Bachelor Project

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Control Engineering

Automatic vehicle following on the F1/10 platform

Jakub Hortenský

Supervisor: Ing. Jiří Vlasák
May 2021

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Hortenský Jakub**	Personal ID number:	**483471**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Control Engineering**		
Study program:	**Cybernetics and Robotics**		

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Automatic vehicle following on the F1/10 platform

Bachelor’s thesis title in Czech:

Automatické sledování vozidla na platformě F1/10

Guidelines:

Automatic vehicle following on the F1/10 platform
Práce bude vypracována v jazyce: ……………………české / anglické*……………………………………
Práce je zadána dle požadavku průmyslu, nebo je vedena někým z průmyslu (vedoucí, konzultant, oponent): ANO / NE *

Pokyny pro vypracování (uvádějte, prosím, ve výše uvedeném jazyce):
1. Get familiar with the communication framework ROS (Robot Operating System), the F1/10 project, and the KCF tracker.
2. Research the algorithms for trajectory reconstruction from the camera and LiDAR of the followed car (e.g., slalom.) Consider that the followed vehicle may disappear from view for a few seconds.
3. Combine the chosen solution with the algorithm for the trajectory following that can avoid the static obstacles detected by the LiDAR.
4. Test the implementation on the F1/10 platform and racetrack in our lab. Evaluate the error of the trajectory following. Discuss found problems.
5. Document and evaluate the work.

Bibliography / sources:

[1] Teck Chew Ng, J. I. Guzman and M. D. Adams, "Autonomous vehicle-following systems : a virtual trailer link model," 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, Edmonton, Alta., 2005, pp. 3057-3062, doi: 10.1109/IROS.2005.1545427.
[2] A. Muller, M. Manz, M. Himmelsbach and H. J. Wunsche, "A model-based object following system," 2009 IEEE Intelligent Vehicles Symposium, Xi'an, 2009, pp. 242-249, doi: 10.1109/IVS.2009.5164285.
[3] Chan Wei Hsu, Tsung Hua Hsu and Kuang Jen Chang, "Implementation of car-following system using LiDAR detection," 2012 12th International Conference on ITS Telecommunications, Taipei, 2012, pp. 165-169, doi: 10.1109/ITST.2012.6425157.

Name and workplace of bachelor’s thesis supervisor:

Ing. Jiří Vlasák, Department of Control Engineering, FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Ing. Michal Sojka, Ph.D., Embedded Systems, CIIRC

Date of bachelor’s thesis assignment: **26.01.2021** Deadline for bachelor thesis submission: **21.05.2021**

Assignment valid until:
by the end of summer semester 2021/2022

Ing. Jiří Vlasák	prof. Ing. Michael Šebek, DrSc.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Acknowledgements

I want to express my profound gratitude to my supervisor, Ing. Jiří Vlasák, for his valuable guidance, insight, advice, and encouragement that helped shape this thesis. Also, I want to thank Ing. Jaroslav Klapálek for his help with the F1/10 platform. Most importantly, I want to thank my parents, beloved ones, and friends for their endless support throughout not only my study but my whole life.

Declaration

Abstract

The main interest of this bachelor thesis is the automatic following of a vehicle on the F1/10 platform. Two means of target localization were described and implemented. Two variants of the algorithm (to generate trajectory) for vehicle following were implemented, tested, and compared. Both variants of the trajectory following algorithm were merged with an obstacle-avoiding algorithm. The result is a system for vehicle following capable of avoiding obstacles.

Keywords: F1/10 car model, ROS, car detection, car following

Supervisor: Ing. Jiří Vlasák

Abstrakt

Hlavním zájmem této bakalářské práce je automatické sledování vozidla na platformě F1/10. Byly popsány a implementovány dva způsoby lokalizace objektu. Dvě varianty algoritmu, který generuje trajektorii, byly implementovány, testovány a porovnány. Obě varianty byly spojeny s algoritmem pro vyhýbání překážek. Výsledkem je systém pro sledování vozidla schopný vyhýbat se překážkám.

Klíčová slova: F1/10 model auta, ROS, detekce auta, sledování auta

Překlad názvu: Automatické sledování vozidla na platformě F1/10

Contents

Figures

Tables

Chapter 1
Introduction

The automobile industry is slowly and surely evolving, replacing obsolete parts and methods with new ones. One of these obsolete parts may in the near future be the driver.

Automated driving currently gains increased popularity, powered by the dream of self-driving cars. Even though self-driving cars exist, the idea of each vehicle's automatization is still a few years away.

One of the primary abilities of an automated car is to follow another object without human interaction. The automatic following is the topic of this thesis.

In Chapter 2, we delve into the theoretical background of the automated following. The description of the used vehicle and its system is situated in Chapter 3. Chapter 4 proposes the implementation. We present and we evaluate the experiments on F1/10 platform in Chapter 5. We conclude this thesis in Chapter 6.

Chapter 2
Background

Following a vehicle without any human interaction is not an easy task. To follow a vehicle, the car needs to automatically perceive the environment, compute its trajectory with speeds and curvatures that are realistic and achievable, and follow the trajectory as accurately as possible.

In the following sections, we describe the theory we use to localize the followed vehicle, compute the correct trajectory, and ensure that our car stays on the computed trajectory.

2.1 Target localization

The first task to be accomplished is the *target* localization – the localization of the vehicle to follow. Commercial vehicles often use a combination of radar, LIDAR, and camera to perceive the surrounding area. This work will minimalize that by leaving out the radar component, similar to Kumar et al. [4].

2.1.1 KCF with LIDAR

To find the target on the camera footage, we need a tracker. The chosen tracker must be fast and computationally undemanding but at the same time accurate. A good combination of those traits is Kernelized Correlation Filter (KCF).

KCF is a tracker that compares received image with the image of the tracked object, and finds the most similar one. Its main benefits are the low computational requirements, the speed, and that it requires only one image of the tracked object. KCF package we use [11] returns the pixel position of the tracked object.

Before calculating the angle between the target and the car's axis, to receive a more accurate estimation of the angle, we need to calibrate the camera and undistort the image.

Distortion

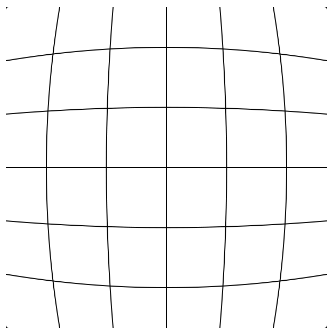
There are two main distortion types:

- **Radial distortion**. This distortion causes straight lines to appear curved; it becomes more prominent farther the point is from the center. The radial distortion can usually be classified as either barrel distortion or pincushion distortion, as shown in Figure 2.1. Radial distortion can be represented as follows:

$$x_{distorted} = x\left(1 + k_1r^2 + k_2r^4 + k_3r^6\right) \tag{2.1}$$

$$y_{distorted} = y\left(1 + k_1r^2 + k_2r^4 + k_3r^6\right) \tag{2.2}$$

  k_1, k_2, and k_3 are coefficients obtained from camera calibration, r is the distance of the point from the center, x is the undistorted x-coordinate, and y is the undistorted y-coordinate.


(a) : Barrel distortion visual example [12]

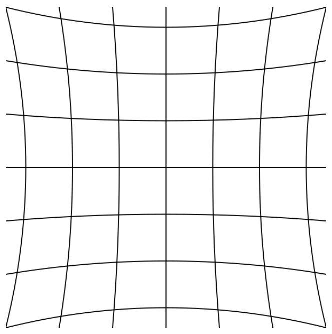

(b) : Pincushion distortion visual example [13]

Figure 2.1: Radial distortion types

- **Tangential distortion**. This distortion causes some points to appear closer than expected. Tangential distortion can be represented as follows:

$$x_{distorted} = x + \left[2p_1xy + p_2\left(r^2 + 2x^2\right)\right] \tag{2.3}$$

$$y_{distorted} = y + \left[2p_2xy + p_1\left(r^2 + 2y^2\right)\right] \tag{2.4}$$

  p_1 and p_2 are coefficients obtained from camera calibration, r is the distance of the point from the center, x is the undistorted x-coordinate, and y is the undistorted y-coordinate.

Camera calibration

To calibrate the camera, we require calibration tools and a calibrating object – a chessboard. Its straight edges and right angles are perfect for measuring distortions. The calibration is a process during which the chessboard is rotated, and the calibration tools measure the distances of chessboard fields and compute the camera's distortions.

The result of the camera calibration is its camera matrix K and its distortion coefficients k_1, k_2, k_3, p_1, and p_2.

Calculating the angle

With the camera calibrated, we now rectify the error caused by distortion. We use equations 2.1 – 2.4 to get the undistorted point. The angle α is then calculated using the formulas below.

$$u = K^{-1}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad v = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\alpha = \cos^{-1}\left(\frac{uv}{\|u\|\|v\|}\right) \tag{2.5}$$

Target's position

We use LIDAR (laser imaging, detection, and ranging) to compute the distance to the target. The principle of the LIDAR is that the LIDAR targets the object with a laser pulse, measures the time it takes the pulse to reflect and return back to the sensor, and computes the distance from the measurement.

Thanks to the angle α calculated from the camera, we can select only those rays that hold the distance d of the target. The relative coordinates of the target are calculated as follows:

$$\begin{bmatrix} x \\ y \end{bmatrix} = d\begin{bmatrix} \cos\alpha \\ \sin\alpha \end{bmatrix} \tag{2.6}$$

The combination of LiDAR and the camera creates a sensor fusion capable of reliably detecting the target and pinning his position on the map.

2.1.2 LIDAR only

The second approach we suggest in this work is excluding the camera and tracking the target purely with LIDAR. To achieve the level of precision and accuracy to localize a moving target with just LIDAR, an algorithm capable of finding an object in LIDAR data must be implemented. A viable algorithm to use is *obstacle detector package* [10].

The obstacle detection algorithm merges LIDAR points into obstacles by interpolating them with lines and circles. If this algorithm can interpolate these points with a line, it is taken as a wall and is ignored. Points interpolated with a circle are designated as obstacles, thus are candidates to be the target.

2.2 Next position calculation

The second task is to compute the advised position of the *ego car* – the following vehicle our algorithm runs on. The algorithm to succeed in this task must be simple, flexible, and computationally undemanding. Muller et al. [5] offer an insight into following the car using predictions and scouting the track ahead with tentacles. Wei et al. [3] deals with adaptive cruise control and keeping a safe distance in front of the ego car. Chew et al. [9] present the idea of the virtual link. The virtual link offers a lot of flexibility and reliability, as a concept of a link connecting a car and a trailer is widely used in cargo transportation. This thesis describes the algorithm for the next position calculation based on the virtual link theory.

For the purpose of the ego car, we can simplify and distribute the trailers into two types; the direct-hooked trailer and the off-hooked trailer. The former has only one solid rod. This approach is less difficult to understand and compute, as there is practically just one variable to work with. The latter approach is computationally more complicated but offers a smaller tracking error.

2.2.1 Direct-hooked trailer

The direct-hooked trailer features only one solid rod of length l_v, as in Figure 2.2. There is practically no angular computations as there is just one crucial angle α. The new position Y_1 can be calculated using equation 2.7, where X_1 is the position of the ego car.

$$Y_1 = X_1 - l_v \begin{bmatrix} \cos\alpha \\ \sin\alpha \end{bmatrix} \tag{2.7}$$

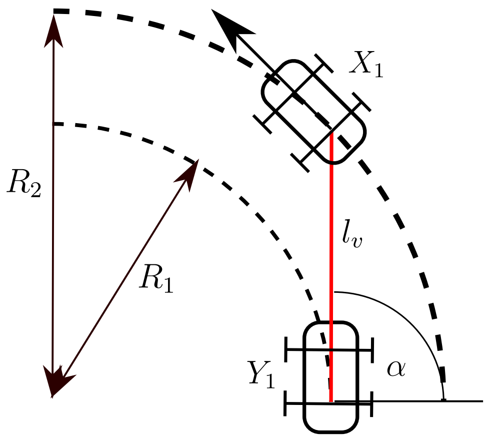


Figure 2.2: Direct-hooked trailer

As depicted in Figure 2.2, when the target drives along a circular trajectory with radius R_2, the ego car follows along a similar trajectory with radius R_1, where $R_1 < R_2$. The tracking error ϵ is equal to the difference between R_2 and R_1.

2.2.2 Extension to Direct-hooked trailer

To create a non-linear trajectory, we require more points than two. We improve the initial concept by adding additional points. We call these points *curvature points*, we can see curvature points c_{p1} and c_{p2} in Figure 2.3.

$$c_{p1} = Y_0 + \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \frac{\sqrt{X_1^2 - Y_0^2} - l_v}{r_1} \tag{2.8}$$

$$c_{p2} = Y_1 - \frac{Y_1 - c_{p1}}{r_2} \tag{2.9}$$

$$Y_1 = X_1 - \frac{l_v}{\sqrt{X_1^2 - c_{p1}^2}}(X_1 - c_{p1}) \tag{2.10}$$

Equation 2.8 gives us the first curvature point c_{p1}, where Y_0 and θ describe the ego car's position and orientation, respectively, X_1 is the target's position, l_v is the length of the link, and r_1 is a constant determining how far the c_{p1} lies from the target. The curvature point c_{p1} resembles the orientation of the ego car. Its purpose is to force the trajectory to begin in the same direction as the target car is facing.

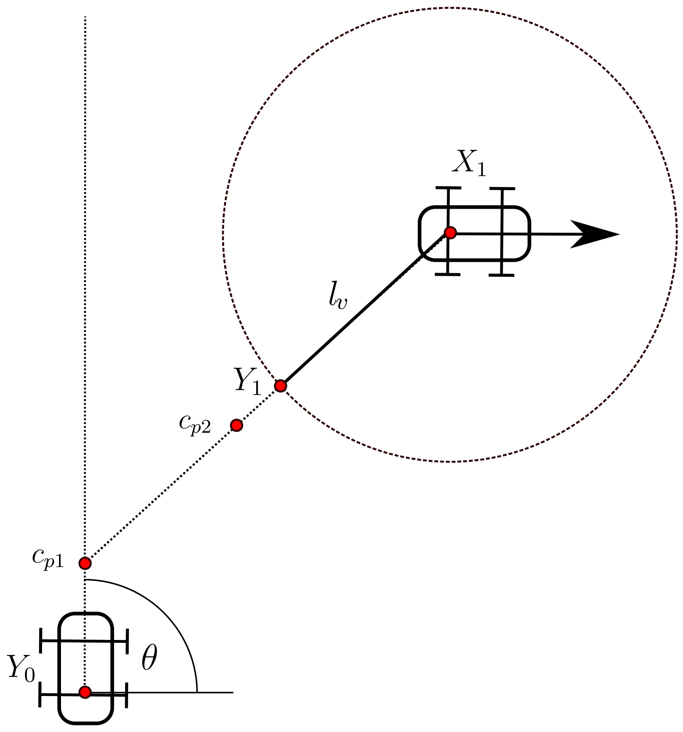


Figure 2.3: Direct-hooked trailer computed points

The second curvature point c_{p2} is given by Equation 2.9, where Y_1 is the advised position of the ego car, and r_2 is a constant determining how far the c_{p2} lies from the ego car's advised position. c_{p2} compels the ego car to arrive at the advised position from the correct direction.

Using Equation 2.10, we compute the ego car's advised position Y_1.

2.2.3 Off-hooked trailer

The off-hooked variant has two solid rods connected with a joint. This method offers more complex computations with smaller and less significant errors. In Figure 2.4 we can see that using this method, the ego car follows the target on the same radius R.

$$Y_1 = X_1 - l_{v1} \begin{bmatrix} \cos\alpha \\ \sin\alpha \end{bmatrix} - l_{v2} \begin{bmatrix} \cos\gamma \\ \sin\gamma \end{bmatrix} \tag{2.11}$$

The Equation 2.11 computes the advised position Y_1, where X_1 is the position of the target, l_{v1} is the length of the link connecting the ego car and the joint, l_{v2} is the length of the link connecting the joint and the target, α is the angle determining the target's orientation, and γ is the angle describing the ego car's orientation. Figure 2.11 visualizes the off-hooked trailer and the parameters in Equation 2.11. In Figure 2.11, Z resembles the position of the joint.

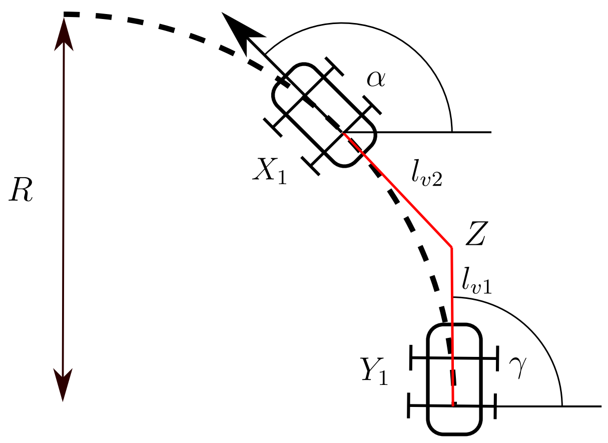


Figure 2.4: Off-hooked trailer

2.2.4 Extension to Off-hooked trailer

Similarly, as for the direct-hooked trailer variant, we add one or two curvature points; we can see curvature points c_{p1} and c_{p2} in Figure 2.5. This extension differs from the direct-hooked extension only in replacing the target's position X_1 with the joint position Z.

The position of the joint depends on the orientation of the target. If the target is in motion, the target's heading can be computed using its last and current position. The joint position can be calculated using Equation 2.12

$$\begin{aligned} d_l &= \frac{X_1 - X_0}{\|X_1 - X_0\|} \\ Z &= X_1 - l_{v2}\, d_l \end{aligned} \tag{2.12}$$

where $X_1 - X_0$ is the direction of travel of the target, X_1 the current position of the target, and l_{v2} is the length of the link connecting the joint and the target.

If the target is stationary and the orientation is unknown, we assume the orientation is the same as the vector from the ego car to the target. In other words, when the target is stationary, the joint is on the line connecting the ego car Y_0 and the target X_1. In these circumstances, the position of the joint Z is calculated using Equation 2.13

$$\begin{aligned} d_l &= \frac{X_1 - Y_0}{\|X_1 - Y_0\|} \\ Z &= X_1 - l_{v2}\, d_l \end{aligned} \tag{2.13}$$

where Y_0 is the position of the ego car, X_1 is the position of the target, and l_{v2} is the length of the link connecting the joint and the target.

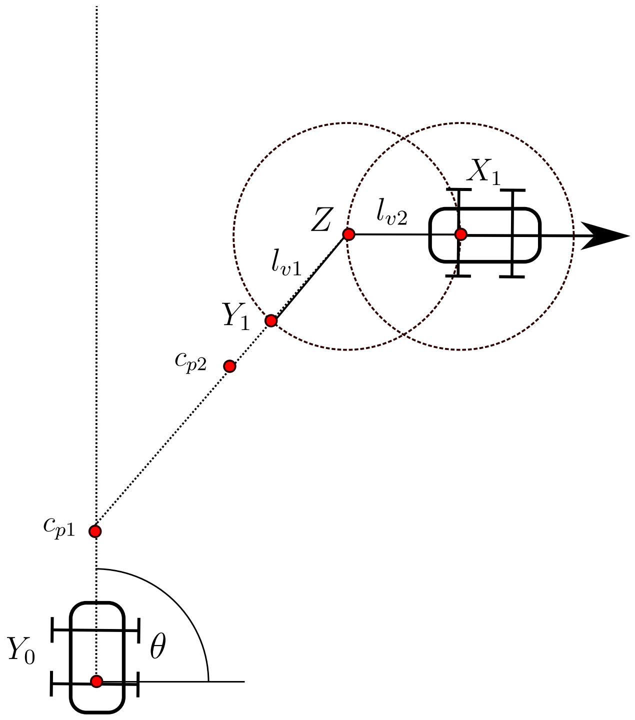


Figure 2.5: Off-hooked trailer new position

The curvature points and the advised position can be calculated using Equations 2.14 – 2.16

$$c_{p1} = Y_0 + \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \frac{\sqrt{Z^2 - Y_0^2} - l_{v1}}{r_1} \tag{2.14}$$

$$c_{p2} = Y_1 - \frac{Y_1 - c_{p1}}{r_2} \tag{2.15}$$

$$Y_1 = Z - \frac{l_{v1}}{\sqrt{Z^2 - c_{p1}^2}}(Z - c_{p1}) \tag{2.16}$$

where Y_0 is the ego car's position, θ its orientation, Z current position of the joint, l_{v1} link connecting the ego car to the joint, l_{v2} connecting joint and the target, and constants r_1 and r_2 determining how prominent and influential are c_{p1} and c_{p2}.

2.3 Trajectory creation

In this section, we describe the theory behind trajectory creation.

The *trajectory* is a set of consecutive points in time, where each point carries information about position, curvature, and velocity in the given position.

The motion of the ego car should be smooth to avoid wear and jerkiness. To achieve such motion, we require more points than those we compute in the virtual link algorithm. We interpolate these points with a Bézier curve and use it to calculate the trajectory points.

The input to an interpolating algorithm is Bezier curve defined by points P_0, ..., P_n, where the n is the order (e.g., $n = 1$ for linear, $n = 2$ for quadratic, ...). P_0 is the starting point and P_n is the ending point. The rest of the points determine the curvatures. The parameter t ranges from 0 to 1, determining which point on the curve we want to calculate.

Bézier curves can be defined for any degree n. The virtual link algorithms described in Section 2 and Section 3 generate three or four points. Therefore, we will limit ourselves to the quadratic Bézier curve and the cubic Bézier curve. We can map mathematical points P_0 to P_n to their values assigned in Section 2 and Section 3. P_0 is the ego car position Y_1, P_1 to P_{n-1} are the curvature points c_{pn}, in our case c_{p1} and c_{p2}. P_n is then the advised position of the ego car.

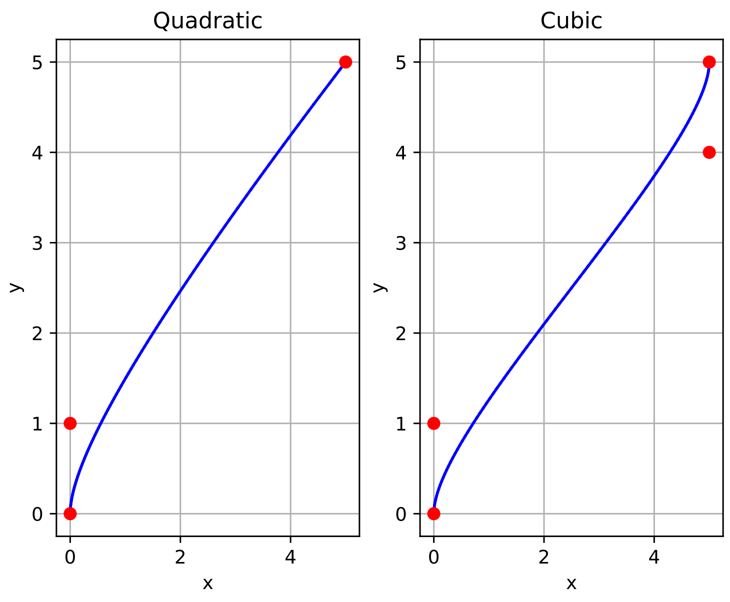


Figure 2.6: Comparison between quadratic and cubic Bézier curve

2.3.1 Quadratic Bézier curve

The quadratic Bézier curve is made of three points, i.e. n=2, where P_1 defines the initial orientation of the ego car. Any point on the curve can be calculated using Equation 2.17.

$$B(t) = (1-t)^2 P_0 + 2(1-t)tP_1 + t^2 P_2 \tag{2.17}$$

To calculate a curvature in a point on a curve, we need its first and second derivatives with respect to t.

$$B'(t) = 2(1-t)(P_1 - P_0) + 2t(P_2 - P_1) \tag{2.18}$$

$$B''(t) = 2(P_2 - 2P_1 + P_0) \tag{2.19}$$

2.3.2 Cubic Bézier curve

The cubic Bézier curve is made of four points, i.e. n=3. In cubic variant, P_2 defines the end orientation of the ego car. Any point on the curve can be calculated using Equation 2.20.

$$B(t) = (1-t)^3 P_0 + 3(1-t)^2 tP_1 + 3(1-t)t^2 P_2 + t^3 P_3 \tag{2.20}$$

To calculate a curvature in a point on a curve, we need its first and second derivatives with respect to t.

$$B'(t) = 3(1-t)^2(P_1 - P_0) + 6(1-t)t(P_2 - P_1) + 3t^2(P_3 - P_2) \tag{2.21}$$

$$B''(t) = 6(1-t)(P_2 - 2P_1 + P_0) + 6t(P_3 - 2P_2 + P_1) \tag{2.22}$$

2.3.3 Curvatures

Each point of the trajectory must contain the trajectory's curvature. The curvature K of any trajectory is equal to the inverse value of the radius of the circular arc, which best approximates the curve at that point. This radius R can be computed using first and second derivatives. We derivate the Bezier curve $B(t)$, which means we derivate with respect to the parameter t. For simplification, $x = x(t)$ and $y = y(t)$.

$$R = \frac{\left[(x')^2 + (y')^2\right]^{\frac{3}{2}}}{|x'y'' - y'x''|} \tag{2.23}$$

$$K = \frac{1}{R} \tag{2.24}$$

2.3.4 Velocities

To calculate velocities, we propose a model that operates in two modes; the cruise mode and the normal mode.

The cruise mode activates when the target is stationary. Its purpose is to drive to the target with maximal precision at low speeds. This speed is set at one meter per second.

The normal mode is activated when the target is moving. In determining whether we need to increase or decrease the speed, we calculate the current distance d_c between the current ego car's position and the advised ego car's position and compare it with its previous value d_l. At the end of the procedure, d_l becomes d_c.

$$d_c = \|Y0 - Y1\| \tag{2.25}$$

Y_0 is the current position of the ego car and Y_1 its next advised position, v is the ego car's current velocity.

- First case is that $d_c = 0$. The velocity of the ego car is set to zero, and the ego car stops.
- If $d_c > d_l$, the ego car increases its velocity by a parameter v_{delta}. This parameter is set for testing at $v_{delta} = 0.1$.
- If $d_c < d_l$, we need to examine whether the ego car is close to the target or not. If the condition in Equation 2.26 is true, then we decrease the speed by the parameter of v_{brake}, computed in Equation 2.27. Otherwise, we speed up by a parameter of v_{delta} to catch up to the target.

$$d_c \leq 0.1v^2 \tag{2.26}$$

$$v_{brake} = 3v^2 v_{delta} \tag{2.27}$$

- In the last case, the distance is the same, and the ego car keeps its velocity.

2.4 Trajectory following

To follow the created trajectory, we use the *pure pursuit* algorithm [2]. The pure pursuit algorithm creates an imaginary point in front of the ego car, which it follows. This trajectory tracking algorithm offers sufficient tracking capabilities while also requiring only basic information about the track – the position, the curvature, and the velocity.

To avoid obstacles, we switch to *follow the gap* algorithm [8]. As the name suggests, follow the gap searches for the largest gap in front of the ego car.

Chapter 3
Platform F1/10

The theory described in Chapter 2 is designed to be implemented on the F1/10 platform. The F1/10 vehicles are RC-based models that are in a 1:10 ratio with ordinary cars. It is safer and cheaper to develop and test new algorithms and control methods on this platform instead of on regular cars.

In this chapter, we describe the hardware and software of the car we use in this thesis.

3.1 F1/10 competition

Having been built for the F1/10 competition [6], in order to understand why was the car built this way, it might be usefull to glance over the F1/10 competition first.

F1/10 competition is a bit more than just a competition. It includes education, research, develepoment, and of course the race itself. The goal is to create an autonomous vehicle that is safe and push it to its racing limits.

Safety. There are two main rules established to ensure the safety of all the competitors and the car itself. First, the car has to have an emergency disconnect switch, that would remove all the power from the engine. Effectivelly this is a big red stop button. Second, the car must possess the ability to switch from the autonomous mode to the manual mode at any time.

Communication. As a theme of the whole competition suggests, the vehicle has to be fully autonomous and self-contained. This means no transmitters nor any communication is allowed. The only exception is WiFi, but this too cannot be used for controling the vehicle. For navigation, the car may effectively use only components that are part of the vehicle. Moreover, cooperation amongst different vehicles or track modification is strictly prohibited.

Physical parameters. Before the competition, vehicles are subjected for a series of tests, that will determine, whether the vehicle is acknowledged as a race worthy, and be granted a race license plate (RCL). First, the car must fit inside a box, which parameters are announced before the competition. This also applies for the vehicle's weight. Second, the vehicle is prohibited to include any parts, which sole purpose is to inflict damage or deceive another car.

Sensors. Each team has to choose a sensor configuration using just the following: at most 2 cameras, 1 LIDAR, and 1 IMU. Having Wifi is obligatory.

Computation. NVIDIA Jetson TK1 or anything of a lower spec must be used for planning and perception. A team can choose between MBed or Teensy for the car control.

Chassis. The car must be a Traxxas Rally 1/10 with any suspesion.

3.2 Hardware

The car that is used for the purpose of testing the ability to follow another vehicle is a third-generation car developed by the team at CIIRC, each generation slightly better than the previouse one. Hardware list of this vehicle can be seen in Table 3.1.

Vehicle Part	Part Model
CPU	nVidia Jetson TX2 on Orbitty Carrier
Chassis	Traxxas Slash 1:10 4WD VXL TQi TSM OBA RTR
Engine	Velineon 3500
Engine CPU	Teensy 3.1
Controller	VESC
Transceiver	B3-STX Delluxe 2.4GHz F.H.S.S.
Batteries	LiPo 3S, LiPo 2S
LiDAR	Hokuyo UST-10LX
Camera	AUSDOM AF640

Table 3.1: Hardware list

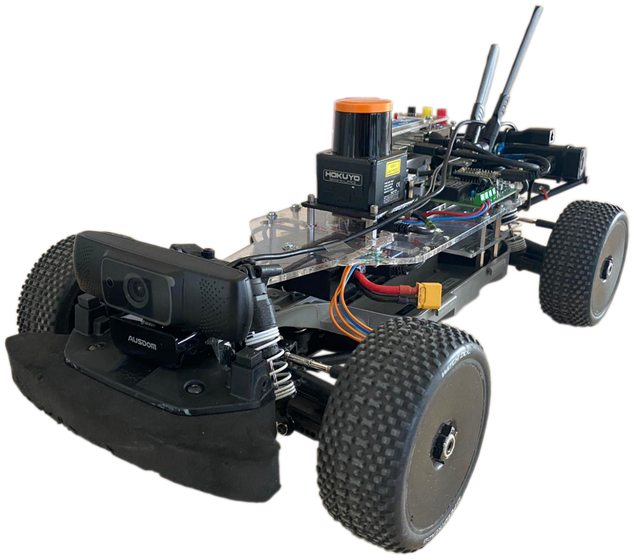


Figure 3.1: Ego car

Requirements for the F1/10 competition (Subsection 3.1) clearly restrict the chassis' selection; however, after some debate with the organizers, the exception to use the model mentioned in Table 3.1 has been made. Concerning batteries, LiPo 2S is used to power the engine, LiPo 3S to power the CPU.

3.2.1 Engine

We use brushless motor that is highly optimized for the F1/10 cars. It uses neodymium magnets to have a massive torque. The motor can achieve up to 50 000 RPM. Because of the absence of any brushes or a commutator, this particular motor's maintenance is effortless. Control over the brushless motor is established by an open-source motor controller VESC. VESC uses pulse-width modification (PWM), received from Teensy, to control the motor.

3.2.2 LiDAR

One of the main sensors used in algorithms for navigation is LiDAR. The choice of the model (Hokuyo UST-10LX) complies with the rules (Subsection 3.1). The LIDAR works on the principle of firing laser pulse and receiving the reflected light. From this reflected light, it measures the distance of the object from the sensor. This particular LiDAR works on a 40 Hz frequency, which fulfills its job without any complications. Even though LiDARs used in modern civilian cars can often scan a 3D space, LIDAR used on described platform limits its view to a 2D plane, which is an allowed simplification for the F1/10 cars. Its 270 degrees field of view with 0.25 degrees angular resolution guarantees that all objects in front of the vehicle up to a distance of 10 meters will be spotted. The LIDAR communicates with the car via the Ethernet interface.

Figure 3.2: Hokuyo UST-10LX

3.2.3 Camera

The camera ranks amongst the essential sensors for environment perception. The camera model available for the car's sensor composition is AUSDOM AF640. This particular full HD camera offers a decent view range thanks to its 90 degrees diagonal view angle.

Figure 3.3: Ausdom AF640

3.3 Software

The hardware setup is only half of the car. To control it, we need optimally working software. The most critical piece of software is a communication system, which enables the communication between sensors and various algorithms. A well-known communication framework in robotics is ROS [7].

3.3.1 ROS

ROS is a communication system for a robot. Its idea is to enable communication between different packages containing robot's algorithms.

Each running program is called a node. ROS has its own master node labeled ROSCore. Nodes communicate with each other by subscribing to a topic – a data flow of messages, where a message is a piece of data formatted in a specific way. Nodes can either subscribe to a topic (receive messages) or to publish a topic (send messages).

For user readability and better code distribution, groups of nodes are contained in packages. Each package has its own name and list of dependencies. Nodes can be written in C++ or Python.

In order to ensure the proper functionality of F1/10 simulators, the distribution used in our car is ROS Kinetic.

RVIZ

RVIZ is a 3D visualization tool for ROS applications. It can visualize the robot's position, its sensor data, or the result of its computations. We use this software component to visualize the racetrack, the target, and the ego car.

3.3.2 Car's Operating System

ROS currently operates only on Linux-based platforms, primarily tested on Ubuntu and Mac OS. We use Ubuntu (free and open-source operating system) as the operating system for the car. The advantage is the vast selection of Ubuntu-supported packages. Using ROS Kinetic, Ubuntu 16.04 is the Ubuntu version running on the car.

3.3.3 Architecture

A simple architecture of ROS packages was established based on Behere et al. [1]. The main idea is to split the whole environment into three modules: Perception, Decision and Control, and Vehicle Platform.

Perception

The first layer is called Perception. The main goal is to understand the environment. This module takes raw sensor data as an input and returns a structured information about the environment (e.g. position on the map), which is then used in another module. Perception has been split into three blocks.

Sensors. The first block is responsible for gathering data from the sensors, and converting them into ROS messages. Each sensor type has its own ROS package.

Preprocessing. The second block is used to adjust or slightly alter the data coming from the first block. Its main goal is to precompute or clean input data, to make the following computations simpler.

Recognition. The final block takes raw or preprocessed data, and creates structured objects that can be used in the Decision and Control module.

Decision and Control

The second module evaluates the input data from the Perception module and, through various algorithms, computes the best decision for the car to make. This decision is then pushed into the Vehicle Platform module.

Vehicle Platform

The Vehicle Platform module is responsible for the low-level control of the car. As stated in Subsection 3.2.1, both-the motor and the servo-are controlled via PWM signals. The parameters of the PWM signals may differ in different vehicles, so the Vehicle Platform module is designed to solve this issue of hardware dependancy. The emergency stop demanded by the rules is implemented in this module.

Chapter 4

Implementation

In this section, we discuss the implementation. We glance over programming languages. Then we move on to the target localization and the trajectory planning. We finish this chapter by discussing the implementation of avoiding obstacles while following the target.

4.1 Programming language

ROS currently supports only Python and C++. The main advantage of Python is its versatility and readability. The weakness of Python compared to C++ is its speed and performance. Therefore, in parts where speed is essential, C++ is preferred.

4.2 Target localization

The first task is to localize the target, as described in Section 2.1. According to the architecture established in Section 3.3.3, target localization is part of the Perception module.

4.2.1 KCF with LIDAR

The target's position is obtained by the camera and LIDAR. The idea is to find the target on a camera image using the KCF, calculate the angle between the ego car's axis and the target, and finally measure the distance using LIDAR. By the approach described above, we obtain the relative position of the target to the ego car.

Target's position in the camera image

First, we need to determine where on the camera footage the target is. This is done using KCF mentioned in Section 2.1. We updated the implementation from [11] to make the output compatible with ROS. The tracker is written in C++ to ensure its high computation speed. The return value is the ROS message of type *Point* which consists of x and y coordinates in pixels.

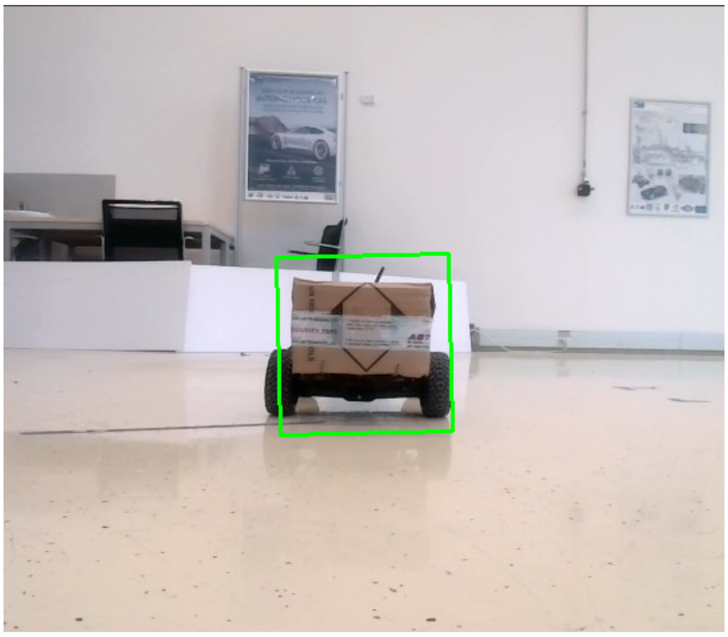


Figure 4.1: Localization using KCF

Camera calibration

For camera calibration, we use OpenCV tools. OpenCV requires us to input the camera image, chessboard size, and the length of one chessboard field. When measuring the size of the chessboard, what we are looking for is the number of intersections. Our chessboard size is 8x6, and the length of one field is 28.6 mm.

The result of the camera calibration is its camera matrix K and its distortion coefficients k_1,k_2,k_3,p_1, and p_2. For our case, those parameters are as follows:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 510.752 & 0 & 296.879 \\ 0 & 512.582 & 233.267 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

$$\begin{aligned} k_1 &= 0.022255 \\ k_2 &= -0.173634 \\ k_3 &= -0.012291 \\ p_1 &= -0.000857 \\ p_2 &= 0.009136 \end{aligned} \tag{4.2}$$

Calculating the angle

With the camera calibrated and the target position in the image undistorted, we compute the angle between the ego car's axis and the target's position. Angle between the target and the ego car, computed by Equation 2.5, is not oriented; we know only its absolute value. To overcome this issue and determine the orientation of the angle, we use the pixel's x coordinate. If the pixel is on the right side of the image, the angle is negative; otherwise, it is positive.

Target's position

We use LIDAR to get the distance from ego car to target. We use the computed angle between the target and ego car to select only the LIDAR rays that points to the target. To decrease the influence of noise and sensor inaccuracy, we found empirically that the average value of five rays suits well relatively to target size and average distance.

To project these coordinates to the real world, we need to know the ego car's position and orientation computed in Section 4.2.3.

4.2.2 LIDAR only

We use the obstacle detector ROS package [10]. This algorithm groups points found by LIDAR into lines or circles. To find the circle that is most likely the target, we filter out circles with a speed of less than 0.2 m/s and order these moving circles by their ascending distance from the target's last known position. If we found no moving circles, we order them by their angle between the circle and the target's last known position instead. If the distance between the first circle and the target's last known position is less than 60 cm and the angle between these positions is less than 180 degrees, the circle is proclaimed as the target.

If the distance is greater than 60 cm or the angle is greater than 180 degrees, the ego car slowly drives into the last position of the target and stops. During its movement, the ego car scans the area to detect new circles to re-find the target.

4.2.3 Ego car's position and orientation

The ego car's current position and orientation are provided by odometry. Odometry returns a pose that contains a position in Euclidean space and orientation given by quaternion q. We can compute Yaw angle from the quaternion by the Equation 4.3.

$$\theta = \operatorname{atan2}(2q_xq_y + 2q_wq_z, q_w^2 + q_x^2 - q_z^2 - q_y^2) \tag{4.3}$$

4.3 Next position calculation

The computation of the next position is a part of the Decision and Control module.

To calculate the next position of the ego car, we use the algorithm described in Section 2.2. Our goal is to implement the direct-hooked (Section 2.2.2) and the off-hooked (Section 2.2.4) virtual link to compare their tracking errors and efficiency.

4.3.1 Preparation and simplification

We assume that the vector going from the last position of the target to its current position has the orientation as the target's heading.

4.3.2 Direct-hooked trailer

The direct-hooked virtual link connects the target and the ego car directly. The next advised position lies on the circle with a center at the target's position X_1 and of a radius l_v and on the line between the X_1 and the first curvature point c_{p1}, as shown in Figure 2.3. To ensure the ego car does not lose track of the target and does not hit it while braking, the length of the link is set to 0.75 m. The process of computing the advised ego car position and curvature points using the direct-hooked algorithm can be seen in Algorithm 1.

Algorithm 1 Direct-hooked trailer – compute advised position

```
1: procedure DIRECT-HOOKED POINTS COMPUTATION($Y_0$,$X_1$,$l_v$,$\theta$,$r_1$,$r_2$)
2:     if distance($X_1$, $Y_0$) $\leq l_v$ then
3:         Stop the ego car
4:     end if
5:     $c_{p1} \leftarrow$ First curvature point given by Equation 2.8
6:     if distance ($X_1$, $c_{p1}$) $\leq l_v$ then
7:         Stop the ego car
8:     end if
9:     $Y_1 \leftarrow$ Advised position given by Equation 2.10
10:    $c_{p2} \leftarrow$ Second curvature point given by Equation 2.9
11:    return $c_{p1}$, $c_{p2}$, $Y_1$
12: end procedure
```

The values of r_1 and r_2 influence the curvature of the trajectory. If the values are too low, the ego car never turns. If the values are too high, the curvature points lose their purpose. To ensure that the ego car turns reliably, we choose $r_1 = 5$ and $r_2 = 5$.

4.3.3 Off-hooked trailer

The off-hooked virtual link connects the target and the ego car through a joint. The next advised position Y_1 lies on the line from the first curvature point c_{p1} to the joint position Z. The distance between Z and Y_1 is l_{v1}.

In calculating the curvature points and the advised position, we implement Equations 2.14 – 2.16 and merge them with conditions ensuring the car stops when necessary. The process of computing the advised ego car position and curvature points using the off-hooked algorithm can be seen in Algorithm 2.

Algorithm 2 Off-hooked trailer – compute advised position

```
 1: procedure OFF-HOOKED POINTS COMPUTATION($Z$,$Y_0$,$X_1$,$l_{v1}$,$\theta$,$r_1$,$r_2$)
 2:     if distance($Z$, $Y_0$) $\leq l_{v1}$ then
 3:         Stop the ego car
 4:     end if
 5:     $c_{p1} \leftarrow$ First curvature point given by Equation 2.14
 6:     if distance ($Z$, $c_{p1}$) $\leq l_{v1}$ then
 7:         Stop the ego car
 8:     end if
 9:     $Y_1 \leftarrow$ Advised position given by Equation 2.16
10:     $c_{p2} \leftarrow$ Second curvature point given by Equation 2.15
11:     return $c_{p1}$, $c_{p2}$, $Y_1$
12: end procedure
```

For minimal tracking error, the links should be of equal length, as proposed by Chew et al. [9]. If the link is too long, the ego car may easily lose the target. On the other hand, if the link is too short, there is a high possibility of a collision. We choose each link to have half a meter, or rather $l_{v1} = 0.5\,\text{m}$ and $l_{v2} = 0.5\,\text{m}$.

We choose $r_1 = 5$ and $r_2 = 5$, although, in this variation, we feel comfortable submitting a lower value.

4.4 Trajectory creation

The result of the virtual link algorithm is three or four points. If we choose three points, the maximum order of interpolation curve is two (quadratic curve). Four points offer the maximum order of three (cubic curve).

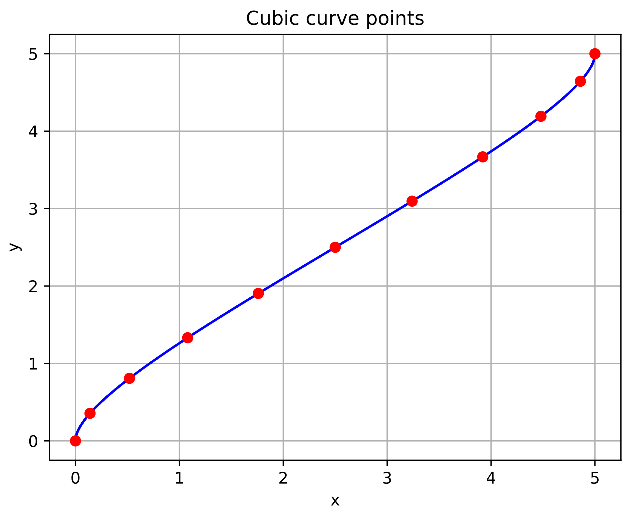


Figure 4.2: Points on cubic Bézier curve

For implementation, we decided to use the cubic curve. It offers an arrival with a smaller heading difference.

The number of points received from interpolation is nearly infinite, but as we work with short trajectories, we empirically found that eleven points in total suffice, as is shown in Figure 4.2.

4.5 Obstacle avoidance

After computing the points of the trajectory, the points need to be examined, whether there is no obstacle in the way. Using the LIDAR, we check the surroundings for obstacles. To be as efficient as possible, we scan the area of 180 degrees in front of the ego car, and we consider only every fifth ray. We calculate the distance between each of the trajectory points and the point found by LIDAR as an obstacle. If there is a distance that is less than 20 cm (the padding we chose according to the size of the car), the Follow the gap algorithm is activated. Once there are no distances less than 20 cm, the Follow the gap algorithm is deactivated.

Chapter 5
Experiments

This chapter describes the experiments conducted with the implemented algorithms, and visualizes and analyzes the results.

5.1 Target localization

We have implemented both the camera with LIDAR variant and the LIDAR only variant to accomplish the target localization. Testing showed that we can't use the camera, as it creates a delay of 2 to 4 seconds. This amount of delay makes the ego car uncontrollable. Therefore, we use LIDAR only for target localization in the following tests.

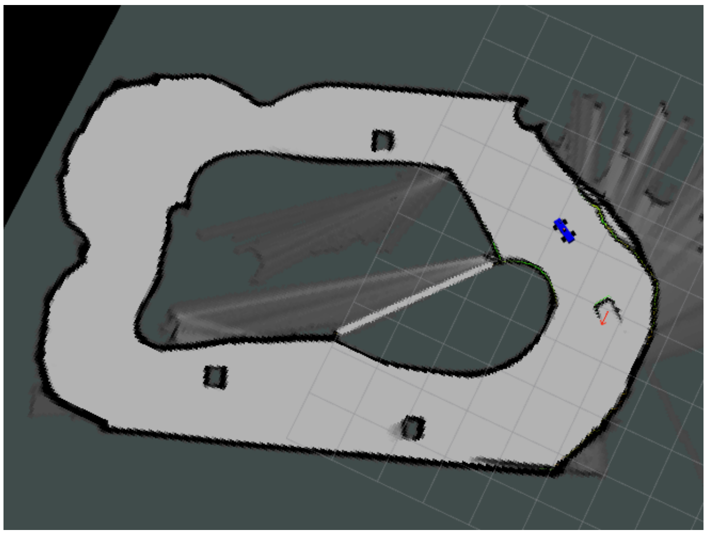


Figure 5.1: The detection of the target using LIDAR is visualized using RVIZ on the map of the racetrack. The ego car's position and orientation are represented by the blue car, the red arrow represents the target's position.

Figure 5.1 represents the localization of the target on our racetrack using LIDAR only. During testing, the target is frequently lost. We solve this problem by re-finding the target using the approach described in Section 4.2.2.

5.2 Target following

We have prepared two scenarios in which we showcase and compare both algorithms. The first scenario is the following of the target on a straight line. We expect minimal errors from this experiment, as there are no turns to lose the car in or to develop a deviation from the target's trajectory.

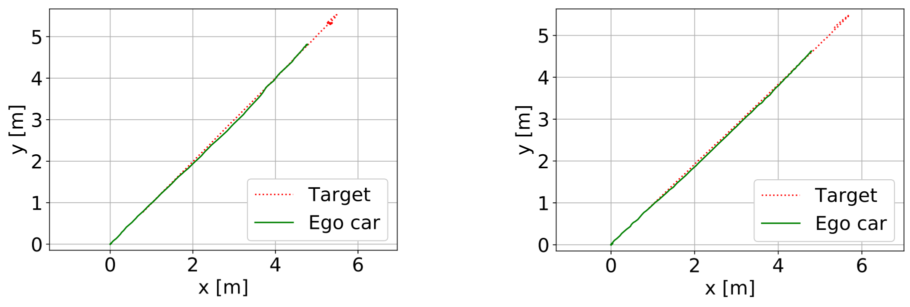
The second scenario is the one lap of our lab's racetrack. The vehicles are driving in a clockwise direction. In this experiment, we anticipate more severe deviations from the target's trajectory; eventual obstacles are handled by the follow the gap algorithm (Section 4.5).

In the following sections, we compare the direct-hooked and off-hooked algorithms in their abilities to follow a target. We first present the trajectories the ego car and the target followed. Next, we compare the velocities of the target and the ego car. Then we discuss the distances measured between the ego-car and the target. Finally, we present the tracking errors of both algorithms.

5.2.1 Experiments in a straight line

We begin our experiments by driving in a straight line. The racetrack for this experiment is a straight line 7 m long. The starting position of the target is 0.6 m ahead of the ego car.

At first, we log the positions of the target and the ego car on the map and recreate the trajectories of both vehicles.

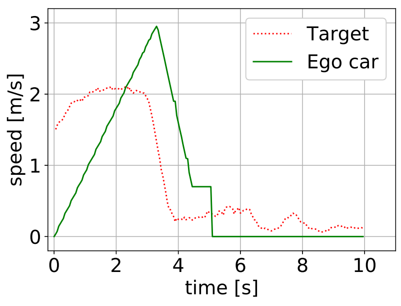


(a) : Direct-hooked trailer **(b) :** Off-hooked trailer

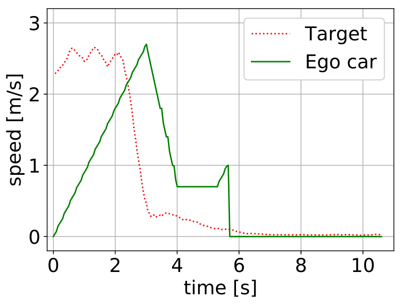
Figure 5.2: Positions of the target (red dotted line) and the ego car (green line) in a straight line.

In Figure 5.2, we can see the comaparison of positions in a straight line driving. The ego car's start is at the *origin* – the intersection of the x-axis and the y-axis. The localization error rises when the target suddenly stops; this is visible at the end of the target's line of movement in both figures.

The next important aspect of the vehicle following is the ego car's and target's velocity. We have implemented a straightforward incrementing model to ensure the ego car catches up with the target in the shortest possible time, so we expect a noticeable difference between the velocities.



(a) : Direct-hooked trailer

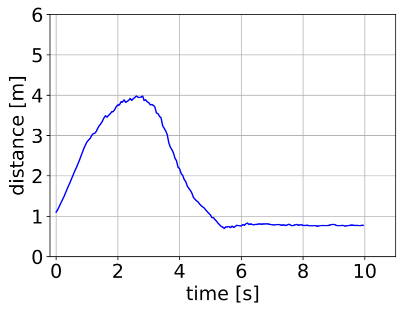


(b) : Off-hooked trailer

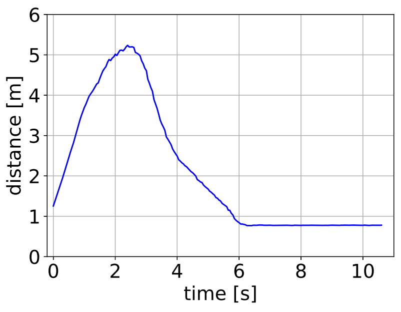
Figure 5.3: The estimated speed of the target (red dotted line) and the actual speed of the ego car (green line) in a straight line.

Figure 5.3 depicts the comparison of velocities of the off-hooked and the direct-hooked variant in a straight line driving. We presume the target's speed is based on its position. This means the inaccuracy of target localization influences the values of the target's speed. In the direct-hooked variant, the target reaches lower speeds than in the off-hooked variant. Therefore, the target takes longer to reach the 7-meter mark. Consequently, the ego car accelerates to higher speeds, and it takes less time to reach the target.

In Figure 5.4, we can see the measurement of the distance between the target and the ego car. In the straight line, the distance should be 0.75 meters for the direct-hooked trailer and 1 meter for the off-hooked trailer, as those are the values assigned to the links.



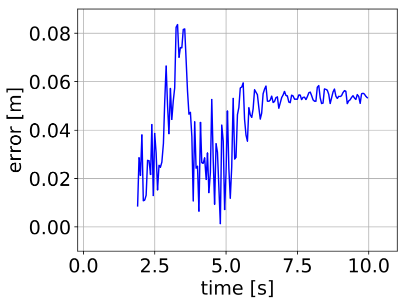
(a) : Direct-hooked trailer



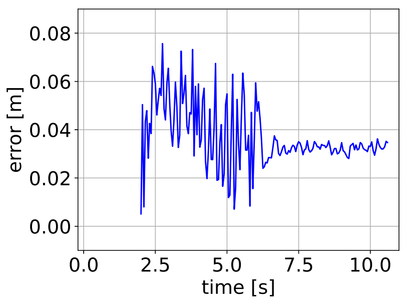
(b) : Off-hooked trailer

Figure 5.4: Distance between the target and the ego car during driving in a straight line.

In Figure 5.4, we can see the distance the off-hooked and the direct-hooked variant keep between the target and the ego car. The zero on the x-axis in Figure 5.4 indicates the moment when the ego car receives the first impulse to move. The distance corresponds to the conclusion we made for the velocities. Finally, in Figure 5.5 we can see the tracking error.



(a) : Direct-hooked trailer



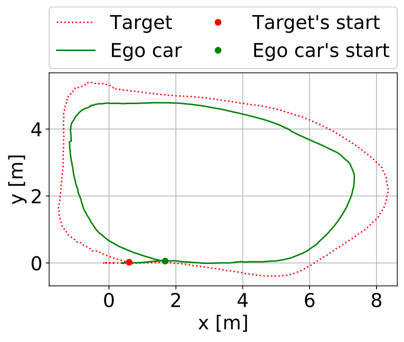
(b) : Off-hooked trailer

Figure 5.5: Tracking errors during driving in a straight line.

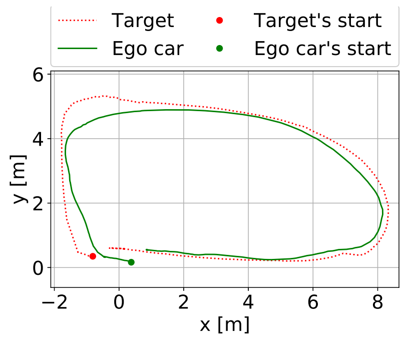
The starting point of the graphs in Figure 5.5 is when the ego car reaches the target's starting position. The tracking error is influenced by the odometry error and the target localization error. Both variants end up similarly, as is visible in the positions shown in Figure 5.2. The tracking error does not exceed 8 centimeters, which is acceptable concerning all the added errors.

5.2.2 Experiments on a racetrack

In this experiment, we expect the variants to behave differently; in particular, we expect the off-hooked variant to have smaller tracking errors.



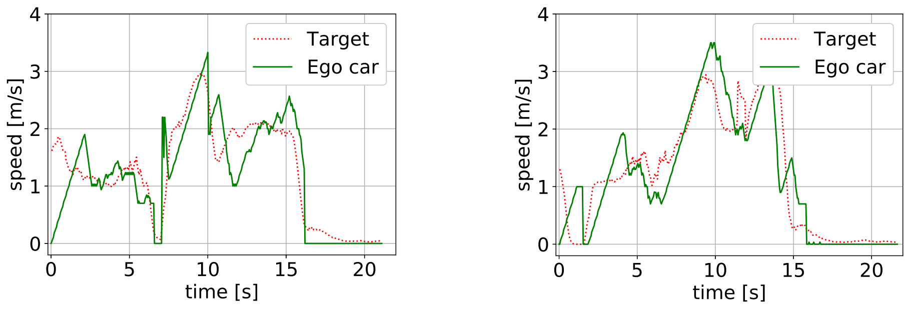
(a) : Direct-hooked trailer



(b) : Off-hooked trailer

Figure 5.6: Positions of the target (red dotted line) and the ego car (green line) on a racetrack.

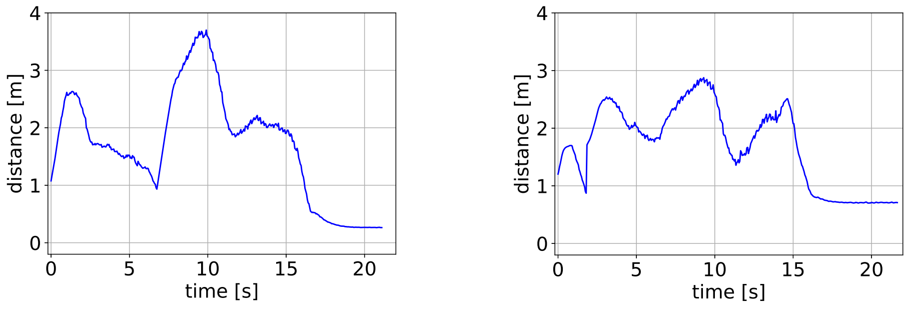
In Figure 5.6 we can see that the off-hooked variant follows the target more tightly than the other variant.



(a) : Direct-hooked trailer **(b) :** Off-hooked trailer

Figure 5.7: The estimated speed of the target (red dotted line) and the actual speed of the ego car (green line) on a racetrack.

In Figure 5.7, we can see the velocities of the target and the ego car. The ego car's speed successfully tracks the target's speed. Spikes of the ego car's velocity curve that seem out of place are caused by switching on follow the gap algorithm. This irregularity can be seen, for instance, in Figure 5.7 (a) in the eleventh second.

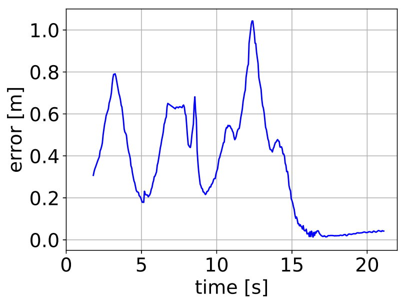


(a) : Direct-hooked trailer **(b) :** Off-hooked trailer

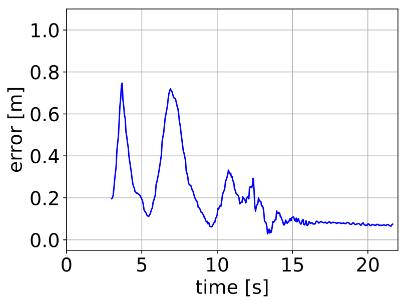
Figure 5.8: Distance between the target and the ego car during driving on a racetrack.

Figure 5.8 depicts that the distance between the target and the ego car is not constant. The distance increases when the ego car needs to adjust its speed quickly. The ego car is then able to decrease the distance with time.

Recall the racetrack in Figure 5.6, in Figure 5.9, we can see that the direct-hooked variant has a significant tracking error for the whole duration of the lap. The off-hooked variant cuts steep corners, but the tracking error is significantly lower in less complicated turns.



(a) : Direct-hooked trailer



(b) : Off-hooked trailer

Figure 5.9: Tracking errors during driving on a racetrack.

Chapter 6
Conclusion

In this thesis, we described the implementation of two localization algorithms and two trajectory planning algorithms. We merged these algorithms with the algorithm capable of avoiding static obstacles. We tested these implementations on the F1/10 platform.

The target localization using the camera created uncontrollable delays. Therefore, we used the LIDAR only method, which is capable of reliably localizing the target and re-finding it if lost.

The both trajectory planning algorithms perform well, but the tracking error of the off-hooked variant is smaller by approximately 30% compared to direct-hooked variant. In experiments on racetrack, cutting corners by the direct-hooked algorithm is compensated by the obstacle avoidance algorithm.

We discovered two areas where discussed improvements would be a major upgrade to this work; the target localization and the velocity planner. The obstacle detector package is made for tracking cones on the road but not for tracking moving vehicles. Replacing the obstacle detector package with a specialized algorithm would significantly improve the target localization.

To improve velocity planner, a more advanced and elaborate velocity model would ensure the car accelerates faster and keeps the distance constant with minimal deviancy. This would help the ego car to keep the distance between itself and the target constant.

Appendix A

Bibliography

[1] Sagar Behere and Martin Torngren. A functional architecture for autonomous driving. In *2015 First International Workshop on Automotive Software Architecture (WASA)*, pages 3–10, 2015.

[2] R. C. Coulter. Implementation of the pure pursuit path tracking algorithm. Technical Report ADA255524, Defense Technical Information Center, January 1992.

[3] Chan Wei Hsu, Tsung Hua Hsu, and Kuang Jen Chang. Implementation of car-following system using lidar detection. In *2012 12th International Conference on ITS Telecommunications*, pages 165–169, 2012.

[4] G Ajay Kumar, Jin Hee Lee, Jongrak Hwang, Jaehyeong Park, Sung Hoon Youn, and Soon Kwon. Lidar and camera fusion approach for object distance estimation in self-driving vehicles. *Symmetry*, 12(2), 2020.

[5] A. Muller, M. Manz, M. Himmelsbach, and H. J. Wunsche. A model-based object following system. In *2009 IEEE Intelligent Vehicles Symposium*, pages 242–249, 2009.

[6] F1/10 organizers. The rules. `https://f1tenth.org/misc-docs/rules.pdf`.

[7] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[8] Volkan Sezer and Metin Gokasan. A novel obstacle avoidance algorithm: "follow the gap method". *Robotics and Autonomous Systems*, 60(9):1123–1134, 2012.

[9] Teck Chew Ng, J. I. Guzman, and M. D. Adams. Autonomous vehicle-following systems : a virtual trailer link model. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3057–3062, 2005.

[10] tysik. obstacle detector. `https://github.com/tysik/obstacle_detector`, 2018.

[11] wentasah. kcf. `https://github.com/CTU-IIG/kcf/tree/master`, 2019.

[12] WolfWings. *Barrel distortion visual example.* Sep 2008.

[13] WolfWings. *Pincushion distortion visual example.* Sep 2008.