A. B., Goulart, P. J., and Lygeros, J. (2014). Model-based current limiting for traction control of an electric four-wheel drive race car. In *European Control Conference (ECC)*.

Buehler, M., Iagnemma, K., and Singh, S. (2008). Editorial. *Journal of Field Robotics*, 25(10):725–726.

Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I., and Leonard, J. J. (2016). Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age. *IEEE Transactions on Robotics*, 32(6):1309–1332. 00335.

Casanova, D., Sharp, R., and Symonds, P. (2000). Minimum time manoeuvring: The significance of yaw inertia. *Vehicle system dynamics*, 34(2):77–115.

Chetverikov, D., Svirko, D., Stepanov, D., and Krsek, P. (2002). The trimmed iterative closest point algorithm. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, volume 3, pages 545–548. IEEE.

Coulter, R. C. (1992). Implementation of the pure pursuit path tracking algorithm. Technical report, Carnegie-Mellon UNIV Pittsburgh PA Robotics INST.

Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1.

de la Iglesia Valls, M., Hendrikx, H. F. C., Reijgwart, V. J. F., Meier, F. V., Sa, I., Dubé, R., Gawel, A., Bürki, M., and Siegwart, R. (2018). Design of an autonomous racecar: Perception, state estimation and system integration. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2048–2055.

Delaunay (1934). Sur la sphére vide. *Bull. Acad. Science USSR VII:Class. Sci. Mat. Nat.*

Dellaert, F., Fox, D., Burgard, W., and Thrun, S. (1999). Monte carlo localization for mobile robots. In *IEEE International Conference on Robotics and Automation (ICRA99)*.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. Ieee.

Dhall, A., Dai, D., and Van Gool, L. (2019). Real-time 3D Traffic Cone Detection for Autonomous Driving. *arXiv e-prints*, page arXiv:1902.02394.

Domahidi, A. and Jerez, J. (2014). FORCES Professional. embotech GmbH (http://embotech.com/FORCES-Pro).

Doumiati, M., Charara, A., Victorino, A., and Lechner, D. (2013). *Vehicle Dynamics Estimation using Kalman Filtering*. ISTE Ltd and John Wiley & Sons, Inc.

Engel, J., Schps, T., and Cremers, D. (2014). LSD-SLAM: Large-Scale Direct Monocular SLAM. In *Computer Vision ECCV 2014*, volume 8690. Springer International Publishing, Cham.

Funke, J., Brown, M., Erlien, S. M., and Gerdes, J. C. (2017). Collision avoidance and stabilization for autonomous vehicles in emergency scenarios. *IEEE Transactions on Control Systems Technology*, 25(4):1204–1216.

Funke, J., Theodosis, P., Hindiyeh, R., Stanek, G., Kritatakirana, K., Gerdes, C., Langer, D., Hernandez, M., Mller-Bessler, B., and Huhnke, B. (2012). Up to the limits: Autonomous Audi TTS. In *2012 IEEE Intelligent Vehicles Symposium*, pages 541–547.

Goldfain, B., Drews, P., You, C., Barulic, M., Velev, O., Tsiotras, P., and Rehg, J. M. (2018). AutoRally An open platform for aggressive autonomous driving. *ArXiv e-prints*, page arXiv:1806.00678.

Gosala, N. B., Bühler, A., Prajapat, M., Ehmke, C., Gupta, M., Sivanesan, R., Gawel, A., Pfeiffer, M., Bürki, M., Sa, I., Dubé, R., and Siegwart, R. (2018). Redundant Perception and State Estimation for Reliable Autonomous Racing. *ArXiv e-prints*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Hermann, R. and Krener, A. J. (2004). Nonlinear controllability and observability. *International Journal of Humanoid Robotics*, AC-22(01):157–173.

Hess, W., Kohler, D., Rapp, H., and Andor, D. (2016). Real-time loop closure in 2d LIDAR SLAM. IEEE.

Himmelsbach, M., v. Hundelshausen, F., and Wuensche, H. . (2010). Fast segmentation of 3d point clouds for ground vehicles. In *2010 IEEE Intelligent Vehicles Symposium*, pages 560–565.

Iagnemma, K. and Buehler, M. (2006a). Editorial for journal of field robotics – Special issue on the darpa grand challenge. *Journal of Field Robotics*, 23(8):461–462.

Iagnemma, K. and Buehler, M. (2006b). Editorial for journal of field robotics – Special issue on the darpa grand challenge. *Journal of Field Robotics*, 23(9):655–656.

Kelly, D. and Sharp, R. (2010). Time-optimal control of the race car: a numerical method to emulate the ideal driver. *Vehicle System Dynamics*, 48(12):1461–1474.

Klomp, M., Olsson, K., and Sandberg, C. (2014). Non-linear steering control for limit handling conditions using preview path curvature. *International Journal of Vehicle Autonomous Systems*, 12(3):266–283.

Koenig, N. and Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2149–2154. IEEE.

Kritayakirana, K. and Gerdes, J. C. (2012). Autonomous vehicle control at the limits of handling. *International Journal of Vehicle Autonomous Systems*, 10(4):271–296.

Labb, M. and Michaud, F. (2019). RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. *Journal of Field Robotics*, 36(2).

Lee and Schachter (1980). Two algorithms for constructing a delaunay triangulation. *International Journal of Computer and Information Sciences*.

Liniger, A., Domahidi, A., and Morari, M. (2015). Optimization-Based Autonomous Racing of 1:43 Scale RC Cars. *Optimal Control Applications and Methods*, 36(5):628–647. arXiv: 1711.07300.

Liniger, A. and Lygeros, J. (2019). Real-time control for autonomous racing based on viability theory. *IEEE Transactions on Control Systems Technology*, 27(2):464–478.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. (2016). SSD: Single Shot MultiBox Detector. In Leibe, B., Matas, J., Sebe, N., and Welling, M., editors, *Computer Vision ECCV 2016*, pages 21–37, Cham. Springer International Publishing.

Milliken, W., Milliken, D., and of Automotive Engineers, S. (1995). *Race Car Vehicle Dynamics*. Premiere Series. SAE International.

Montemerlo, M. and Thrun, S. (2007). *FastSLAM 2.0*, pages 63–90. Springer Berlin Heidelberg, Berlin, Heidelberg.

Montemerlo, M., Thrun, S., Koller, D., and Wegbreit, B. (2002). *FastSLAM: A factored solution to the simultaneous localization and mapping problem*, volume 593598.

Montemerlo, M., Thrun, S., Koller, D., and Wegbreit, B. (2003). FastSLAM 2.0 : An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges. In *Proceedings of the 18th international joint conference on Artificial intelligence*, pages 1151–1156.

Mur-Artal, R. and Tardos, J. D. (2017). ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262.

Pacejka, H. B. and Bakker, E. (1992). The magic formula tyre model. *Vehicle system dynamics*, 21(S1):1–18.

Paden, Cap, Yong, Yershov, and Frazzoli (2016). A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE TRANSACTIONS ON INTELLIGENT VEHICLES*.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788.

Redmon, J. and Farhadi, A. (2016). YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242.

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597.

Rosolia, U., Carvalho, A., and Borrelli, F. (2017). Autonomous racing using learning model predictive control. In *American Control Conference (ACC)*, pages 5115–5120. IEEE.

Tanzmeister, G., Friedl, M., Lawitzky, A., Wollherr, D., and Buss, M. (2013). Road course estimation in unknown, structured environments. In *2013 IEEE Intelligent Vehicles Symposium (IV)*, pages 630–635. IEEE.

The CGAL Project (2018). *CGAL User and Reference Manual*. CGAL Editorial Board, 4.13 edition.

Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press.

Tulsiani, S. and Malik, J. (2015). Viewpoints and keypoints. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Verschueren, R., De Bruyne, S., Zanon, M., Frasch, J. V., and Diehl, M. (2014). Towards time-optimal race car driving using nonlinear mpc in real-time. In *Conference on Decision and Control (CDC)*, pages 2505–2510. IEEE.

Viola, P. and Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–I.

Xue, Z. and Schwartz, H. (2013). A comparison of several nonlinear filters for mobile robot pose estimation. In *2013 IEEE International Conference on Mechatronics and Automation*.

Zanelli, A., Domahidi, A., Jerez, J., and Morari, M. (2017). FORCES NLP: an efficient implementation of interior-point methods for multistage nonlinear nonconvex programs. *International Journal of Control*.

Zhang, J., Xu, S., and Rachid, A. (2001). Sliding mode lateral motion control for automatic steering of vehicles. In *European Control Conference (ECC)*, pages 3636–3641. IEEE.

A Appendix

A.1 Observability Analysis

Observability analysis plays a pivotal role for design of a redundant filter. A system is said to be observable if the state $\mathbf{x}(t)$ can be fully determined (reconstructed) using measurements $\mathbf{z}(t)$ and control inputs $\mathbf{u}(t)$. If any sensor fails, it is important to understand if state estimate converges to the true value or not.

In case of non-linear models such as ours, the observability can be determined by performing the rank test on matrix $\mathcal{O}$ using Lie derivative (Hermann and Krener, 2004), which is defined recursively as:

$$L_f^{l+1}h(\mathbf{x}) = \frac{\partial L_f^l h}{\partial \mathbf{x}} f(\mathbf{x}, \mathbf{u}), \qquad (18) \qquad \mathcal{O}(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} \frac{\partial L_f^0 h(\mathbf{x})}{\partial \mathbf{x}}, \frac{\partial L_f^1 h(\mathbf{x})}{\partial \mathbf{x}}, ... \end{bmatrix}^T, \qquad (19)$$

with $L_f^0 h(\mathbf{x}) = h(\mathbf{x})$. The Observability matrix is defined in (19).

The system is weakly locally observable if $\mathcal{O}$ is full rank, and not observable otherwise. By iteratively removing measurements and performing the observability analysis it can be concluded that to estimate the full 9 states, a velocity measurement is needed. If all velocity measurement fail, the model is converted from a full dynamic model to a partial kinematic one assuming slip ratio is close enough to zero. In practice this assumption holds for speeds under $6\,\mathrm{m\,s^{-1}}$.

A.2 Slip Ratio Estimation

The racecar wheels can slip upto 30% at high accelerations and even higher under heavy braking. This leads to biased velocity estimates from wheel odometry. Wheel slip is therefore estimated and used to correct for misleading wheel odometry information.

The slip ratio is one of the measures to express slipping behavior of the wheel defined as

$$\mathrm{sr}_{ij} = \begin{cases} \frac{\omega_{ij} R - v_{ij}}{v_{ij}} & \text{if } |v_{ij}| > 0 \\ 0 & \text{if } |v_{ij}| = 0 \end{cases}, \qquad (20)$$

sr_{ij} denotes the slip ratio of wheel ij, where i $\in$ {Front, Rear} and j $\in$ {Left, Right}. Here, ω_{ij} and v_{ij} are the angular and linear velocities of wheel ij respectively. R is the radius of the wheel.

Wheelspeed measurement updates are split into two components, linear hub velocity and slipratio in the measurement model given by (21) and both the components are updated simultaneously using wheelspeed measurement in order to update slip ratio ratio in EKF. Here, b denotes the distance between right and left wheels, δ is steering angle, l_F and l_R are distance of front and rear axle respectively from the car center of gravity. r is the yaw rate of the car. This method requires the slip ratio to be a part of the state of the EKF.

$$\begin{bmatrix} \omega_{FL} \\ \omega_{FR} \\ \omega_{RL} \\ \omega_{RR} \end{bmatrix} = \begin{bmatrix} ((v_x - r\frac{b}{2})\cos(\delta) + (v_y + r\,l_F)\sin(\delta))\frac{(sr_{FL}+1)}{R} \\ ((v_x + r\frac{b}{2})\cos(\delta) + (v_y + r\,l_F)\sin(\delta))\frac{(sr_{FR}+1)}{R} \\ (v_x - r\frac{b}{2})\frac{(sr_{RL}+1)}{R} \\ (v_x + r\frac{b}{2})\frac{(sr_{RR}+1)}{R} \end{bmatrix} + \mathbf{n}_\omega. \qquad (21)$$

A.3 FastSLAM 2.0 Algorithm Details

The goal of the mapping phase is to create a 2D map of cones and to simultaneously localize within it. Each landmark λ_n consists of a Gaussian position estimate $[\mu_n, \mathbf{\Sigma}_n]$, a color estimate $c_n \in$

{yellow, blue, orange, unknown} and two counters $[n_{s,n}, n_{m,n}]$ which indicate how often the landmark is seen or not seen (missing) respectively. A map is then defined as a set of N landmarks:

$$\mathbf{\Lambda} = \left([\boldsymbol{\mu}_1, \mathbf{\Sigma}_1, c_1, n_{s,1}, n_{m,1}], ..., [\boldsymbol{\mu}_N, \mathbf{\Sigma}_N, c_N, n_{s,N}, n_{m,N}]\right). \tag{22}$$

Each particle S_k^m at time step k with particle index $m \in 1, ..., M$ carries a map and a pose, both defined in a global frame which is initialized at the beginning of the mapping phase.

Observations and Data Association

The map is updated when a new set of observations is received. For each particle, a new pose is sampled from a Gaussian distribution obtained using an EKF approach. The prior $[\boldsymbol{\mu}_{s_k,0}, \mathbf{\Sigma}_{s_k,0}]$ of this filter is the composition of the pose at the previous time step and the odometry estimate. Additionally, each observation's delay is compensated using the integrated odometry between the time of observation and the current time of the filter. Subsequently, the landmark observations are associated to already existing landmarks in the map using the nearest neighbour method. The Mahalanobis distance of an observation $\mathbf{z}_k$ to the Gaussian distribution of a landmark λ_n is used as a measure for the likelihood of correlation. If the maximum likelihood of an observation correlating to any landmark is below a threshold l, the observation is assumed to belong to a new landmark (de la Iglesia Valls et al., 2018).

Proposal Distribution and Map Update

After data association, the pose proposal EKF is iteratively refined by incorporation of the matched observations as measurements (Montemerlo and Thrun, 2007):

$$\mathbf{\Sigma}_{s_k,n} = (\mathbf{G}_{s,n}^T \mathbf{Z}_n^{-1} \mathbf{G}_{s,n} + \mathbf{\Sigma}_{s_k,n-1}^{-1})^{-1}, \tag{23}$$

$$\boldsymbol{\mu}_{s_k,n} = \boldsymbol{\mu}_{s_k,n-1} + \mathbf{\Sigma}_{s_k,n} \mathbf{G}_{s,n}^T \mathbf{Z}_n^{-1} (\mathbf{z}_{k,n} - \hat{\mathbf{z}}_n), \tag{24}$$

where

$$\mathbf{Z}_n = \mathbf{R} + \mathbf{G}_{\theta,n} \mathbf{\Sigma}_{s_k,n-1} \mathbf{G}_{\theta,n}^T, \tag{25}$$

and $\mathbf{R}$ is the measurement noise, and $\mathbf{G}_{\theta,n}$ and $\mathbf{G}_{s,n}$ are the Jacobians with respect to the landmark position and the particle pose respectively. A new pose is then sampled from the Gaussian distribution given by:

$$\mathbf{s}_k \sim \mathcal{N}(\boldsymbol{\mu}_{s_k}, \mathbf{\Sigma}_{s_k}). \tag{26}$$

Subsequently, the landmark positions are updated in a straight forward way using the EKF equations. Observations that were classified as "new landmarks" are initialized at the respective location. The landmark color is described by a categorical distribution with K = 3 possible outcomes. Bayesian inference on the color is applied using a Dirichlet distribution as conjugate prior. A discrete, counter based system updates the probabilities of each possible color for each landmark. This process happens separately for each sensor, and is tuned to suit the expected color accuracy of them.

Particle Weighting and Resampling

Each particle is weighted according to how well its map is in coherency with new landmark observations (Gosala et al., 2018)

$$w_k = w_{k-1} l^\nu w_b^\kappa w_c^\gamma \prod_{n \in \mathbf{a}_k} w_{k,n}, \tag{27}$$

where w_{k-1} is the particle weight at the previous time step, l the weight assigned to new landmarks and ν is the number of new landmarks, w_b a penalty for landmarks that are in sensor range but were not observed

and κ is the number of not observed landmarks, w_c penalizes a color mismatch for all γ landmarks whom color estimate don't agree with the observations and

$$w_{k,n} = \frac{1}{2\pi\sqrt{\det(\mathbf{L}_n)}} \exp\left(-\frac{1}{2}(\mathbf{z}_{k,n} - \hat{\mathbf{z}}_n)^T \mathbf{L}_n^{-1}(\mathbf{z}_{k,n} - \hat{\mathbf{z}}_n)\right), \tag{28}$$

are the importance weights of all matched landmarks where $\mathbf{L}_n$ is the special innovation covariance incorporating both measurement and motion models as well as the landmark uncertainty (Montemerlo and Thrun, 2007) and $\det(\cdot)$ is the determinant operator. The particle weight variance naturally increases over time and therefore resampling is enforced once the effective sample size N_{eff} of the particles drops below a certain threshold, $N_{\text{eff}} < 0.6N$.