POLITECNICO DI TORINO

DEPARTMENT OF CONTROL AND COMPUTER ENGINEERING (DAUIN)
MASTER'S DEGREE IN MECHATRONIC ENGINEERING
2020/2021

Design and Control of an Autonomous Vehicle from a Radio Controlled 1/10" Car

Author:

Girardi Delia
274575

Prof. Tonoli Andrea
Prof. Amati Nicola
Eng. Bonfitto Angelo
Eng. Luciani Sara
Eng. Feraco Stefano

Abstract

In recent years driverless cars passed from being just an idea to taking a real shape, which brings along with it the possibility of a drastic decrease of car crashes and pollution. One of the problems to face when dealing with the production of Autonomous Vehicles is the high cost of prototypes, which might be severely damaged during testing sessions. The present thesis proposes a low-cost scaled model of a vehicle which is provided with a full line of perception, control, and actuation and can be used as a testing platform for complex algorithms, in a completely safe environment.
After a deep analysis of the state of the art, the work explores the entire process of building an autonomous vehicle starting from a 1:10 RC car. The core of the platform is a Raspberry Pi 4 Single Board Computer (SBC) which integrates a high computing capacity and a great compatibility with a wide range of sensors. The perception pipeline features a camera and an Inertial Measurement Unit (IMU), both managed in MATLAB environment using Raspberry Pi's dedicated support package. A Computer Vision algorithm performs lane detection and real-time estimation of variables such as road curvature, lateral deviation and heading error. Vehicle modelling is based on the well know bicycle model, whose parameters are estimated through CAD modelling and Grey-Box estimation. The complete definition of the model allows the implementation of lateral and longitudinal dynamics control techniques and of a Kalman filter for IMU data. Finally, the vehicle is actuated by means of a servo motor and a DC motor.
In the last part of this work, the results of real-time testing are presented. The vehicle showed the capability of driving autonomously under different light conditions on roads with curvatures radiuses up to 15m at a speed of 1m/s, with the possibility of reaching speeds up to 2m/s on less curvy roads. The only limits in terms of operating distances are set by the battery capacity, which lasts about 20 minutes under the most common conditions.

Contents

Introduction **7**
Thesis Motivation . . . 8
Thesis Outline . . . 8

1 State of the Art **11**
1.1 Rapid Prototyping for Autonomous Vehicles . . . 11
1.2 Perception . . . 12
1.2.1 LiDAR . . . 14
1.2.2 Radar . . . 14
1.2.3 Ultrasonic sensor . . . 15
1.2.4 Inertial Measurement Unit . . . 15
1.2.5 GPS . . . 16
1.2.6 Vision . . . 16
1.3 Control . . . 18
1.3.1 Pure Pursuit Controller . . . 23
1.3.2 Stanley controller . . . 25
1.3.3 Model Predictive Control . . . 27
1.3.4 Extended Kalman Filter . . . 28

2 Hardware **29**
2.1 Architecture of a RC Car . . . 29
2.1.1 DC Motor . . . 31
2.1.2 RC servo . . . 33
2.2 Architecture of the scaled Autonomous Vehicle . . . 33
2.2.1 Choice of board . . . 34
2.2.2 Working Principle in steps . . . 36

3 Perception **39**
3.1 Vision sensor . . . 39
3.1.1 Camera Calibration . . . 40
3.1.2 Lane Detection Algorithm . . . 43

3.2 Motion Detection-Inertial Measurement Unit 46
3.2.1 IMU Calibration 47
3.2.2 IMU Measurements and Integration 50
3.3 Longitudinal Speed Measurement 52

4 Control **55**
4.1 Plant Model: parameters Estimation 55
4.1.1 CAD Modeling 56
4.1.2 Grey Box Identification 59
4.2 Kalman Filter 63
4.3 Lateral Control 66

5 Actuation **67**
5.1 Steering Actuation 67
5.2 Control of DC motor 69
5.2.1 Characterization of DC Motor 71

6 Results **75**
6.1 Pre-loaded steering profile 75
6.2 Steering command from perception and constant longitudinal velocity . 78

7 Conclusions and Future Works **83**

List of figures **87**

List of tables **89**

References **91**

Introduction

Nowadays nobody would be surprised to see a car braking in an emergency, steering to stay inside the lane or maintaining a fixed velocity without human action[1].
Getting used to Advanced Driver Assistance Systems (ADAS), the World is preparing itself to a revolution of mobility where fully autonomous vehicles will be the main actors of the road.
The attractiveness of self-driving cars is not only related to science-fiction movies but mostly relies on much more tangible aspects. Features such as computer vision and sensors like radar, lidar, GPS are often more reliable than the human eye and allow faster reaction times making Autonomous Vehicles (AVs) less prone to crash risk. An extensive use of them will lead the way to cars which can travel closer and faster, thus enhancing road capacity[2]. Costs due to traffic congestion and parking will decrease together with pollution emissions and general comfort will increase[3].
The history of AVs has roots in the last century and includes radio controlled "driverless" cars and failed projects of roads with embedded electronics, but the event which unarguably set a turning point in this field was the DARPA Grand Challenge of 2004[4][5].
The U.S. Defense Advanced Research Projects Administration (DARPA) hosted a competition that consisted in traveling autonomously through the desert over a 142-mile course in Nevada[4]. Its aim was to find vehicles able to drive around war zones without risking the life of a human operator[6]. The prize for the winning team: 1million US Dollars.
Unluckily, there was no team able to complete the race, so DARPA organized another challenge for the next year, doubling the prize and increasing the difficulty. In the 18 months dividing the two events, hundreds of teams worked tirelessly to overcome the technological limits and in the end Stanford University's car, Stanley, won the race and many other teams succeeded to complete the route.
As stated by the 2005 DARPA Grand Challenge Program Manager, those who were touched by this experience would "create a new future, one that includes autonomous ground vehicles"[7]. This event was indeed a great boost for the development of AVs.
From those times a lot of efforts were made by Universities and Companies towards autonomous mobility. To name a few:

- Google Waymo, which developed its autonomous car in 2010, led by one of the designers of Stanley's car[8]
- Tesla, whose "Autopilot" is the most famous autonomous vehicle system in the world[9]
- Audi, which announced a SAE Level 3[10] A8 to be on the market in 2017[11]. Later the project was abandoned, much likely for legislative reasons

- Uber, which fits in the landscape of autonomous taxis. It is estimated that this field will be the most impacted one, in terms of cost reduction, by the advent of AVs [12].

Nevertheless, ethical and legislative issues set a high barrier to the appearance of self-driving vehicles as a common means of transport. Recent studies report that this will not happen before 2060s[12]. Anyway, I am confident that if a great technical effort is made in this direction, everybody will be more and more convinced that the pros outweigh the cons, and this target will get closer.

Thesis Motivation

The goal of this thesis is to build a prototype Autonomous Vehicle basing on a 1/10 scale RC car model. The latter will be equipped with all the hardware and software necessary to allow the completion of perception, control, and actuation task of a full-scale vehicle, to consent the testing of complex algorithms in a safe and cheap environment. Moreover, it will be made available for didactic purposes and, after the development of a twin vehicle, it will be a platform for vehicle-to-vehicle communication.

Thesis Outline

The thesis is organized as follows:

- *Chapter 1* introduces the current state of the scientific research on the subject of Autonomous Vehicles and all the related topics considered useful for the completion of this work, such as rapid prototyping for autonomous vehicle, perception, and control.
- *Chapter 2* presents the architecture of a RC car and all the modifications made from the hardware point of view, in order to transform the vehicle into an autonomous model.
- *Chapter 3* describes the perception pipeline, showing the calibration techniques employed to achieve the best results from the two sensor used: a camera and an IMU. Moreover, it illustrates the lane detection algorithm and the issues met during the use of the IMU.
- *Chapter 4* deals with vehicle modeling and proposes a strategy to obtain unknown plant parameters through CAD modeling and grey-box estimation. The identification of the model is then exploited to design a Kalman Filter and a controller for the vehicle.
- *Chapter 5* analyses the actuation system, composed by a micro servo and a DC motor, clarifying the working principle of both devices and the method used to control them through Raspberry Pi.
- *Chapter 6* reports the results obtained in two different stages of testing, to assess the correct working of the actuators and of the perception and control algorithms. First, the steering command is pre-loaded, then it is chosen in real-time from a lateral controller Stanley, driven by camera and IMU information.

- *Chapter 7* discusses the final considerations about this master thesis work and proposes new path which can lead to further improvements of this project.

Chapter 1

State of the Art

1.1 Rapid Prototyping for Autonomous Vehicles

As it occurs in many other fields, the development of an autonomous vehicle passes through a series of stages, the latest being testing. Needless to say, this is necessary not only to validate the project from the technical point of view, but also to ensure safety in the real-world employment.
In the last years, several software packages have been developed to allow high-fidelity simulation, involving complex mathematical models able to realistically represent the system dynamic. Thanks to these tools, it is possible to customize the whole sensors equipment and the operating scenario, also reproducing existing maps [13].
Although simulators are becoming more and more accurate, they still cannot account for all the variables into play. Moreover, the use of real car-based testbed is not only highly expensive but also poses serious safety concerns that hinder development and exploration [14]. From this perspective, scaled vehicle models prove to be a good comprise and fosters research in autonomous systems, making it much more affordable.
Beside the cost, the advantages of using scaled prototypes are countless. They can be equipped with single-board computers like Jatson Nano or Raspberry Pi, which allow to process a huge quantity of data occupying a space comparable to a credit card. In this way, almost every aspect of the vehicle can be tested, like perception, path planning and control [15], using the same strategies that would be used in a full-scale car. The absence of safety issues allows testing in real time and in the most extreme circumstances reaching the limits of performance.
All these features make these scaled models also very interesting from the didactic point of view. For this reason, there are a lot of universities and research groups, like TUM [16] and MIT [17], which carry on in parallel research on small prototypes and full-scale models, being also very successful on autonomous cars races, like Roborace.
In particular, MIT was one of the first to host a race for autonomous RC cars [17]. In 2014, control theory students were invited to build their RACECAR, Rapid Autonomous Complex Environment Competing Ackermann-steering Robot, using mainly a Nvidia Jetson TK1 and a 1:10-scale RC car. The challenge was to manufacture a mini-robot car that could zip around a tunnel maze track while navigating its twists and turns. To elect the winner each car was timed while driving alone around the basement hallways of the university. During the next years, new models of the RACECAR were developed, the latest being the one in figure 1.1.

Another race featuring scaled-down autonomous vehicles which was of the upmost im-

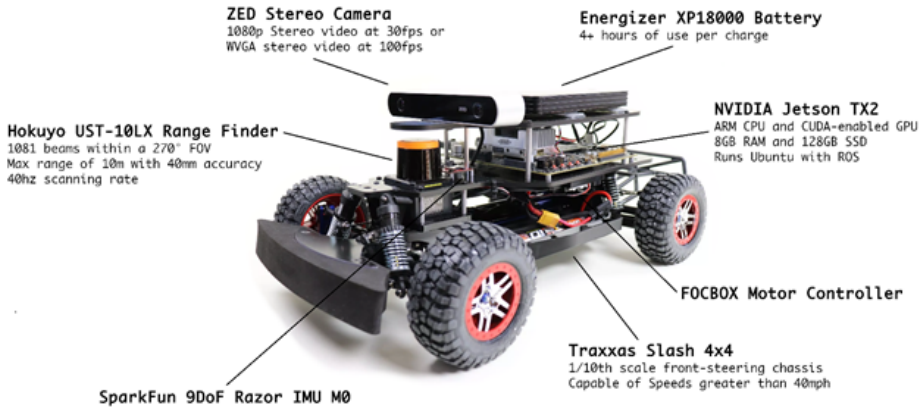


Figure 1.1: Newest MIT RACECAR platform [17]

portance, was the DIYRobocars hackathon of 2016. The importance of this race does not lie in any revolutionary discovery but in a project that was born in that occasion: the Donkey Self Racing Car [18].
The two authors of the Donkey Car met during the hackathon and started collaborating to build a model which could drive itself by means of computer vision and machine learning techniques. They began with a RC car, got rid of the receiver and added mainly a Raspberry Pi and a camera. Then, when the project was over, they decided to make it open source, creating a community where everybody is invited to experiment, innovate and share.
The Donkey Car Community is probably the widest existing, but it is not the only one. There is indeed a group counting more than 20 institutions, starting from University of Pennsylvania, University of Virginia and UNIMORE. They are named F1/10 [15] and built an open-source, high performance 1/10 scale autonomous vehicle testbed which carries a full suite of sensors, perception, planning, control and networking software stacks that are similar to full scale solutions.
While the audience of these two communities is slightly different, the purpose is the same: giving a boost to the development of autonomous systems, making knowledge accessible and affordable to anyone.
In the last five years alone, researchers from all over the world used these prototypes in an impressive range of applications. It was indeed possible to test complex control schemes, path planning and obstacle avoidance algorithms, and deep learning techniques, often exploiting computer vision and a wide range of sensors [19] [20] [21] [22].

1.2 Perception

The architecture of an autonomous system can be divided into two main parts: hardware and software. The first has the duty to interface with the environment through sensors, communication and actuation, while at the software side data coming from sensors and communication are collected and elaborated in order to extract relevant knowledge from the environment (perception). This knowledge is exploited from the

planning block to make a decision, which is executed by the actuators following the lead of the control block. A graphical representation of this workflow is given in figure 1.2.

The perception pipeline is essential for the operation of an autonomous vehicle as

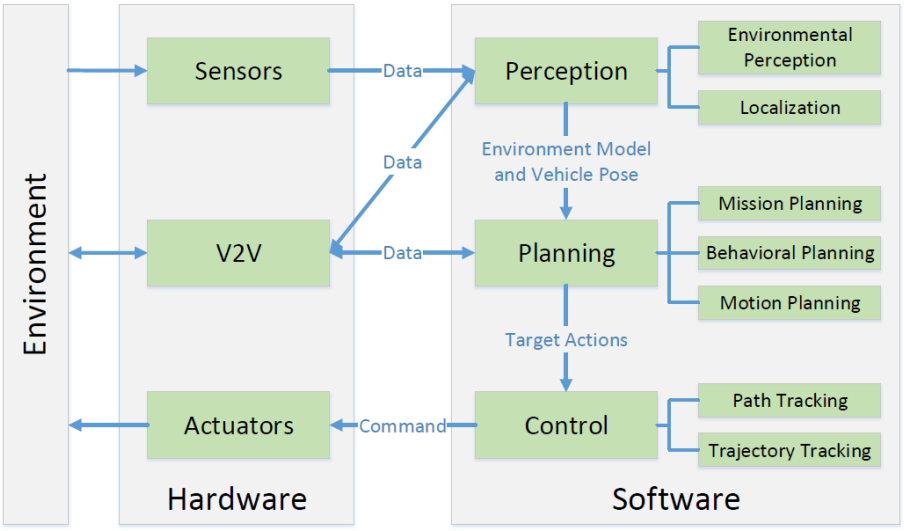


Figure 1.2: Architecture of an autonomous system [23]

it measures the conditions outside the vehicle and provides its position relative to its environment. To this purpose sensors like LiDAR, radar, ultrasonic range, GPS, IMU and Vision are of great help. Since they have overlapping and complementary capabilities, they can be used together along with data fusion strategies that combine real-time information (figure 1.3).

The main sensors used for navigation and control will be described in this section.

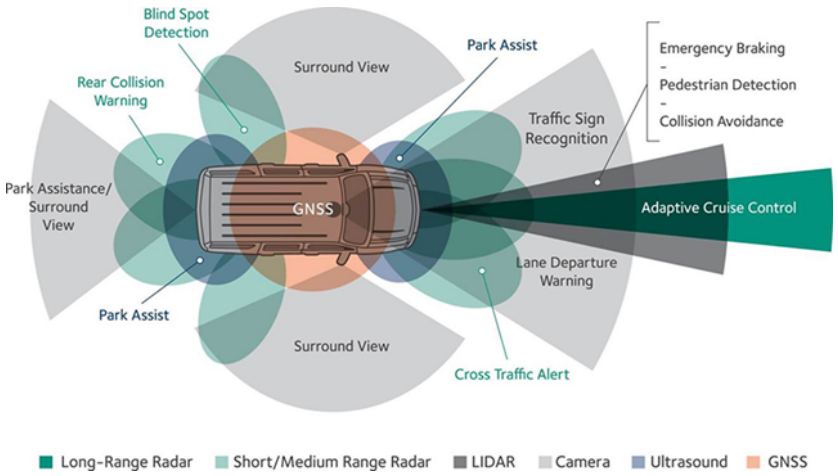


Figure 1.3: Sensors and their range in an autonomous vehicle [24]

1.2.1 LiDAR

LiDAR (Light Detection and Ranging) is a device which sends millions of light pulses per second and register the amount of light reflected back by any non-absorbing object or surface. Thanks to the rotation about its axis, it can create a 3D high-resolution point cloud representing the surroundings. The target distance is computed as speed of light times the measured time period at multiple angles. Then the distance is used to construct a three-dimensional map of the world including the objects around it.
Figure 1.4 shows the ideal detection from a 3D LiDAR with all the moving object identified. Anyway, problems like scan point sparsity, missing point and unorganized patterns, can make the visualization of the scan point hard to understand from human beings [23]. On the contrary, the output of a camera is much easier to interpret but with the flaw that camera-based object detection is sensitive to light and environment conditions and does not work well in rain, snow, and dust. The behaviour of a LiDAR, in this case, is unaffected.

To make the perception pipeline robust to sensor failure, several methods have been

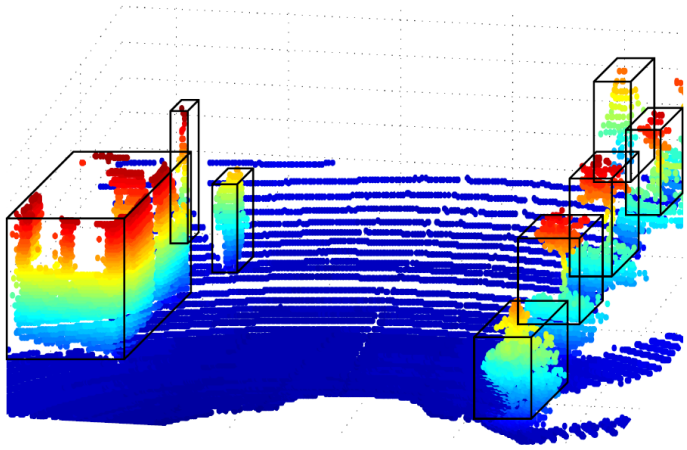


Figure 1.4: Ideal detection from a 3D LiDAR[23]

proposed, such as the deployment of parallel and independent sensing and estimation pipelines based on camera and LiDAR, allowing to get the most accurate information on the distance of the detected object and also on their visual features like colour [25].

1.2.2 Radar

Radar (Radio Detection and Ranging) has a working principle very similar to LiDAR. It transmits electromagnetic pulses and senses the echoes to detect and track objects. In this way the relative distance, velocity and orientation of an object can be detected. It can work for short, medium, and long ranges ($\simeq$ 30-200m) with frequencies going respectively from 100+GHz to 3MHz. Since they require less computational effort than LiDAR, currently they are used in many ADAS applications, such as advanced cruise control and object detection [26].

1.2.3 Ultrasonic sensor

In the case of ultrasonic sensor, it is an ultrasonic pulse wave which is sent and reflected back. Their range is very short ($\simeq$ 2m) but they present the advantage of not being affected by light and weather conditions. For example, there is a very small amount of light reflected by a glass, while an ultrasonic wave is perfectly reflected. These features make this sensor suitable for low speed ADAS modules like parking space detection and assistance [27].
Figure 1.5 shows a cheap sonar which is commonly used by makers thanks to its easy integrability with most microcontrollers and SBCs.

Figure 1.5: Ultrasonic sensor HC-SR04

1.2.4 Inertial Measurement Unit

An Inertial Measurement Unit (IMU) is a device used for navigation purposes where the position and orientation of a system are of interest. Typically, its output is the 3-DOF acceleration and the 3-DOF angular rate with respect to the body's reference frame, which are measured by means of an accelerometer and a gyroscope. Beside these 6-DOF IMUs, there are devices with an embedded magnetometer which also returns the body's orientation with respect to Earth's magnetic field (9-DOF IMU). In some cases, also a barometric pressure sensor is available and can be used to calculate altitude. Anyway, 10-DOF IMUs are mainly employed in aerospace or underwater applications, where pressure variation is high and becomes a meaningful information.
Depending on the quality of the parts, the price of an IMU ranges from $\simeq$ €10 for sensor used in smartphones up to tens of thousand euros for military and navigation applications. Although there are many ways to fabricate accelerometers, gyroscope and magnetometers, the most spread are Micro Electro-Mechanical System (MEMS).
By simple integration of accelerometer data and knowing the initial conditions, the body's velocity can be known, and through another step of integration also the position with respect to a given reference frame can be computed. Similarly, thanks to the gyroscope, angular rate and position can easily be obtained. Furthermore, if a magnetometer is available, several techniques of sensor fusion can be applied to get more accurate measurements.
From the practical point of view, things are not that simple. For instance, acceleration measures are strongly affected by gravity, whose influence on measurements is not negligible [28]. Imagine, for example, a car breaking. Due to inertia, the vehicle will tend to bend towards the front part and the axis of the accelerometer oriented as the

longitudinal axis of the vehicle will be suddenly hit by gravity, thus recording a longitudinal acceleration which actually does not exist. This example clearly shows that accelerometer data are not ready for use.
Furthermore, due to the influence of semiconductor thermal noise and electromagnetic interference, the output of the sensor often suffers from a random noise and drift [29]. A first correction of these errors can be made by calibration, which must be repeated frequently over time. Moreover, since IMU data are collected at very high sampling rates, which are accurate only on a short time scale, they can be combined with sensors with a lower sampling rate whose information does not drift over time. To this end a sensor very used in the automotive field is GPS [30] and the fusion of the two information is often performed by means of a Kalman Filter or a modified version of it.

1.2.5 GPS

GPS (Global positioning system) was initially developed by the U.S. Department of Defence to get precise positioning for rocketry and later extended to civil applications thanks to its versatility.
The system uses a constellation of 18-30 medium Earth orbit (MEO) satellites spread between several orbital planes whose position is generally descripted in geocentric cartesian reference frame, with origin in the Earth's centre, the Z axis in the direction of the Earth rotation axis and axes X and Y on the equatorial plane. These satellites maintain their position relative to earth and broadcast reference signals [26]. A GPS receiver collects some of these signals (generally from 4 or more satellites) generates a replica of them and send them back to the satellites. From the time shift of the signal and the knowledge of signal speed, the distance from each satellite is computed and the location of the receiver is detected using a spatial intersection [31]. This working principle is represented in figure 1.6.
This technique, known as point positioning, allows a precision in the order of meters but there exists also another technique, the differential positioning. In this case another actor is involved, which is a point whose position is known a priori and is used as a further reference to improve accuracy. Through this method the location of a receiver can be assessed within centimetre accuracy.

1.2.6 Vision

Most of the above-described sensors measure the distance by sending signals to the target and computing the time shift between echoes of this signal. One of their main inconvenient is the potential confusion of echoes from subsequent pulses or the unpredictable reflection of waves while meeting object edges [32]. Vision sensors capture more visual information, hence tracking the surrounding environment more effectively than other sensors. They are categorized into mono and stereo types.
Mono vision systems use one camera to estimate the distance. To do so, a precise process of calibration must be performed. In particular, the mapping between world coordinates and pixel coordinates passes through two transformation matrices. First, a roto-translation matrix (extrinsic parameters) allows to pass from world coordinates to camera coordinates. Secondly, a matrix defined by intrinsic camera parameters like

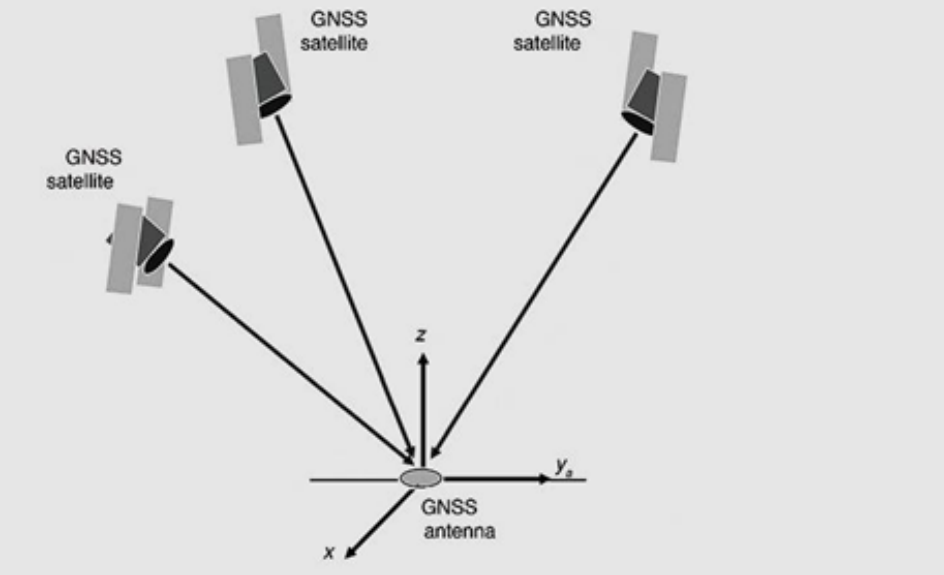


Figure 1.6: GPS working principle [31]

focal length, optical centre and skew coefficient, projects camera coordinates in 2-D pixel coordinates [33]. Thanks to the calibration process, pixel distance is translated in real world distance. Figure 1.7 depicts the mapping procedure described above.
If a wider field of view is needed, wide angle cameras can be used as well. In this case

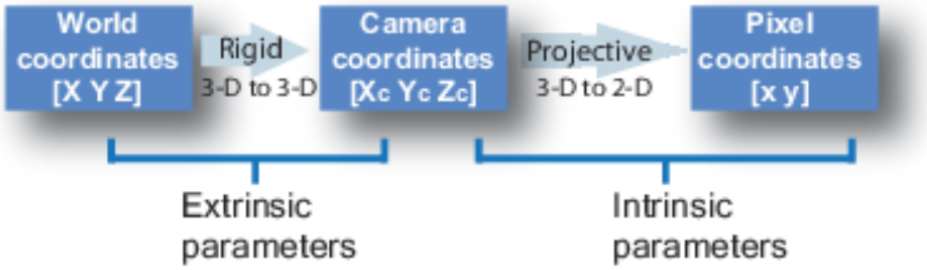


Figure 1.7: Mapping from World coordinates to pixel coordinates [34]

a further calibration process is needed to correct lens distortion [35].
Mono vision methods are particularly effective when the type of the object is known such as in lane detection, basic object detection and road sign detection [26]. Anyway, they become very complex when a previous object recognition is required, as it leads to high computational efforts and more uncertain results. In these circumstances, a stereo vision system must be preferred.
A stereo camera is simply a device made of two mono cameras placed at a known distance (figure 1.8). The basic concept is to record a scene from two different viewpoints and then exploit the disparity to compute the position relation and structure of objects in the scene [32]. A visual explanation is provided in figure 1.9.

Figure 1.8: Stereo camera [36]

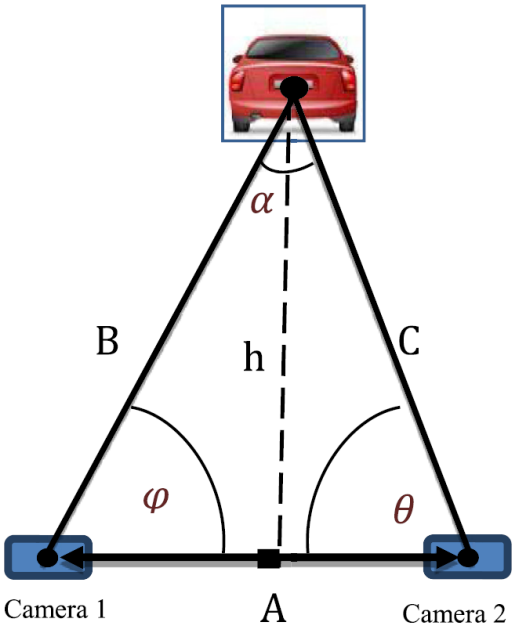


Figure 1.9: Stereo camera working principle [32]

1.3 Control

The motion of any rigid body can be described considering six Degrees of Freedom: three rotations and three translations about the axes of a 3-D reference frame. In the case of vehicles, ISO 8855-2011 established a standard to set a vehicle's reference frame (figure 1.10).

The reference frame is placed in the vehicle's centre of gravity. The longitudinal axis X points in forward direction, the vertical axis Z points against gravity and the lateral axis Y is set following the right-hand rule. Three angles are defined, which represent respectively the counterclockwise rotation about axes X, Y and Z: roll ϕ, pitch θ, yaw ψ.

Although several models can be found in the literature to fully describe a vehicle's motion, this section will not be focused on the vehicle's vertical dynamic. Hence only three of the above cited degrees of freedom will be considered: longitudinal position x, lateral position y and rotation about the z axis ψ. Those can be well visualized in a top view as in figure 1.11.

Both the front wheels are steering and to avoid slippering it must be

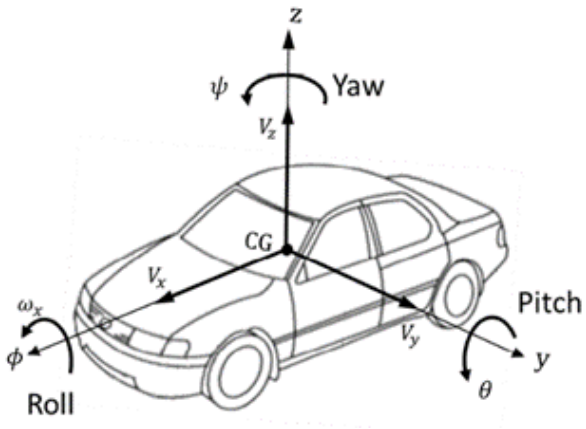


Figure 1.10: ISO 885-2011 reference frame [37]

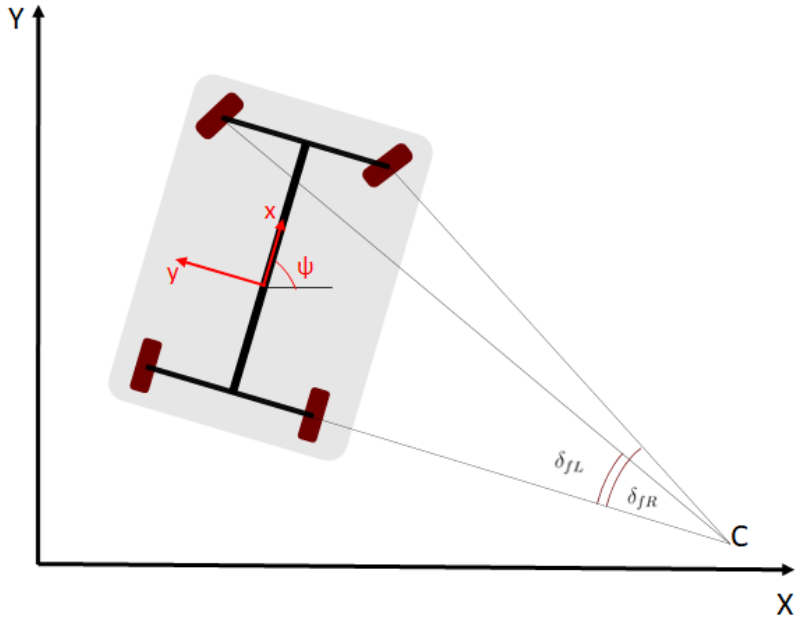


Figure 1.11: Vehicle top-view

$$\delta_{f,Right} \neq \delta_{f,Left}$$

Anyway, the difference between the two steering angles can be considered negligible and thus the model can be further simplified. Figure 1.12 represents the so-called *bicycle model*, where the vehicle is modeled as a single-track.

With respect to an inertial reference frame X,Y, the vehicle-fixed frame x, y is translated in the plane and rotated of an angle ψ. This relation is described by the equations below:

$$X = xcos\psi - ysin\psi \tag{1.1}$$
$$Y = xsin\psi + ycos\psi \tag{1.2}$$
$$\Psi = \psi \tag{1.3}$$

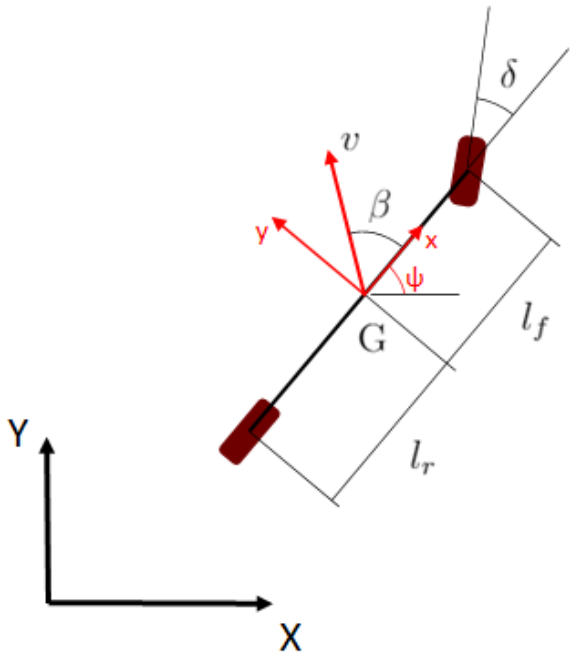


Figure 1.12: Bicycle model

The equations of motion of the model in the vehicle-fixed reference frame can be expressed as:

$$\begin{cases} \sum F_{ext_{x_v}} = ma_{G_x} \\ \sum F_{ext_{y_v}} = ma_{G_y} \\ \sum M_{ext_G} = I_G \ddot{\Psi} \end{cases} \tag{1.4}$$

In order to explicit the terms inside the sum, a kinematic analysis of the model must be done. To this end it is useful to define some variables, with reference to figure 1.12:

- G: vehicle's centre of mass
- l_f: distance between G and the vehicle's front axle
- l_r: distance between G and the rear axle
- δ: steering angle
- v: velocity of the centre of mass
- β: sideslip angle, is the direction of the velocity vector with respect to the vehicle-fixed reference frame

The longitudinal and lateral velocity are then expressed as:

$$V_x = V cos\beta \tag{1.5}$$
$$V_y = V sin\beta \tag{1.6}$$

From kinematic considerations, the acceleration of the centre of mass is:

$$a_{G_x} = a_x - V_y\dot{\psi} \tag{1.7}$$
$$a_{G_y} = a_y + V_x\dot{\psi} \tag{1.8}$$

During motion the centrifugal force causes the deformation of tires (figure 1.13), so the steering is no longer kinematic but has its own dynamic which influences the vehicle's trajectory.

The slip angle is defined as the angle between the rolling direction of the tire and the

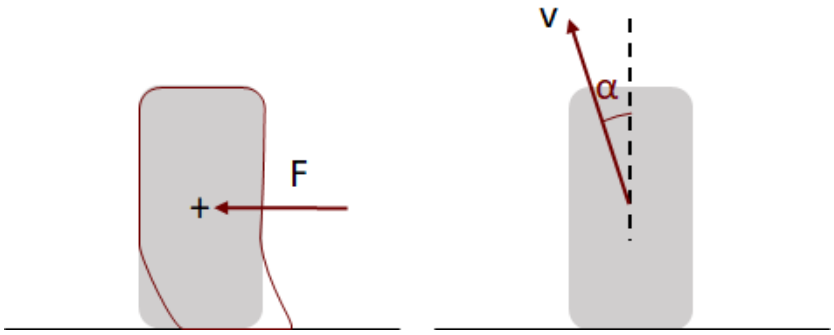


Figure 1.13: Tire deformation and slip angle

direction of its velocity. So, for the rear wheel it is:

$$\alpha_r \simeq tg\alpha_r = \frac{V_{y_r}}{V_{x_r}} = \frac{V_y - \dot{\psi}l_r}{V_x} \tag{1.9}$$

A similar expression is obtained for the front wheel, which is the steering one:

$$\alpha_f \simeq tg\alpha_f = \frac{V_{y_f}}{V_{x_f}} + \delta = \frac{V_y + \dot{\psi}l_f}{V_x} + \delta \tag{1.10}$$

With this in mind, a more complete model can be defined.

Figure 1.14 shows the 3-DoF dynamical model, in which also the forces interacting with the vehicle are present. The lateral tire forces at the front and rear wheels are considered perpendicular to the rolling direction of the tire. Considering the assumption of small slip angles, the lateral tire forces can be modeled as:

$$F_{yf} = 2C_{\alpha f}\alpha_f \tag{1.11}$$
$$F_{yr} = 2C_{\alpha r}\alpha_r \tag{1.12}$$

where $C_{\alpha f}$ and $C_{\alpha r}$ are the tire stiffness[38].

Now all the bases are set to rewrite equations in (1.4) explicitly:

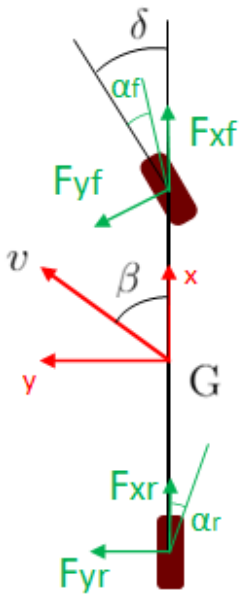


Figure 1.14: Dynamical model

$$\begin{cases} m(a_x - V_y\dot{\psi}) = F_{xr} + F_{xf}cos\delta - F_{yf}sin\delta \\ m(a_y + V_x\dot{\psi}) = F_{yr} + F_{yf}sin\delta - F_{yf}cos\delta \\ I_z\ddot{\psi} = l_f F_{xf}sin\delta + l_f F_{yf}cos\delta - l_r F_{yr} \end{cases} \tag{1.13}$$

where F_{xf} and F_{xr} are the components of the force provided by the front and rear tires in the rolling direction.
The above equations can be further manipulated in order to obtain a state space description of the system, which can be necessary to apply some control techniques. Considering as state vector

$$x = \begin{bmatrix} v_y \\ \psi \\ \dot{\psi} \end{bmatrix}$$

the equations are written in the form

$$\dot{x} = Ax(t) + Bu(t) \tag{1.14}$$

as follows:

$$\begin{bmatrix} \dot{v_y} \\ \dot{\psi} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} -\frac{2C_{\alpha f} + 2C_{\alpha r}}{mV_x} & 0 & -V_x - \frac{2C_{\alpha f}l_f - 2C_{\alpha r}l_r}{mV_x} \\ 0 & 0 & 1 \\ -\frac{2C_{\alpha f}l_f - 2C_{\alpha r}l_r}{I_z V_x} & 0 & -\frac{2C_{\alpha f}l_f^2 + 2C_{\alpha r}l_r^2}{I_z V_x} \end{bmatrix} \begin{bmatrix} v_y \\ \psi \\ \dot{\psi} \end{bmatrix} + \begin{bmatrix} \frac{2C_{\alpha f}}{m} \\ \frac{2C_{\alpha f}l_f}{Iz} \end{bmatrix} \delta \tag{1.15}$$

1.3.1 Pure Pursuit Controller

Among the most used algorithms used for lateral control there is *pure pursuit*, whose first applications date back to early 80's [39].
Pure pursuit is not a traditional controller but acts as a tracking algorithm for path following purposes. It geometrically determines the curvature that will drive the vehicle to a chosen goal point. The latter is determined as the point of the path which locates at one *lookahead distance* from the vehicle's position. This principle resembles the way humans drive; we look some distance in front of the car and head toward that point, then look farther and adjust the steering to the following point.
Figure 1.15 shows the main parameters of pure pursuit algorithm. The vehicle's coor-

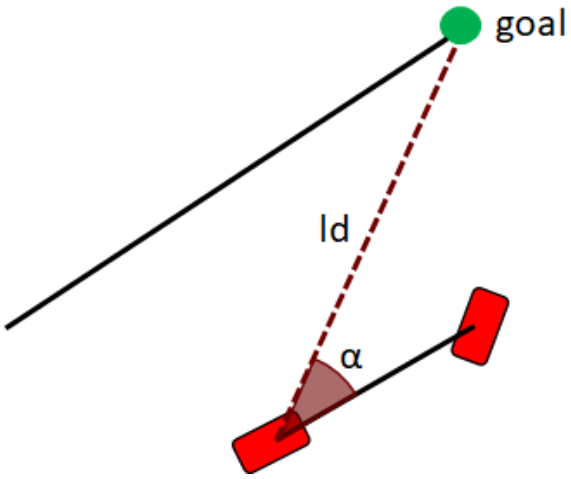


Figure 1.15: Pure pursuit parameters

dinate system is placed at the centre of the rear axle. It has been demonstrated that in this way the propulsion and steering are geometrically decoupled.
The goal is found on the path as the point at one look ahead distance ld, and an angle α is defined between the vehicle's longitudinal axis and the line connecting the rear axle and the goal point. An arc of radius R is then built as in figure 1.16
The aim is to compute the curvature of this arc and the steering angle needed to track the arc. Building an isosceles triangle having the look ahead distance as base and the apex angle equal to 2α, the following relations can be written, starting from the law of sines:

$$\frac{ld}{sin2\alpha} = \frac{R}{sin(\frac{\pi}{2} - \alpha)} \frac{ld}{2sin\alpha cos\alpha} = \frac{R}{cos\alpha} \frac{ld}{sin\alpha} = 2R \tag{1.16}$$

Defining the *cross-track error* e as the distance between the goal point and the line passing from the vehicle's longitudinal axis

$$sin\alpha = \frac{e}{ld} \tag{1.17}$$

the final expression of the curvature k is

$$k = \frac{1}{R} = \frac{2sin\alpha}{ld} = \frac{2}{ld^2}e \tag{1.18}$$

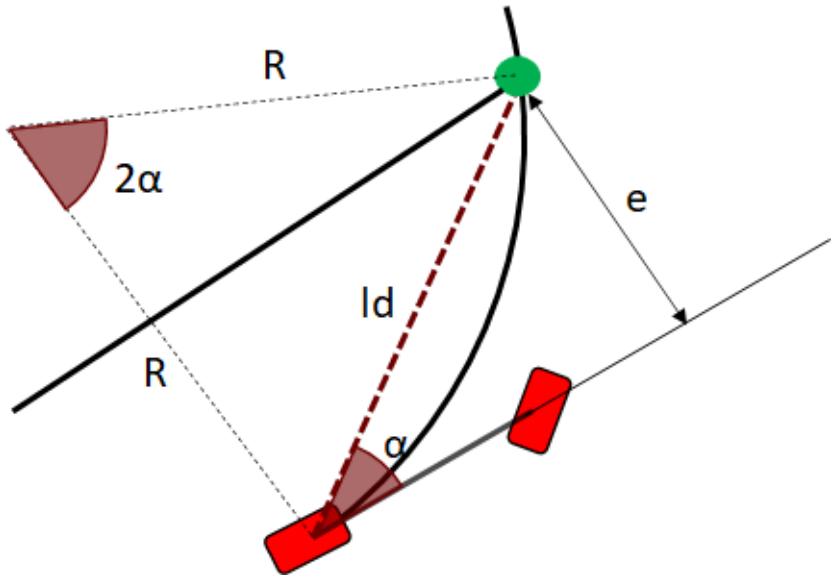


Figure 1.16: Pure pursuit-arc construction

The latter shows a similarity in form to a proportional controller based on the cross-track error whose gain is 2 times the inverse square of the ld.
The steering action to be taken, is computed referring to figure 1.17:

$$\delta = arctan(\frac{L}{R}) = L \cdot k \tag{1.19}$$

where δ is the steering angle and L is the distance between the front and rear axle.
In summary the only parameter to tune in pure pursuit is the look ahead distance.

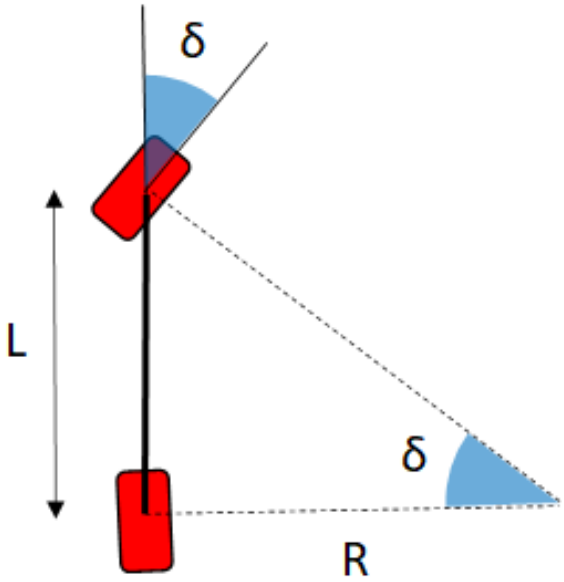


Figure 1.17: Relation between steering angle and curve radius

This can be a tricky task. When ld is too small the goal is reached faster, but the trajectory tends to be oscillatory, thus involving a continuous and useless steering action. Large ld lead to smoother trajectories but slower convergence to the path (figure 1.18). A good procedure can be changing the value of ld proportionally to the longitudinal velocity. Anyway, this control law proves to be not so effective, but still, it is valid thanks to its simplicity and to the fact that it solves the problem of lateral control and path planning at the same time.

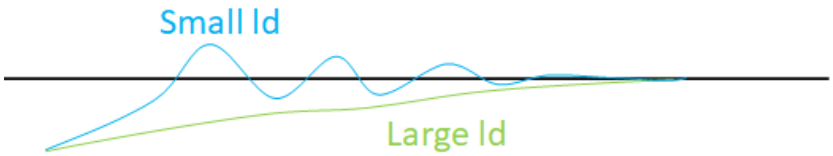


Figure 1.18: Effects of different ld on the trajectory

1.3.2 Stanley controller

The second lateral control strategy examined is Stanley controller, the one implemented by Stanford University's team to win the 2005 DARPA Grand Challenge [40].
The Stanley lateral controller uses a nonlinear control law to minimize the cross-track error and the heading angle of the front wheel relative to the reference path. Depending on the vehicle's velocity two model are considered:

- Kinematic bicycle model: it assumes that the vehicle has negligible inertia and allows the design of a controller stable under that assumption. This works when velocity is kept low

- Dynamic bicycle model: it includes inertial effects as tire slip and steering servo actuation. Despite being more complicate it allows to handle realistic dynamic.

In the Kinematic model, differently from pure pursuit, the crosstrack error $e(t)$ is defined as the distance between the centre of the front axle and the path to track. $\psi(t)$ is the heading angle of the vehicle with respect to the closest trajectory segment and $\delta(t)$ is, as usual, the steering angle. The derivative of the crosstrack error is computed as

$$\dot{e}(t) = v(t)sin(\psi(t) - \delta(t)) \tag{1.20}$$

while the derivative of the yaw angle is

$$\dot{\psi}(t) = -\frac{v(t)sin(\delta(t))}{lf + lr} \tag{1.21}$$

In this frame, the following control law is determined

$$\delta(t) = \begin{cases} \psi(t) + arctan\frac{ke(t)}{v(t)} & if|\psi(t) + arctan\frac{ke(t)}{v(t)}| < \delta_{max} \\ \delta_{max} & if|\psi(t) + arctan\frac{ke(t)}{v(t)}| \geq \delta_{max} \\ -\delta_{max} & if|\psi(t) + arctan\frac{ke(t)}{v(t)}| \leq \delta_{max} \end{cases} \tag{1.22}$$

As it can be seen the steering angle is essentially proportional to the crosstrack error and to the inverse of the velocity. The contribution of the inverse tangent limits the steering for large errors. Moreover, the mechanical limits of steering, saturate the command input.
In the nominal case (i.e., outside the saturation region) the derivative of the crosstrack error becomes

$$\dot{e}(t) = -v(t)sinarctan(\frac{ke(t)}{v(t)}) = \frac{-ke(t)}{\sqrt{(\frac{ke(t)}{v(t)})^2}} \tag{1.23}$$

Which for small track errors, leads to exponential decay characteristics

$$\dot{e}(t) \simeq -ke(t) \implies e(t) = exp(-kt) \tag{1.24}$$

Figure 1.19 shows the effect of Stanley control law on a vehicle having large initial

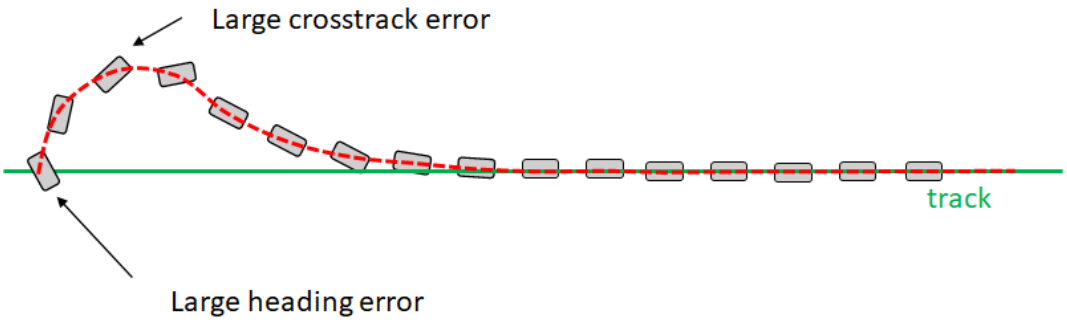


Figure 1.19: Effect of Stanley controller on large heading error

heading error. First, to reduce this error, the vehicle drives in the maximum steering condition. After a while the heading error becomes small but the crosstrack error increases. The controller tries to bring the vehicle straight toward the track and then it converges to the trajectory exponentially with time constant k. With respect to pure pursuit controller, it can be noticed that the obtained trajectory is much smoother.
The presence of the velocity term at the denominator of the *arctan* function can be a problem in case of velocities close to zero, leading to a high gain $\frac{k}{v(t)}$ also in the case of very small e. To prevent numerical instability, a softening constant k_{soft} is added to the control law.
An additional consideration must be done for curvy roads, where lateral acceleration is generated by both front and rear tires while the vehicle points inward. The controller yaw set point should be non-zero and must be related to the steady state yaw ψ_{ss} which

can be found from equation (1.13)

$$\psi_{ss} = \frac{mv(t)\dot{\psi}_{traj}(t)}{C_y(1 + \frac{lr}{lf})} \tag{1.25}$$

The first equation of (1.22) becomes

$$\delta(t) = (\psi(t) - \psi_s s(t)) + arctan\frac{ke(t)}{v(t) + k_{soft}} \tag{1.26}$$

Further terms can be added to consider the steering delay and add active damping to stabilize the yaw dynamics. Anyway equation (1.26) alone is enough to provide an efficient lateral control action which takes into account also the vehicle's dynamic.

1.3.3 Model Predictive Control

Model Predictive Control (MPC) is a powerful control strategy which takes advantage of the knowledge of the system to predict its future behaviour and then choice the control action which provides an optimal solution of the problem. A peculiarity of MPC is its capability of handling input and output constraints, which makes it very suitable to the control vehicle dynamics, where maximum speeds and steering angle must not be overcome.
The MPC takes as input the reference and the state vector and evaluates the future behaviour of the system inside a chosen prediction horizon T_p. Then it minimizes a cost function to determine the optimal steering command inside a given control horizon $T_c \leq T_p$. Only the first control step is taken and then the prediction horizon is moved one-step forward. Everything is computed in discrete time. The optimization problem to be solved at each time step is [38]:

$$\min_{\mathrm{u}} J = \sum_{j=1}^{N_y}\sum_{i=1}^{T_p} ||y_j(k+i|k) - y_{j,ref}(k+i|k)||_{Q_y} + \sum_{j=1}^{N_u}\sum_{i=0}^{T_c-1} ||u_j(k+i|k) - u_{j,ref}(k+i-1|k)||_{R_u} \tag{1.27}$$

subject to

$$x(k+j+1|k) = Ax(k+j|k) + B_u u(k+j|k) + B_d v(k+j|k) \tag{1.28}$$
$$x(k|k) = x(k) \tag{1.29}$$
$$y(k+j|k) = Cx(k+j|k) \tag{1.30}$$
$$|u(k+j|k)| \leq u_{limit} \tag{1.31}$$

Notice that equations (1.28) and (1.30) are basically the state space description of a system in discrete time, with matrix C properly chosen basing on the measured output. Some attention must be paid to the fact that continuous-time matrix A defined in (1.15) is not constant and depends on the longitudinal velocity V_x. One way to solve this problem is to use a variant of MPC, called Adaptive MPC. In this case the plant mathematical model is updated at each operating point and the system is then treated

as linear time-invariant.

1.3.4 Extended Kalman Filter

The knowledge of the plant model can be exploited to filter noisy signals coming from the sensors. One of the most known methods for doing so is Kalman Filter for LTI systems and its Extended version for nonlinear systems. The latter has a more general formulation and can be applied in cases like the one under exam, where the state matrix is variable (in this case matrix A depends on V_x).
A discrete-time nonlinear system is considered [41]:

$$\begin{cases} x_{k+1} = f(x_k, u_k) + d_k \\ y_k = h(x_k) + d_k^y \end{cases} \tag{1.32}$$

where k is the discrete time, x_k is the state, u_k is the input, y_k is the output, d_k is a disturbance and d_y^k is a measurement noise.
Although a continuous time formulation of the problem can be adduced, the discrete time one is simpler and more suitable for on-line implementation.
The goal is to obtain an estimate $\hat{x}_k$ of x_k from current and past measurements y_k and u_k. Defining:

- $F_k = \frac{\partial f}{\partial x}(x_k, u_k)$: Jacobian of f computed in (x_k, u_k)
- $H_k = \frac{\partial h}{\partial x}(x_k)$: Jacobian of h computed in x_k
- $\hat{x}_k$: estimate of x_k, computed at step k
- x_k^p: prediction of x_k, computed at step k-1
- $P_k = E[(x_k - \hat{x}_k)(x_k - \hat{x}_k)^T]$: covariance matrix of $x_k - \hat{x_k}$
- $Q^d = E[d_k d_k^T]$: covariance matrix of d_k
- $R^d = E[d_k^y (d_k^y)^T]$: covariance matrix of d_k^y

The Extended Kalman Filter (EKF) algorithm can be presented.
Prediction:

$$\begin{aligned} \hat{x}_k^p &= f(\hat{x}_{k-1}, u_{k-1}) \\ P_k^p &= F_{k-1} P_{k-1} F_{k-1}^T + Q^d \end{aligned} \tag{1.33}$$

Update:

$$\begin{aligned} S_k &= H_k P_k^p H_k^T + R^d \\ K_k &= P_k^p H_k^T S_k^{-1} \\ \Delta y_k &= y_k - h(\hat{x}_k^p) \\ \hat{x}_k &= \hat{x}_k^p + K_k \Delta y_k \\ P_k &= (I - K_k H_k) P_k^p \end{aligned} \tag{1.34}$$

The algorithm must be initialized with chosen $\hat{x}_0$ and P_0 and values of Q^d and R^d must be set. Typically, they are chosen as diagonal matrices with variances of d_k and d_k^y on the diagonal but a trial and error tuning is often required, especially in those case in which d_k and d_k^y are not exactly known.

Chapter 2

Hardware

The first step to take to transform a RC car into an Autonomous Vehicle, is to define all the modifications that must be done from the hardware point of view. This means studying the working principle of an RC car and understanding which parts can be kept as they are, which must be substituted and which are completely missing.

2.1 Architecture of a RC Car

RC cars can be classified basing on two main features: body type and scale. Although the body type affects the vehicle dynamics, for the purpose of this project, the most crucial choice is the model scale. The vehicle must be big enough to carry all the required hardware, still respecting the principle of being low-cost.
RC cars are produced in standard scales (figure 2.1), going from 1:5 to 1:64. A 1:10 scale car seems to be the most suitable, as it allows to place all the hardware safely, without requiring much energy to work and keeping the cost of all parts low.
Figure 2.2 represents the vehicle chosen for this project, a 1:10 scale, 4 wheel-driving, RC short course truck, capable of reaching speeds up to 46km/h. Its main features are collected in table 2.2.

Once the shell and the top cover are removed, the car appears as in figure 2.3. From

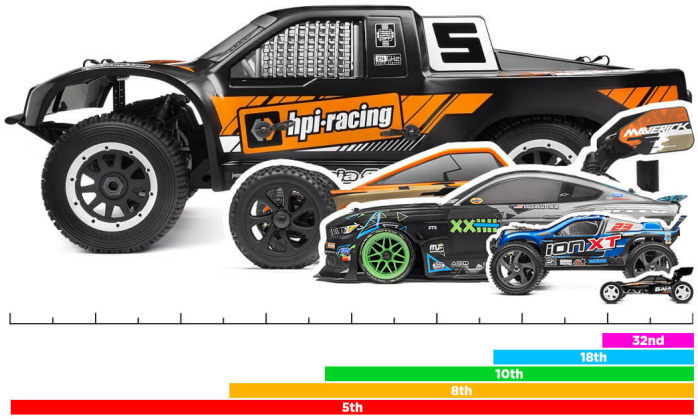


Figure 2.1: RC car scales

this top-view representation, the operating mechanism can be well understood. Two

Figure 2.2: RC car

Table 2.1: RC car components

Mechanical parameters:	Size: 34.5× 30.5×16.5cm^3 Weight:1.530kg
Power:	7.4V 1600mAh Li-po battery Usage: ∼15min
Actuation:	RC Radio receiver & integrated 60A ESC 2x Brushed DC Motor RC servo 2.2kg
Radio Transmitter:	Frequency:2.4GHz Control distance:80m Battery 3xAA
Other parts:	Car body shell Car chassis 4x Rubber wheel 6x Shock absorber

DC motors act on the same gear (white), which is rigidly connected to the propeller shaft. It transfers the motion to both the differentials, which in turn actuate the front and rear axle.

The steering action is performed by means of an RC servo which activates a kinematic chain. The latter is reconstructed in 3D in figure 2.4 for the sake of clarity. The white gear, coaxial to the servo's shaft, rotates together with a small arm to which is rigidly linked. The kinematic chain transfers the motion to a disk with axis of rotation along the vertical direction (with respect to the car's reference frame).The disk's rotation moves two arms (truncated in the figure), connected respectively to the front left and front right wheel, and the steering is completed. For example, when the first arm is pushed backwards, the disk rotates clockwise. The back of the front left wheel is pushed far from the vehicle, while the back of the right wheel is pulled. The vehicle turns right.

Both the DC motors and the RC servo are controlled by the RC receiver. This device has an essential role in the architecture of the car. First, it receives the signal from the controller and then it transforms it into a proper set of signals to be sent to the motors' couple and to the servo. All the power provided by the Li-Po battery is managed by

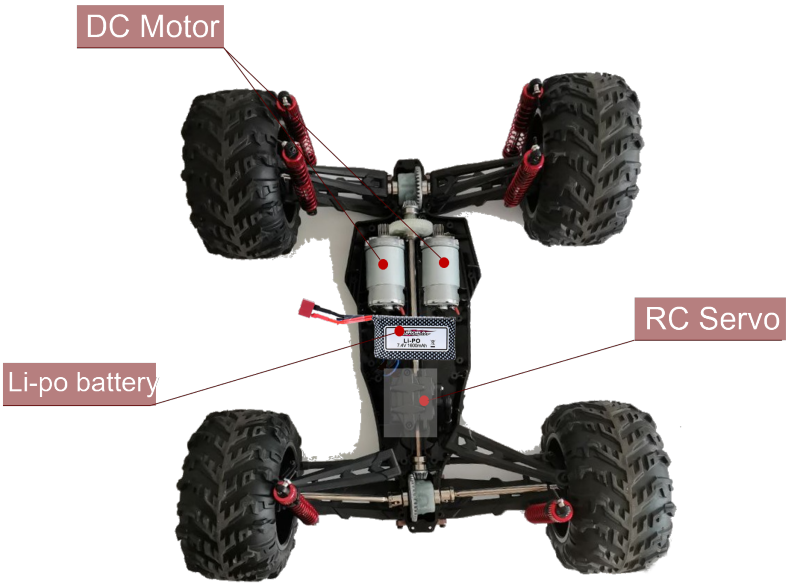


Figure 2.3: RC car without top cover

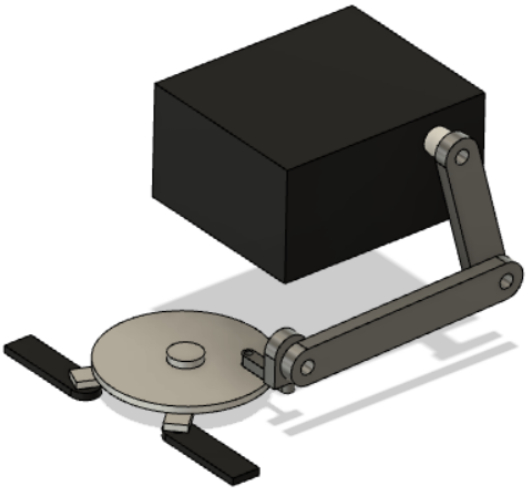


Figure 2.4: Steering mechanism

this device, which also has the duty to distribute it to the actuators.
It must be noticed that all the electronics needed to make the vehicle work is right inside the receiver. To better understand all the implications of this architecture, a deeper analysis on the working principle of the actuators is due.

2.1.1 DC Motor

DC motors exploit electromagnetism to convert electric power into mechanical power. The basic principle behind this transformation is shown in figure 2.5. When a voltage is applied to the sides of a conductor coil, it acts as a resistance and lets the current flow. This flow produces a magnetic field directed according to the screw rule, with the four fingers aligned with the current flow. Placing the coil inside two permanent

magnets, as in figure, the induced magnetic field makes the coil rotate to align to the external field. At this stage, if the polarity at the sides of the coil was switched, the coil would keep rotating accordingly to the new configuration.

Since switching the polarity would imply repetitively crossing the two wires connecting

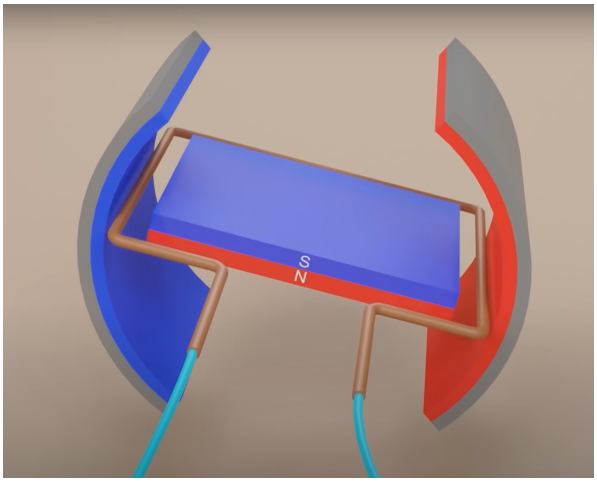


Figure 2.5: Working principle of a DC motor

the coil to the voltage source, the coil is connected to a commutator, a device with two isolated parts which rotate together with the coil and are connected to the power supply by means of sliding contacts called *brushes*. The addition of a higher number of coils, together with a proper commutator, facilitates the motion (figure 2.6).

The description above corresponds to a precise type of motor: the brushed DC motor. It is easy to understand that raising the voltage of the power supply, the motor's speed increases.

The Voltage equation of a DC motor can be expressed as:

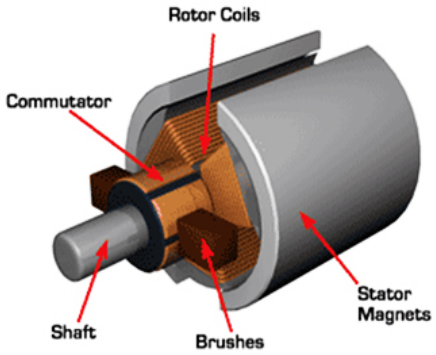


Figure 2.6: Brushed DC motor structure

$$V = E_b + I_a R_a + V_{br} \tag{2.1}$$

where V is the supply voltage, E_b is the back EMF, $I_a R_a$ is the armature resistance drop and V_{br} is the brush drop.

When the motor is switched on, the initial back EMF is 0. Neglecting the last term, the armature resistance drop must equal the supply voltage and since R_a is very small,

the initial I_a must be very high.
In the case of the RC car under exam, as it has been already stated, currents and voltages are managed by the receiver. The latter features an integrated Electronic Speed Controller (ESC) which can provide up to 60A, even if such a high current is only needed to start the motors.
It must be noticed that DC motors can also be brushless. Although this type of motors is used for a wide variety of applications, the illustration of them is outside the scope of this thesis.

2.1.2 RC servo

Servos are devices consisting of a small DC motor driving a train of reduction gears. The output shaft is connected to a potentiometer, which *measures* its position. This measurement is compared to the reference set by the radio control and the computed error is corrected by a sort of proportional controller. The system works in closed-loop [42].

Opening the box in which the RC servo is contained, the gearbox and the internal

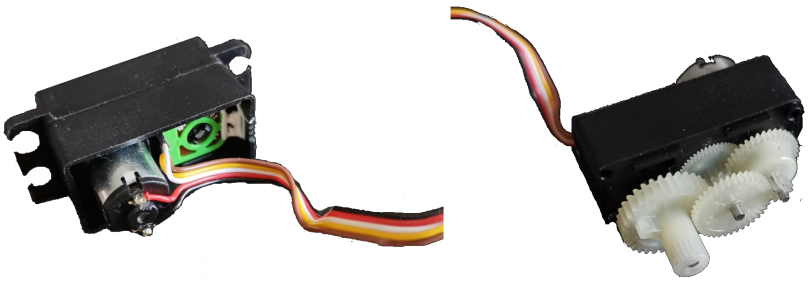


Figure 2.7: RC servo structure

circuitry appear as in figure 2.7. It can be seen that a 5-wire cable exits the servo, making the electrical behaviour of this device straightforward:

- Three wires are connected to the potentiometer: one is the ground reference, one is the supply voltage and the third (usually the central one) is the potentiometer output
- Two wires are connected to the DC motor: as seen in the previous subsection, they provide the supply voltage to the motor, regulating the speed and the verse of rotation.

This configuration seems to lack one element: the electronic part which implements the proportional controller. Actually, this task is entrusted to the radio receiver, which has a pivotal role also in this case.

2.2 Architecture of the scaled Autonomous Vehicle

The architecture of an autonomous vehicle, as seen in chapter 1, needs some additional parts with respect to a traditional vehicle. Proper hardware must be added to perform perception and control tasks which are normally handled by the human user and some

modification is also needed to make the actuation system compatible. All the hardware composing the scaled autonomous vehicle is collected in table 2.2.

Looking at the table, the first thing which stands out is the absence of two key ele-

Table 2.2: Scaled AV components

Mechanical parameters:	Size: 34.5× 30.5×28.9cm^3 Weight:2.00kg
Power:	7.4V 1600mAh Li-po battery Usage: ∼15min 5200 mAh Power bank
Perception:	Okdo 5MP camera LSM9DS1 9-DoF IMU
Actuation:	L98N Dual H-Bridge motor driver 1x Brushed DC Motor Micro servo
Single Board Computer:	Raspberry Pi 4 model B
Other parts:	Car chassis 4x Rubber wheel 6x Shock absorber Solderless Breadboard 8,3mm, 54,5mm x 83,5mm Camera support

ments of the RC car: the radio transmitter and receiver. The place of the transmitter, along with its user, is taken by the perception hardware, made by a camera and an Inertial Measurement Unit (IMU), while it can be simplistically said that the role of the receiver is conducted by a Raspberry Pi 4 model B.

As seen in the previous section, both the DC motors and the RC servo were controlled by the receiver. Anyway, the Raspberry Pi cannot directly provide the right input signal to these devices. Hence a L298N motor driver is added and the RC servo is substituted by a micro servo.

Finally, a battery to power the Raspberry Pi, a breadboard to ease the electric connections and a support for the camera are added. The latter is the main responsible for the variation of size of the autonomous model, while it is clear that the increase in weight is due to the several additional components.

A picture of the complete scaled AV is represented in figure 2.8.

A detailed description of the hardware employed for perception and actuation will be given in the next chapter, while the reasons behind the choice of the Raspberry Pi 4B are explained in the next subsection.

2.2.1 Choice of board

The most important part of the hardware platform is the electronic board, which acts as a bridge between sensors and actuators. This device must also be able to perform a huge number of computations in real-time, so that the control action is performed

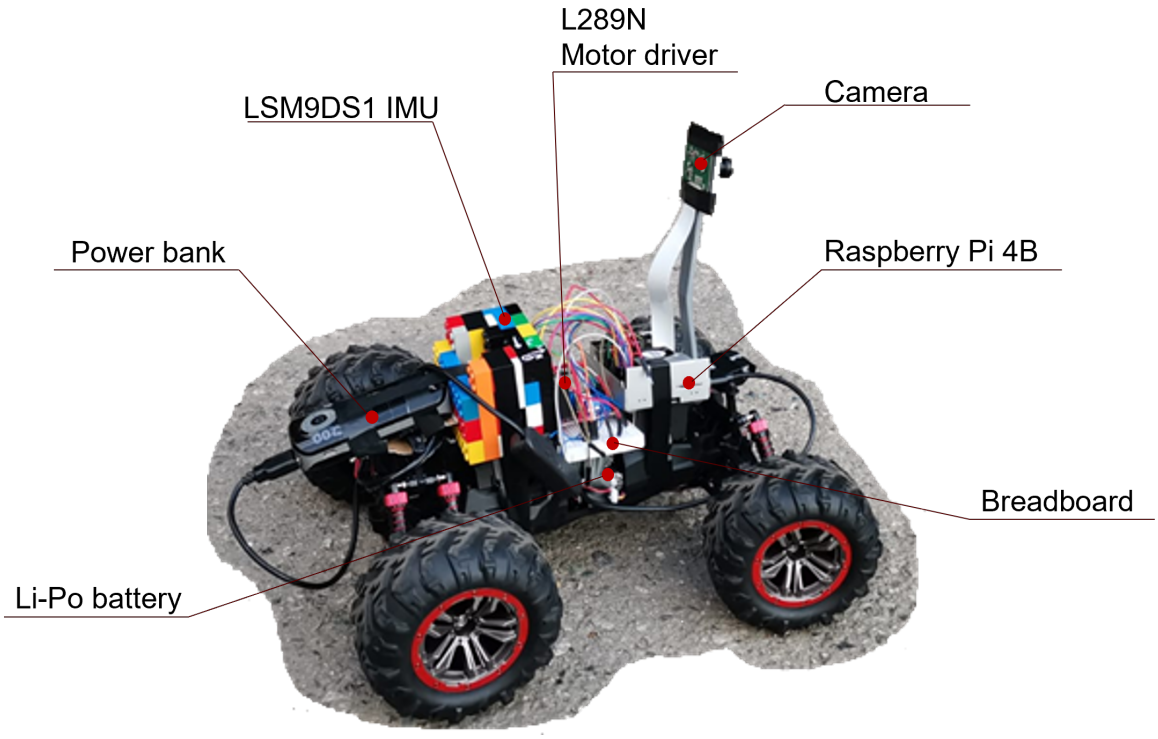


Figure 2.8: Scaled Autonomous Vehicle

as required. Although there exist several devices with these features, the choice of the proper board must be done following the same requirements which lead the whole project: dimension and cost.
Three boards have been selected, which respect the dimensional constraint and can easily manage a wide variety of sensor:

- Nvidia Jetson Nano with 128-core dedicated GPU, 4 GB of shared LPDDR4 RAM and a Quad-core ARM CPU(1.43GHz)
- Raspberry Pi 4 model B, with 8GB of LPDDR4 RAM and a Quad core ARM CPU (1.5GHZ)
- Arduino UNO, with 2KB RAM and a ATmega328 16MHz microcontroller.

Among these, the first two are actually Single Board Computers (SBCs), while the third is just a programmable board based on a microcontroller. The necessity of elaborating images and working in real time at high frequencies, makes the Arduino UNO incompatible for the tasks of this project.
The characteristics of the Raspberry Pi 4B and the Nvidia Jetson Nano are very similar, except for the presence of a dedicated GPU in the Nvidia board. Nevertheless, the price for the Nvidia Jetson Nano almost doubles the one of the Raspberry Pi 4B.
The turning point in the choice is the presence of a MATLAB and Simulink Support Package for Raspberry Pi, which can be a great boost in a project from scratch.
One of the most interesting features of Raspberry Pi is the presence of a 40-pin General Purpose Input Output (GPIO) header which ease the connection of the board with input and output devices [43]. A description of this pins is represented in figure 2.9 for possible future purposes.

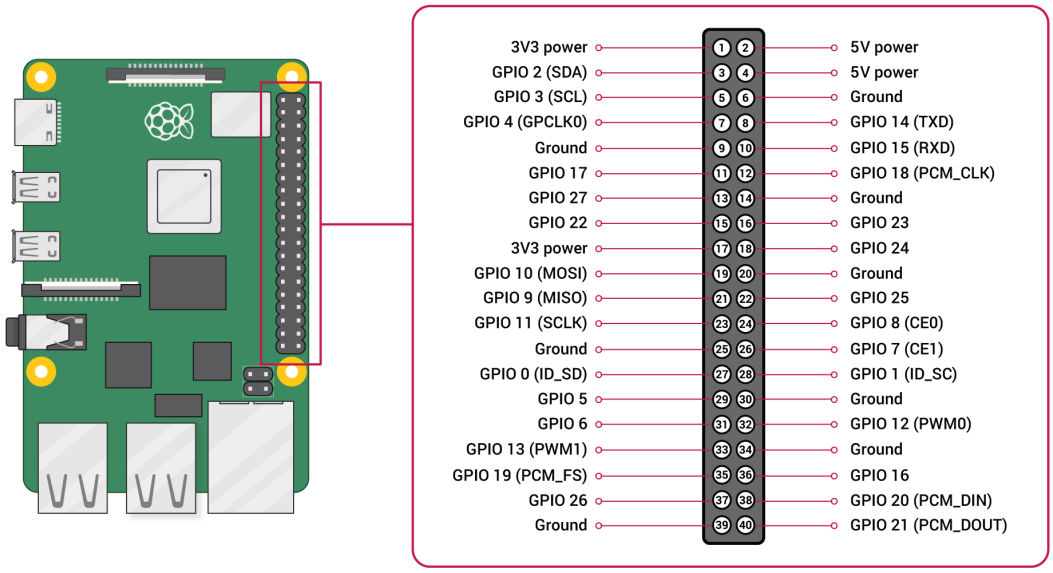


Figure 2.9: Raspberry Pi 4B GPIO pinout [43]

2.2.2 Working Principle in steps

The transformation from RC car to AV can be better described following the relative working principle, which guides the choice of the most suitable hardware.
At the very beginning (figure 2.10), it is the human user which uses its capabilities to drive the vehicle by means of the remote control. It transmits radio signals to the on board receiver, which translate it in suitable signals for the actuators. The couple of DC motors is driven by the ESC integrated in the receiver, while the RC servo is directly controlled.
An intermediate step is added before reaching full automation (figure 2.11.) The vehicle

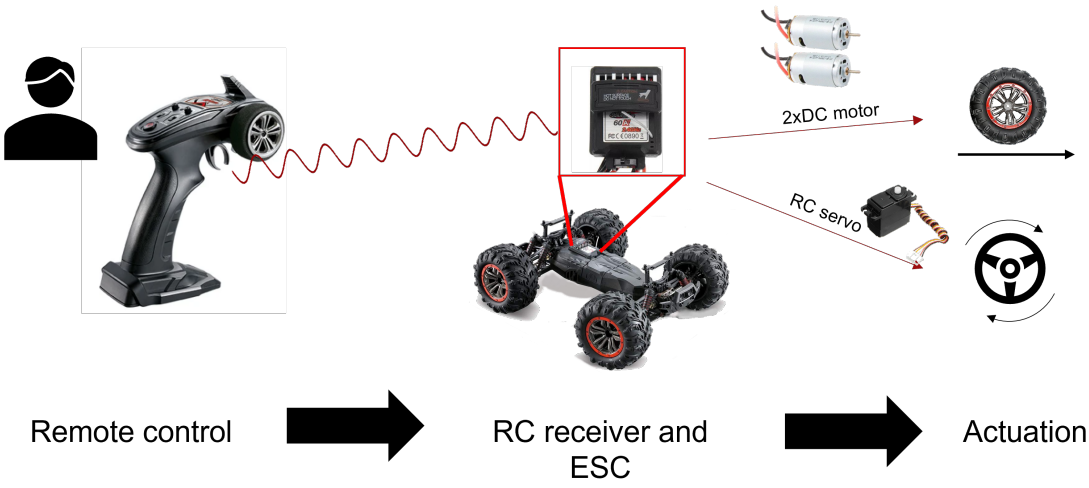


Figure 2.10: Working principle of RC car

is equipped with the Raspberry Pi 4B, which is connected via Wi-Fi to a PC running

MATLAB and/or Simulink. MATLAB/Simulink Support Package for Raspberry Pi allows to control the General Purpose Input Output (GPIO) interface of Raspberry Pi, sending commands to the actuators. In order to save the RC structure for future modifications, just one of the two available DC motors is used. A L298N motor driver gets the input from the SBC and provides the motor with the required voltage. The steering is performed by means of a micro servo which takes the input signal directly from the Raspberry Pi.

In the last step (figure 2.12) no human action is required at all. The vehicle is let

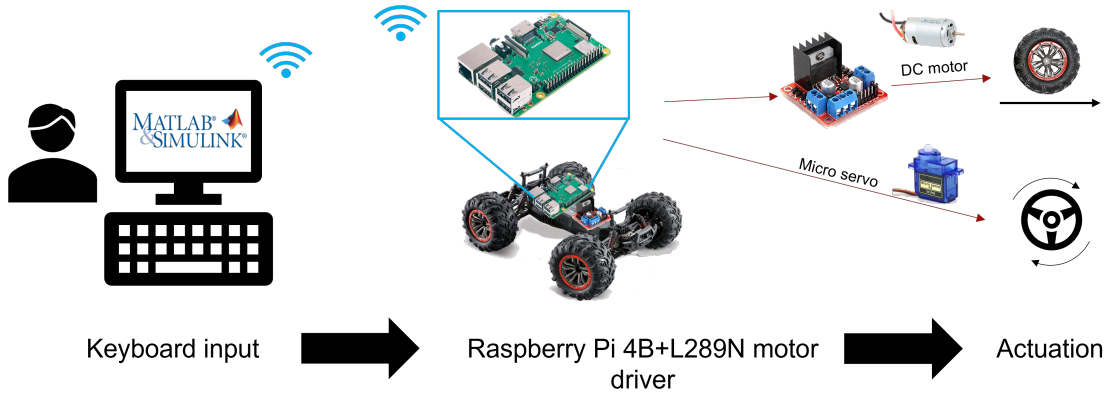


Figure 2.11: Working principle of car guided by keyboard

drive alone on a road with two lane marks. Thanks to the presence of the camera, a lane detection algorithm is implemented on the Raspberry Pi, while the IMU provides information about the vehicle motion. Sensor data are sent to the SBC which uses a dynamic control strategy to select the proper commands for the actuators.

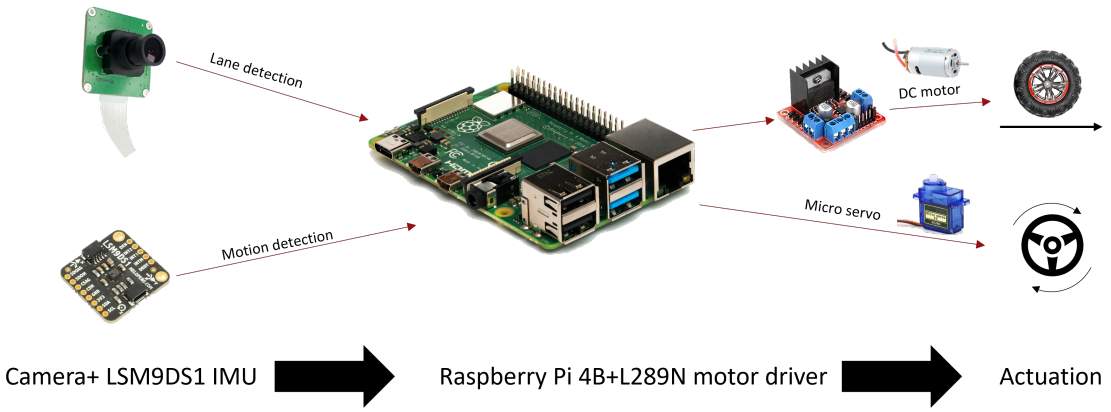


Figure 2.12: Working principle of autonomous car

Chapter 3

Perception

The first step of automation is certainly the sensing of external environment, followed by the capability of extracting useful data for motion planning.
There are several sensors that can perform this task, each with its pros and cons, but for this project two which can be easily integrated with Raspberry Pi are chosen. They are a monocamera and a 9-DOF IMU.
The main advantage of a vision system is that it can be used to estimate distance from many different objects with a low-cost sensor. In terms of functionalities, it is the closest device to the human eye and, in the same way, the information it can provide relies on a limited field of view. In this case, it was chosen to use computer vision for lane detection, so that the vehicle can perceive the characteristics of the road and its position with respect to it.
There are basically two types of sensors available for motion detection: GPS and IMU. In the majority of full-scale AVs, but also in common smartphones, both sensors are present. Anyway, in this work it is very unlikely that a sensor has to be used for localization purposes, considering that the vehicle will be mainly used indoor or in a delimited outdoor environment. Hence, an IMU seems to be sufficient for the present purposes of this work.
It must be noticed that the selected sensors work at very different frequencies. Common cameras work at 30 or 60fps, but the actual frequency that must be considered depends on the time needed for the vision algorithm to extract the desired data.
On the other hand, IMUs can normally detect signals at frequencies in the order of hundreds of kHz.
This information on working frequencies is very important because, as seen in Chapter 1, data from sensors with different acquisition rates can be fused together to produce meaningful estimation of the variables under control.

3.1 Vision sensor

The vision sensor chosen for this project is a 5MP Okdo camera compatible with the Raspberry Pi series, which has almost the same characteristics as Pi camera v1 plus a Fisheye lens to provide a wider field of view. Technical specifications of the camera are collected in table 3.1.

The sensor is connected to Raspberry Pi by means of a flex cable about 15cm long. The existence of this cable represents a small issue, as the sensor cannot be placed too

Table 3.1: Camera technical specifications

Sensor type	OmniVision OV5647 Color CMOS QSXGA
Sensor size	3.67 x 2.74 mm (1/4 inch)
Resolution	5 million pixels
Field of view	Fov(D) = 90° Fov(H) = 74°
Pixel Count(Still Picture Resolution)	2592 x 1944
Focal Length	2.8 mm
Focal Distance	Adjustable
F(N) /Aperture	2.2
Pixel Size	1.4 x 1.4 um
Video	1080p at 30 fps 720p at 60 fps
Board Size	39 x 39 mm (not including flex cable)
TV DISTORTION	<-17%
CRA	10°
Relative Illumination	52%
Minimum Object Distance (M.O.D)	0.1 meter
Element	4G+IR
Lens Diameter	M12
Lens Seat Spacing	22 mm
Mounting Holes	4x D=2.20 mm

far from the board, thus limiting the field of view. In this context, the presence of a Fisheye lens is a significant upgrade with respect to normal cameras.
In order to fully exploit the cable length, a support for the camera is designed to be 3D printed and inserted into the front support of the car body shell (which is no longer present). The support, represented in figure 3.1, is 18.1cm long, but at least 4 of these will be placed under Raspberry Pi level. Moreover, the final part of the arm is bent forward, such that the camera field of view does not start too far from the vehicle front. A thickness of 4mm is established for the part which develops along the vertical direction, accounting for the stiffness of PLA, the material chosen for FDM.
It must be remarked that the camera should provide information about the road, such as curvature and lateral deviation. Thus, if the fisheye lens helps the estimation providing a wider field of view, it also produces distorted images that could lead to misleading information. It is then important to correct image distortion, so that available images can be used in the best way.

3.1.1 Camera Calibration

Camera calibration is a procedure which allows to estimate important parameters which can be used not only to correct lens distortion, but also to measure the size of an object

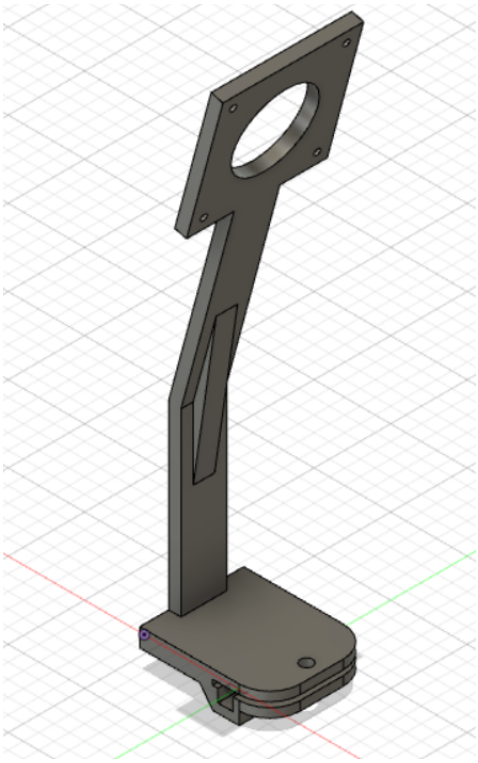


Figure 3.1: 3D design of camera support

in world units or determine the location of the camera in the scene[34].
The tool chosen to perform the calibration of the camera mounted on the vehicle is MATLAB *Camera Calibrator App*. It is based on two well-known camera models:

- Pinhole camera model: for cameras without lenses
- Fisheye camera model: for cameras with a field of view of up to 195°.

These two models are in some sense complementary, because the first is the one which provides camera parameters, while the second accounts for lens distortion. The combination of the two allows to estimate object distance in the real world basing on the distance between image points collected using a fisheye camera.

According to pinhole model, a point (X, Y, Z) in space is projected onto an image

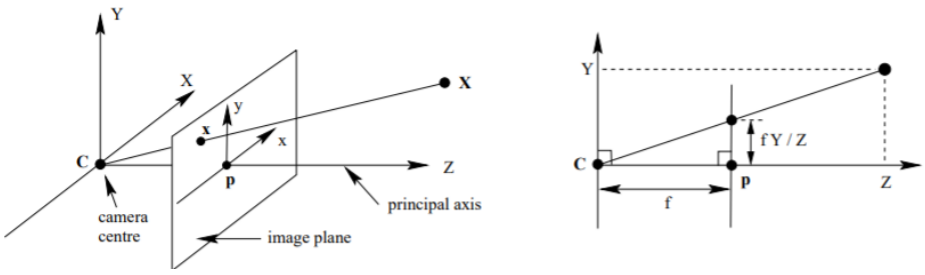


Figure 3.2: Pinhole camera model [44]

plane placed at focal distance f with respect to the camera centre. The axis perpendicular to the image plane which crosses the camera centre is called *principal axis*, while

the point where this axis meets the image plane, is called *principal point* [44]. With reference to figure 3.2:

$$(X, Y, Z)^T \longrightarrow (\frac{fX}{Z}, \frac{fY}{Z})^T \tag{3.1}$$

Using homogeneous coordinates, the mapping can be written as

$$\begin{bmatrix} fX \\ fY \\ Z \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_{cam} \\ Y_{cam} \\ Z_{cam} \\ 1 \end{bmatrix}. \tag{3.2}$$

This mapping is valid when the origin of coordinates in the image plane is the principal point. A more general formulation is then:

$$\begin{bmatrix} fX + Zp_x \\ fY + Zp_y \\ Z \end{bmatrix} = \underbrace{\begin{bmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\text{K}} \begin{bmatrix} X_{cam} \\ Y_{cam} \\ Z_{cam} \\ 1 \end{bmatrix} \tag{3.3}$$

Matrix K is called *camera calibration matrix* or *Intrinsic matrix* and maps the camera coordinates into the image plane.
The location of the camera in the 3-D scene can be described by means of a rototranslation:

$$\boldsymbol{X}_{cam} = \begin{bmatrix} R & t \end{bmatrix} \boldsymbol{X}. \tag{3.4}$$

where the rototranslation matrix is also called *extrinsic matrix*.
Putting everything together, the formula for general mapping of pinhole camera in world coordinate frame $\boldsymbol{x}$ is defined by

$$\boldsymbol{x} = K[R\ t]\boldsymbol{X} = P\boldsymbol{X} \tag{3.5}$$

with P being the camera matrix.
Radial distortion is the phenomenon which occurs when light rays bend more near the edges of a lens then they do at its optical center. Two types of radial distortion exist: pincushion and barrel (figure 3.3). In both cases it is possible to describe the distorted points using three radial distortion coefficients as in

$$\begin{aligned} x_{distorted} &= x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \\ y_{distorted} &= y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \end{aligned} \tag{3.6}$$

with

$$r^2 = x^2 + y^2$$

Camera calibrator App allows to get both the radial distortion coefficients of the lens and the camera matrix, basing on pictures of a calibration checkerboard whose squares' dimension is known. It is clear that for a sensor with adjustable focal distance, the calibration must be performed using the same distance that is expected to be used in the real application.
A working example of the application is shown in figure 3.4. Twelve pictures of the

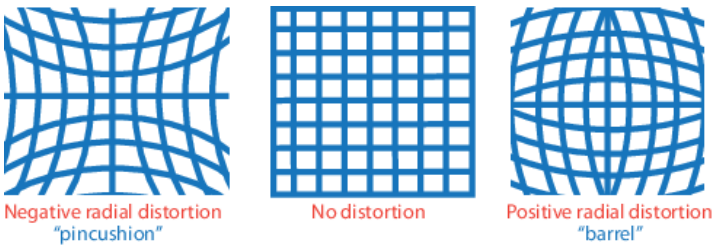


Figure 3.3: Radial distortion [34]

checkerboard are taken from different positions and imported into the software. After setting the real size of the squares, the camera parameters are identified and used to estimate the relative position from which the pictures were taken, both in a camera-centric and in a pattern-centric view.

Figure 3.5 represents one of the pictures before and after distortion correction.

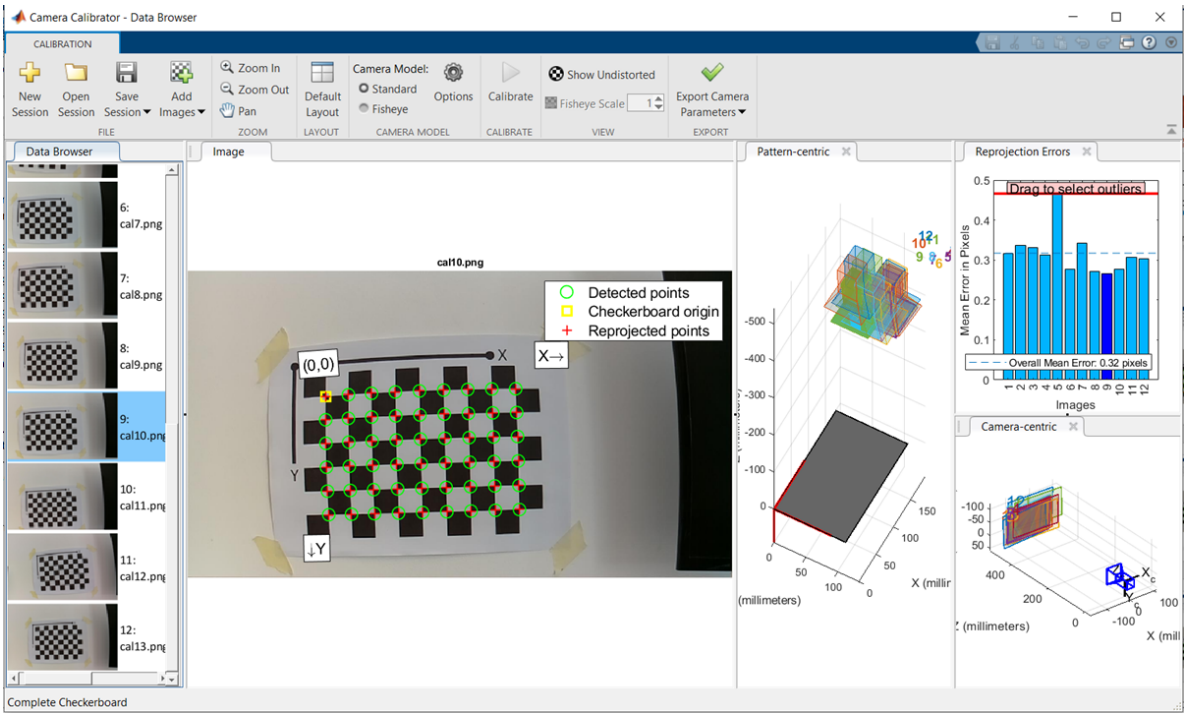


Figure 3.4: Camera Calibrator App

3.1.2 Lane Detection Algorithm

The aim of the lane detection algorithm is the localization of the road and the determination of the relative position between the vehicle and the road.

Each time a frame is taken during vehicle motion it must be analysed undergoing a series of transformation which allow to estimate the desired data.

First, the Region of Interest (ROI) for lane detection must be detected, excluding from the analysis those parts of landscape which are inevitably included in the frame. Due to the imaging perspective effect, the lanes will not appear parallel. It is then useful to perform a perspective transformation to produce a top view image of the road. Such a transformation can be done knowing camera intrinsic parameters such as the focal

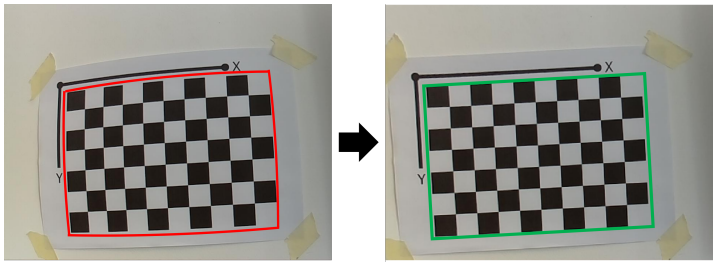


Figure 3.5: Radial distortion correction

length, the position of the principal point and the image size, which are previously obtained by calibration. Moreover, camera's mounting height from the ground and its inclination with respect to the vertical axis must be known. The top-view obtained using these parameters is called *Bird's Eye View* (BEV). The latest step before lane detection is the binarization of the BEV image, which consists in setting a threshold under which all pixels are considered white. Vice versa, all the remaining pixels are painted black. To ensure that lane-like features are correctly spot, the binarization is followed by Parallel Feature Extraction, i.e. the brightness of each pixel is compared to its horizontal left and right neighbors at a chosen distance. In this way, brighter marks on the road are simply neglected.
An example of the above-described procedure is depicted in figure 3.6.

Figure 3.6: ROI, BEV and image binarization

At this stage everything is ready to fit the lane markers. It must be remarked that there is no conventional function which allows to fit two or more curved lines in the same pixel matrix. If lines were straight, it would be sufficient to find the columns of the matrix with the highest number of white pixels, but for curvy roads, even with small curvatures, things are not that simple.
The function which allows to perform the lane detection is *findParabolicLaneBoundaries* [45]. It takes as input the position of the white pixels, the estimated width of lane markers and the maximum number of lane markers to detect, and gives as output an array of objects which hold the three coefficients of a parabola fitting the lane markers. By means of these coefficients, it is possible to plot the estimated lines and overlap them to the original image through an inverse perspective transformation. The result is shown in figure 3.7.

Aside from the graphical representation, the knowledge of the parabola coefficients

Figure 3.7: Lane detection and inverse perspective transformation

and their location inside the 2-D image is nothing less than what is needed to estimate several important variables:

- Road center line
- Road curvature
- Lateral deviation of the vehicle from road center
- Heading error of the vehicle.

The computation of road center line and lateral deviation is straightforward. As far as concerns the road curvature, the function *polynomialCurveCurvature* is used.
Heading error is estimated through simple geometrical considerations. According to figure 3.8, when in the BEV image the road points left, it means that the vehicle is pointing right. Thus, the heading error is computed basing on the angular coefficient of the segment which connects the point in which the vehicle is located, to a point on the center line set at a given distance. If the point is not too far and the road curvature is not too wide, this segment will likely find itself on the tangent to the center line.
An example of the output of the lane detection algorithm is plotted in figure 3.9.

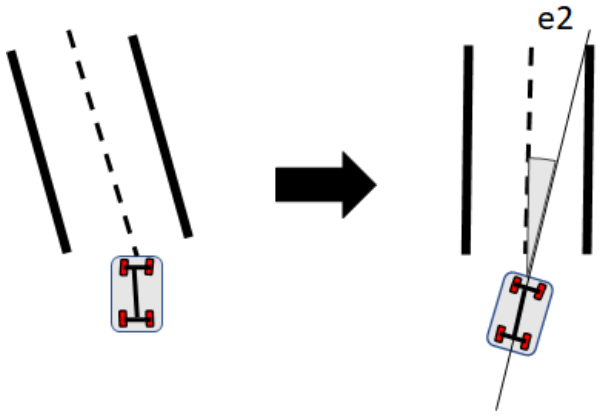


Figure 3.8: Transition from vehicle point of view to external point of view

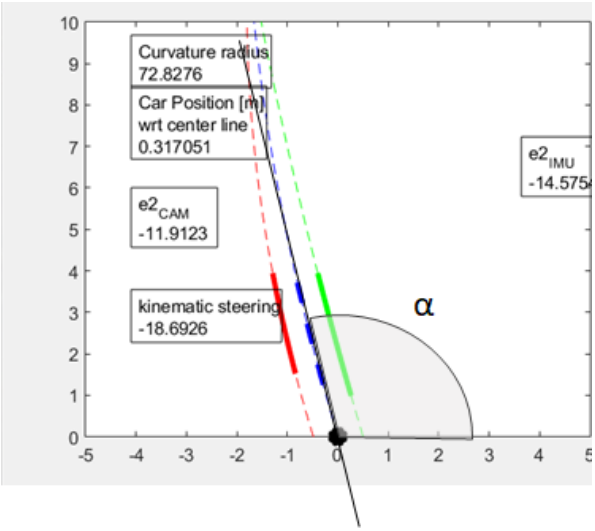


Figure 3.9: Lane detection output

3.2 Motion Detection-Inertial Measurement Unit

When dealing with the choice of an IMU there are essentially two features to account for: price (which comes along with sensor accuracy) and number of variables that can be measured. Choosing a high quality IMU would completely jeopardise every effort made to build a low-cost platform, while inside the same price range it does not make a big difference whether the sensor has six or nine DOF. For this reason, a LSM9DS1 was selected. It is a 9-DOF MEMS module with I^2C interface.
MATLAB Support Package for Raspberry Pi owns several functions to interface with IMUs, which are treated as MATLAB objects. Nevertheless, these objects cannot be used inside Simulink. The only exception is represented by LSM9DS1, for which an ad-hoc block was developed in Simulink environment, to return acceleration normalized with respect to g, angular velocity in degree and magnetic field in μT (figure 3.10).

The sensor is provided with a 3D accelerometer, a 3D gyroscope and a 3D magnetometer, with a linear acceleration full scale of $\pm 2g/\pm 4g/\pm 8/\pm 16$ g, a magnetic field full scale of $\pm 4/\pm 8/\pm 12/\pm 16$ Gauss and an angular rate of $\pm 245/\pm 500/\pm 2000$ dps. It includes a I2C serial bus interface supporting standard and fast mode (100 kHz and

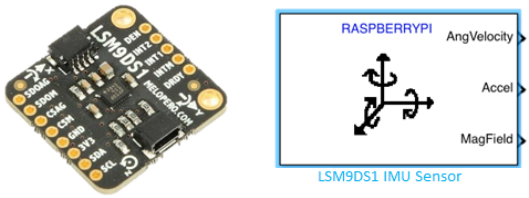


Figure 3.10: LSM9DS1 sensor and Simulink block

400 kHz) and an SPI serial standard interface. The pinout present itself like this:

- 3V3: Input power pin
- GND: Ground pin
- SDA: I2C SDA/ SPI MOSI, compatible only with 3.3V logic
- SCL: I2C SCL/SPI SCLK, compatible only with 3.3V logic
- INT1: Accelerometer/Gyroscope Interrupt pin
- CSAG: SPI Chip Select for Accelerometer/Gyroscope
- CSM: SPI Chip Select for Magnetometer
- SDOAG: SPI Data Output for the Accelerometer and Gyroscope, SPI MISO
- SDOM: SPI Data Output for the Magnetometer, SPI MISO

For I^2C use, only the first four pin must be connected to Raspberry Pi.

3.2.1 IMU Calibration

Before starting to collect meaningful data from an IMU, a calibration procedure is due. It is important to know that the errors which commonly affect these sensors are [46] [47]:

- Gyroscope bias: keeping the IMU steady, angular velocities measured on the three axes oscillate around a non-zero value
- Accelerometer scale factor and bias: the error in accelerometer data is also proportional to the value of acceleration
- Magnetometer hard iron and soft iron distortion: materials in the sensor or close to the sensor can add a magnetic field or distort earth magnetic field

In order to perform a clean calibration procedure, the sensor must be placed inside a suitable support with three adjacent faces perpendicular to each other. The reason for this geometry will be soon made clear.
The initial idea for the support was to create a module which made possible to easily insert and extract the IMU without ruining the structure. The result of this concept is pictured in figure 3.11: a Lego house which keeps the IMU in its place, leaving the pin

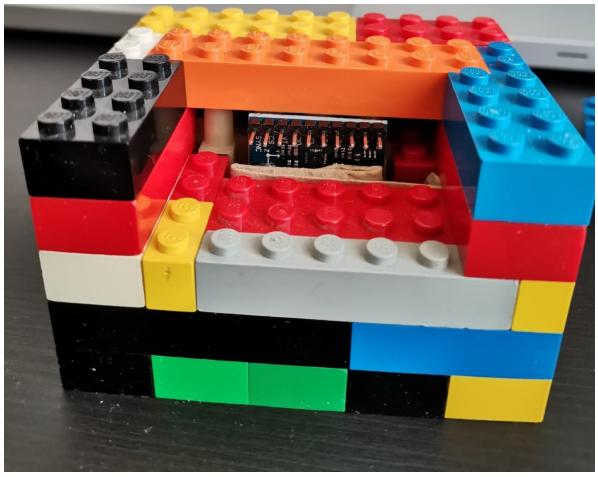


Figure 3.11: IMU support

header available to make connections.

Gyroscope calibration is an easy task, considering that the bias which affects it is constant over relatively small time periods. It is sufficient to keep the sensor steady and measure the value of angular velocity and compute its mean value on each of the three axes. The values obtained are then stored inside a vector G, so the actual values of angular velocity can be found using:

$$\boldsymbol{\omega} = \boldsymbol{\omega_{meas}} - \boldsymbol{G}$$

Calibration results are shown in figure 3.12.

As for the accelerometer, the actual value of the acceleration vector can be expressed as

$$\boldsymbol{a} = \boldsymbol{M} \cdot \boldsymbol{a}_{meas} + \boldsymbol{O} \tag{3.7}$$

where M is a diagonal matrix.
It is implicitly assumed that errors on each axis are independent, otherwise off-diagonal elements would be non-zero. This is not always the case, but potential correlations are neglected for simplicity.
The calibration is performed collecting measurements in three different positions for each axis:

- Axis pointing upward
- Axis pointing downward
- Axis perpendicular to gravity

For each measurement, a mean value over a small time period is considered as the actual measured value. In this way the measured value can be easily compared with the real value of acceleration, which is clearly $\pm g$ or 0.
Figure 3.13 shows that, using data normalized with respect to g, it is possible to detect a line, whose angular coefficient and offset represent the elements of $\boldsymbol{M}$ and $\boldsymbol{O}$.

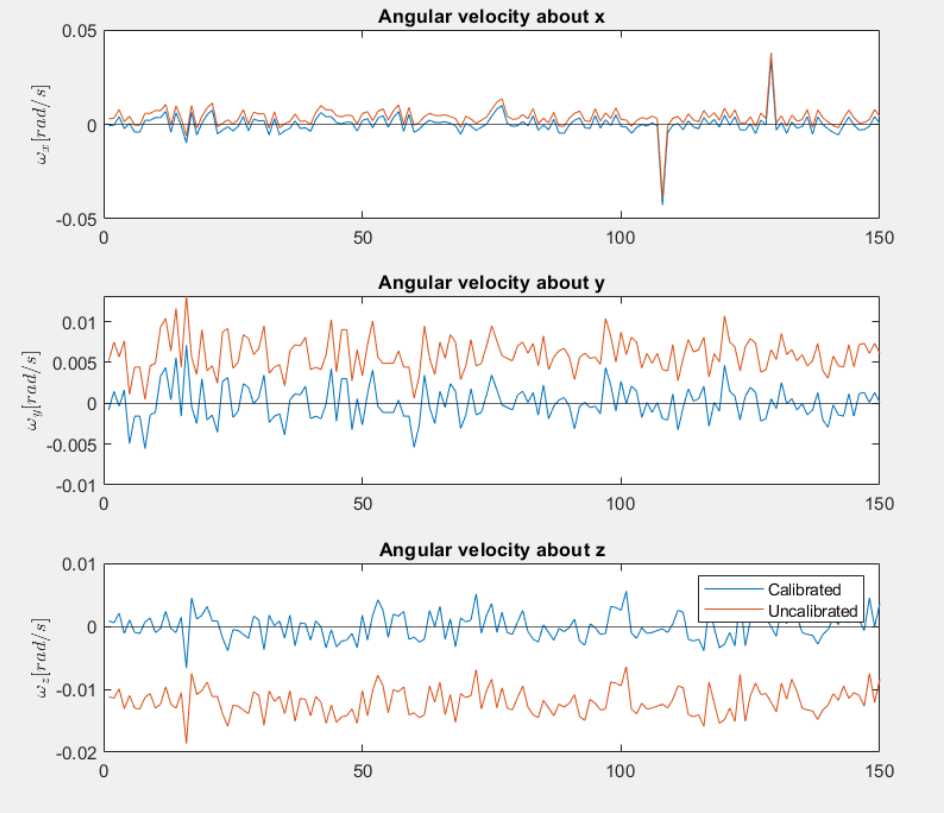


Figure 3.12: Gyroscope calibration offset correction

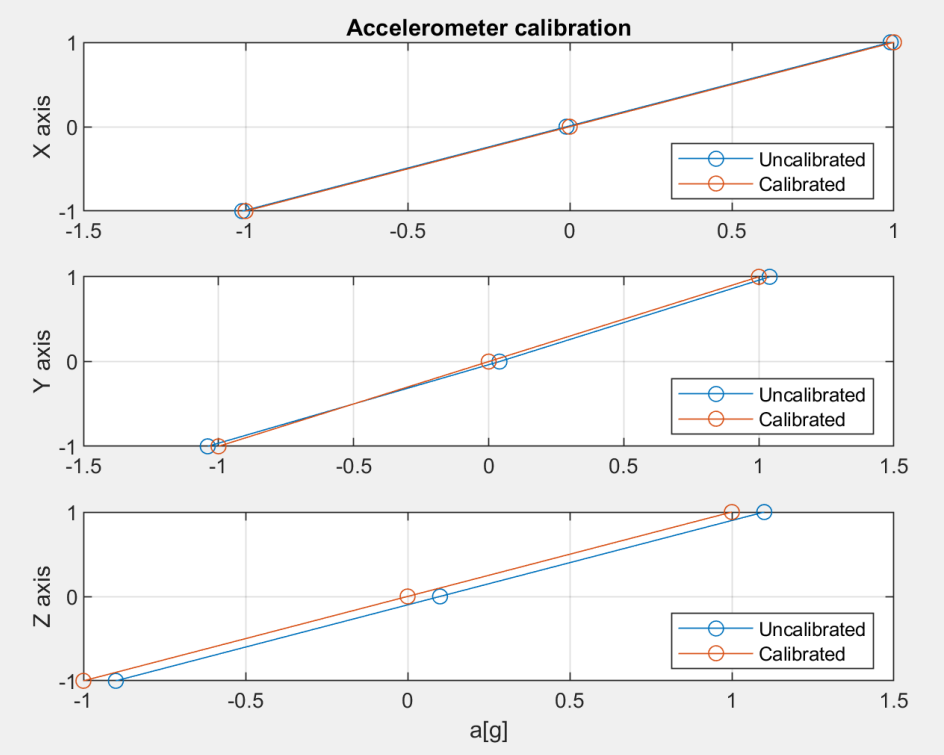


Figure 3.13: Accelerometer calibration

The major responsible for magnetometer errors is hard iron distortion, which is produced by materials that exhibit a constant, additive field to the earth's magnetic field. In order to compensate for this effect, it is necessary to detect this offset.
Calibration consists in imposing a 360° rotation about each of the three axis. The axis under exam points toward gravity. Once the offset is detected, it is possible to remove it as in figure 3.14.

Figure 3.14: Correction of hard iron offset

3.2.2 IMU Measurements and Integration

By means of the IMU, two variables of great interest can be computed: the heading error e_2 and the longitudinal velocity V_x.
From the vehicle model it can be stated that

$$\dot{e}_2 = \dot{\psi} - k \cdot V_x \tag{3.8}$$

where k is the road curvature. If the second term at the right side of the equation was not considered, the error could only be computed for a straight road.
The longitudinal velocity can be directly obtained through integration of the acceleration component direct towards the longitudinal axis. Due to vehicle shape it is likely that the IMU house will not be perfectly parallel to ground, but slightly tilted forward or backwards.
Defining a fixed reference frame F_1 with the longitudinal axis parallel to ground and a mobile reference frame F_2 rotated of an angle ϕ as in figure 3.15, a vector v_1 seen in the fixed frame can be computed from the knowledge of the same vector v_2 seen from the mobile frame as in

$$v_1 = Rv_2$$

where

$$R(\phi) = \begin{bmatrix} cos(\phi) & 0 & sin(\phi) \\ 0 & 1 & 0 \\ -sin(\phi) & 0 & cos(\phi) \end{bmatrix}.$$

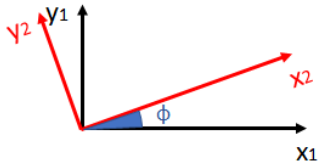


Figure 3.15: Fixed and mobile reference frame

In this way also the longitudinal acceleration a_x can be computed. Both this data and $\dot{e}_2$ must be integrated to get the desired variables.
Simulink provides a continuous-time and a discrete-time integration block, while integration in MATLAB can be performed using different functions. The one chosen for this case is *trapz*, which implements a trapezoidal integration.
It must be considered that integration of an offset error produces a linear error and integration of a linear error produces a square error. A meticulous calibration is therefore crucial before performing integration.
Figure 3.16 shows the integration of two acceleration signals; in the first case (left) the

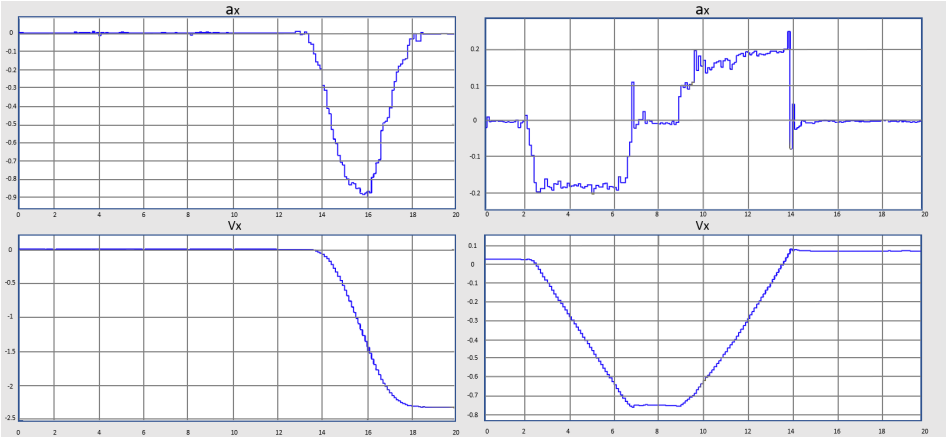


Figure 3.16: Computation of longitudinal velocity using integration

sensor is rotated back and forth towards gravity, while in the second case the operation is repeated twice in two different directions. The obtained velocity is showed in the plots below. From a qualitative point of view, the integrated signal respects the expectations. However, especially looking at the right plot, it can be observed that the acceleration signal is noisy, and the final value of the velocity is different from 0.
The presence of noise is mainly due to the high frequency of the signal. One way to obtain cleaner data is to implement a Low Pass Filter (LPF), which cuts out high frequency components. As can be seen in figure 3.17, where the higher plot represents the filtered signal, the presence of the filter does not modify the amplitude but produces a delay, which can be very damaging for the purpose to which the sensor was chosen. Moreover, this noise correction does not influence the integration neither positively nor negatively. In fact, if in place of a straight line there are two curves whose subtended area has the same value, the integration of the two will lead to the same result. On the basis of this consideration, there is no point in correcting high frequency noise.
When it comes to the problem of obtaining non-zero velocity, the first imaginable solution seems to be adding a saturation block which cancels values close to zero. If this

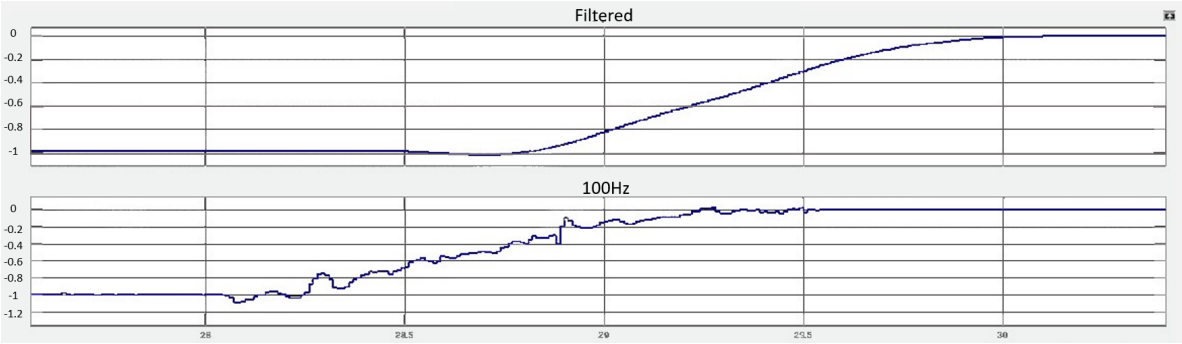


Figure 3.17: Application of Low-Pass Filter

block was added on acceleration data before integration, this would produce high errors in cases where the acceleration is actually occurring. On the other hand, saturating the computed velocity would apparently solve the problem for null velocities without providing any beneficial effect, in cases where the velocity is different from zero. Under normal operating conditions, this will never be the case. So also the saturation proves to be meaningless.
These errors, described with reference to the accelerometer, are the same which occur in the case of the gyroscope. This does not mean that IMU data cannot be used, but that they need a further processing. To solve this problem the implementation of an Extended Kalman Filter is proposed in section 4.2.

3.3 Longitudinal Speed Measurement

Another possible way to measure longitudinal speed of a vehicle is to focus on the angular velocity of its driving wheels. A common way to do this is to use a velocity encoder, which can be mounted on the wheels or, especially in the case of brushless motor, can directly measure the speed of rotation of the motor shaft.
This section proposes a simple solution to estimate longitudinal velocity basing on the same principle of the most used velocity encoders.
The main actor of this tool is a photoresistor (figure 3.18), a device whose resistance

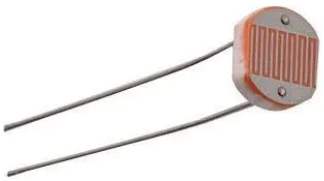


Figure 3.18: Photoresistor

varies depending on the intensity of light.
The device is connected as in figure 3.19, using a series resistor to limit the current

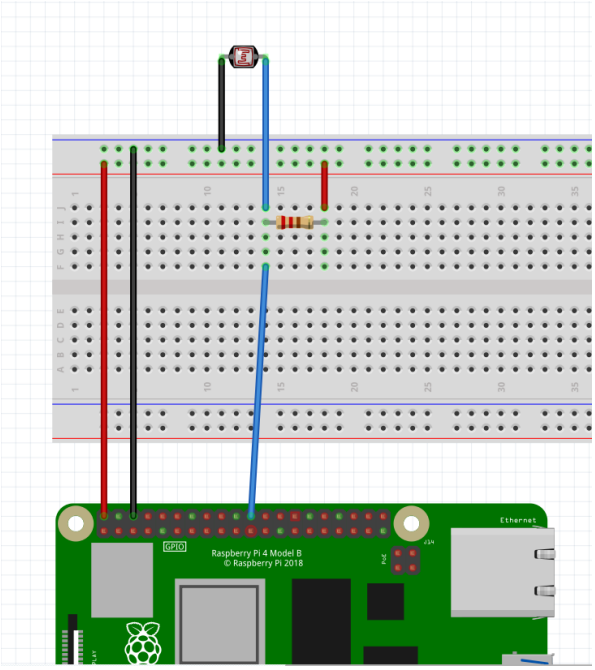


Figure 3.19: Photoresistor circuit

inside the device. The blue wire measures the voltage across the photoresistor, which is an index of the light present.
At this stage, the internal part of one of the back wheels is half painted in white. The photoresistor is placed inside the wheel, together with an LED. When the LED illuminates the white part of the wheel, it reflects a high amount of light which is detected by the sensor. On the contrary, when the black part is on that side, the amount of reflected light is much smaller. In this way, it is possible to count the rotations of the wheel. Knowing also the time interval between each colour switch and the dimension of the wheel, the longitudinal velocity can be estimated.

Chapter 4

Control

Automatic Control theory offers several tools to control different kind of plants in a variety of situations.
One of the features which can push toward the choice of a specific type of controller is the knowledge of the plant. Typically, the most effective control strategies are based on the complete knowledge of the plant, in combination with an accurate estimate of the control variables. Moreover, also in those cases in which a controller which does not require a mathematical model, e.g. a PID, is chosen, the knowledge of the plant can be really useful for simulation purposes, reducing the time needed to tune the controller online.
The advantages of having a model of the plant available are therefore evident.
Several vehicle models can be found in the literature, but for the present case it was chosen to use a 3DOF single track model (bicycle model), which has been extensively described in Chapter 1.
The equations representing the vehicle lateral dynamics are reported here for clarity:

$$a_y = -\frac{2C_{\alpha f} + 2C_{\alpha r}}{mV_x} \cdot V_y - V_x \cdot \dot{\psi} - \frac{2C_{\alpha f} l_f - 2C_{\alpha r} l_r}{mV_x} \cdot \dot{\psi} + \frac{2C_{\alpha f}}{m}\delta \tag{4.1}$$

$$\ddot{\psi} = -\frac{2C_{\alpha f} l_f - 2C_{\alpha r} l_r}{I_z V_x} \cdot V_y - \frac{2C_{\alpha f} l_f^2 + 2C_{\alpha r} l_r^2}{I_z V_x} \cdot \dot{\psi} + \frac{2C_{\alpha f} l_f}{Iz}\delta \tag{4.2}$$

This model can be easily written in state space form, allowing to be used for any type of control strategy considered.

4.1 Plant Model: parameters Estimation

The definition of a model must come together with the knowledge of each parameter and the possibility of controlling and measuring its variables. Looking at equations (4.1) and (4.2), it can be noticed that:

- The steering angle δ can be known from the input command
- The velocities $V_x, V_y, \dot{\psi}$ and the yaw angle ψ can be either measured or derived from the IMU
- The mass m is known

- The parameters $l_f, l_r, I_z, C_{\alpha f}$ and $C_{\alpha r}$ are unknown.

As seen in the previous chapter, the integration of IMU data to get linear velocities and angular positions can bring consistent errors, so it is preferable to use a Kalman Filter to get more cleaned data. Anyway, also for the implementation of this filter, the unknown parameters must be set. Since there is no easy to way to measure these parameters, a different strategy must be evaluated.
All the mechanical parameters such as the longitudinal distance from the center of gravity to front and rear tires and the inertia referred to Z axis can be estimated through CAD modeling of the vehicle.
A different reasoning must be done in the case of the cornering stiffness $C_{\alpha f}$ and $C_{\alpha r}$. The application of the Magic Formula of Pacejka [48] can be rather complex in the case of these tires, not to mention all the additional unknowns that would involve. For this reason, it was chosen to estimate these two parameters in Grey-Box, exploiting all the information already available.

4.1.1 CAD Modeling

The CAD software chosen to model the vehicle is Autodesk Fusion 360, which has a lot of functions to ease the creation of a 3D model, including material characteristics and more visual features. This software also gives the possibility to choose if creating the whole model in a single step or to create components in different environments and then assemble them together. Moreover, it provides a gallery in which widely used components (like SBCs or sensors) are already present and available to be downloaded. With this in mind, it has been chosen to model each part of the vehicle separately and then composing the complete model integrating parts from Autodesk Gallery.
The common principle used for modeling each part was to measure each dimension of the component together with its weight. After modeling the part, it was possible to know a precise estimate of its volume, also in those cases where the geometry was rather complex. Then, it was assumed that the part had a uniform composition, in order to assign to it a density corresponding to its weight and volume.
The origin of the model reference frame was placed at the intersection of the rear axle with the longitudinal axis of the vehicle.
The first part to be modeled was the chassis, which was considered symmetrical with respect to the longitudinal axis. Considering that this model will not be used for reasons other than the estimation of mechanical parameters, some simplifications can be done with respect to the original geometry, which is really elaborate. In this context, the leading principle was to accurately model all the parts which must fit together with other components. This was the case of the two supports of the car body shell, which are converted into supports for the power bank and the camera arm, and also of the bridge which hosts Raspberry Pi. The final result of this design is represented in figure 4.1.
As for wheels, a more faithful representation is provided (figure 4.2). This approach allowed to assign a different material density to the wheel rim and the tire, which was considered hollow. Using typical density values of rubber and plastic, the estimated weight of the designed wheel matched very well the real weight.

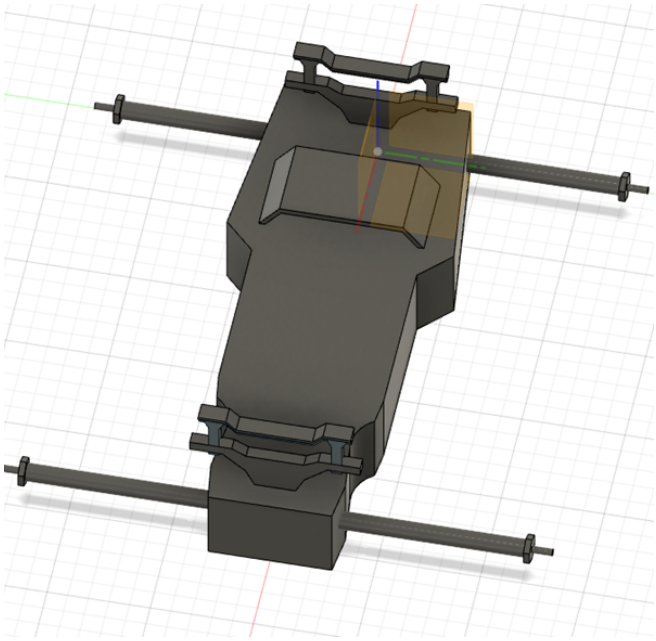


Figure 4.1: 3D representation of the chassis

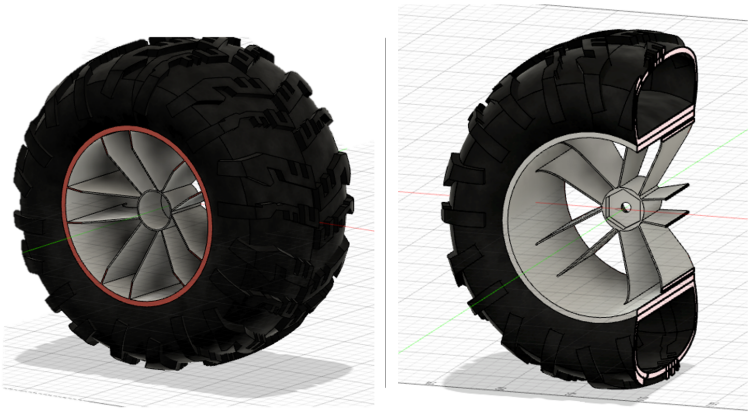


Figure 4.2: 3D representation of car wheel

The last parts which needed to be modeled were the power bank (figure 4.3) and the IMU Lego house (figure 4.4).

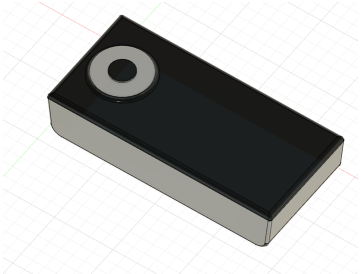


Figure 4.3: 3D representation of power bank

A CAD representation of the IMU itself was not accounted for, as it was weighted

together with its support. Since the weight of the IMU is really small compared to the one of the Lego house, it can be neglected without affecting the final estimation.

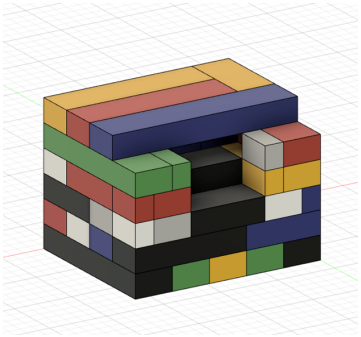


Figure 4.4: 3D representation of IMU Lego house

The final car model must also include the camera support, the Raspberry Pi, the breadboard, and the DC motor driver. The first had already been designed in Fusion 360 and was then available to be added to the project, while all the remaining parts were found on Autodesk gallery and added to the project. In all these cases the weight of the part was verified in order to match real components.
Table 4.1 collects all the components involved in the CAD project along with their size, weight, and material.
Figure 4.5 represents the ensemble with all parts located at their place. At first glance

Table 4.1: Mechanical characteristics of the car's parts

Part	**Size**	**Weight**	**Material**
Chassis	260x268x85mm^3	1054g	Plastic
Wheel	Φ114x64mm^3	122g	Plastic &Rubber
IMU Lego house	81.25x65x60mm^3	132g	Plastic
Power bank	96x45x22mm^3	173g	Mixed
Camera support	42.5x46x181mm^3	30g	Plastic(PLA)
Breadboard	84x56x10mm^3	41.9g	Mixed
L298N motor driver	43x43x26mm^3	29g	Mixed
Raspberry Pi 4B	87x59x19mm^3	56g	Mixed

it can be noticed that all the parts fit together perfectly, also proving the effective design of the camera support and the necessity of a small bridge to raise the Raspberry Pi. If it was not present, there would not be enough space to place the breadboard on a uniform surface. All the significant parameters derived from this model are collected in table 4.2.

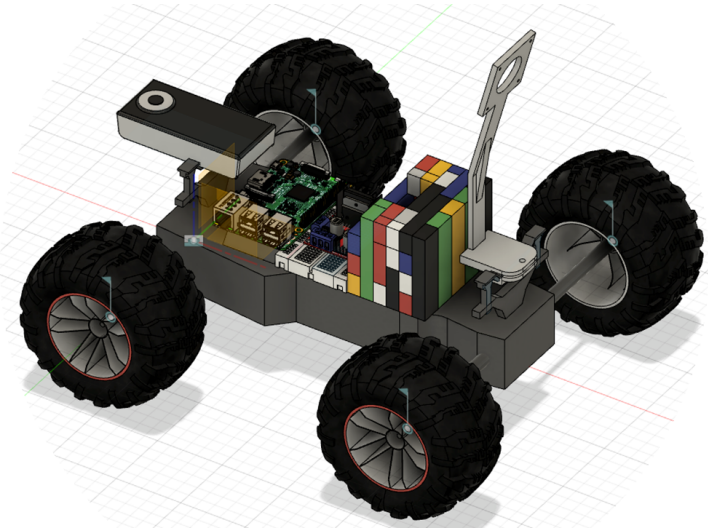


Figure 4.5: 3D representation of the ensemble

Table 4.2: Physical properties

Parameter	**Value**
Mass	2003.01 g
Volume	1.840E+06 mm^3
Area	8.047E+05 mm^2
Position CoM	115.297 mm, -0.339 mm, 33.94 mm
Size	347.919x306.081x289.544 mm^3
Ixx ref CoM	1.064E+07 g mm^2
Iyy ref CoM	1.870E+07 g mm^2
Izz ref CoM	2.501E+07 g mm^2
Lr	100.297 mm
Lf	129.703 mm

4.1.2 Grey Box Identification

Grey box is a system identification approach in which a mathematical model of the system is assumed to be partially known. Through the collection of input and output data, the unknown parameters can be estimated.
This method is very effective when most of the model is exactly known and the output measurements are reliable.
In the case of estimation of cornering stiffness coefficients, this approach seems to be a good choice, as it obviates the need for information on tyres' material properties.
The vehicle model used for the identification is the one defined in equations (1.14) and (1.15).
For identification purposes it is necessary to define the system output in terms of state space. Considering the IMU measures linear accelerations and angular velocities, it is chosen to define as output the vector composed by the lateral acceleration a_y and the yaw rate $\dot{\psi}$. The output equation

$$y = Cx(t) + Du(t) \tag{4.3}$$

is written as

$$y = \begin{bmatrix} a_y \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} A(1,:) & \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_y \\ \psi \\ \dot{\psi} \end{bmatrix} + \begin{bmatrix} B(1,:) \\ 0 \end{bmatrix} \delta \tag{4.4}$$

The idea is then to estimate $C_{\alpha f}$ and $C_{\alpha r}$ from the knowledge of the steering input and the corresponding lateral acceleration and yaw rate (figure 4.6).

Two methods have been evaluated; the first is to use MATLAB function *greyest* [49]

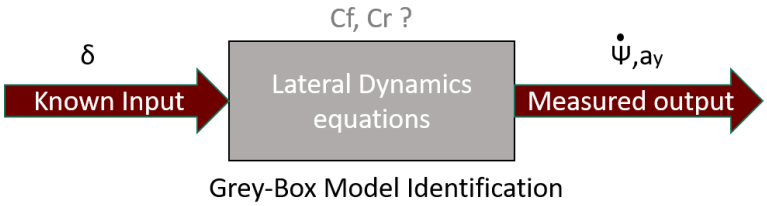


Figure 4.6: Grey box model identification

and the second to use MATLAB *Parameter Estimator App* [50].

In both cases it is useful to test the method with simulated data instead of starting with real IMU data. For this reason, a model with known cornering stiffness coefficients is defined on Simulink (figure 4.7).

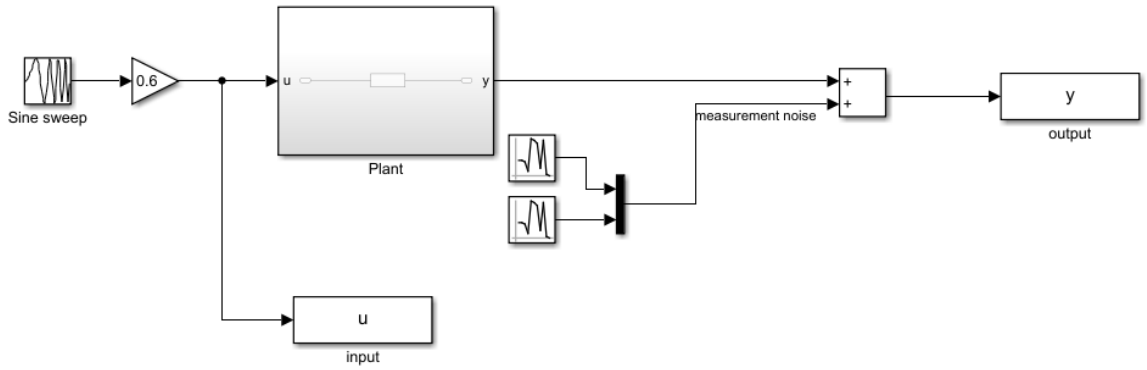


Figure 4.7: Data collection model

The plant is excited with a sine sweep of amplitude 0.6 and input and output data are collected with a sampling time of 0.01s. In order to deal with realistic data, a measurement noise is artificially added with a value corresponding to the standard deviation of IMU data taken with the IMU still.

Using the first method, the state space matrices must be defined inside a function dependent on all the parameters, both known and unknown. An initial estimate of the unknown parameters must be set and a linear grey-box model must be associated with the defined function. At this stage it is possible to indicate which of the given parameters must be estimated. A sample code is provided in figure 4.8.

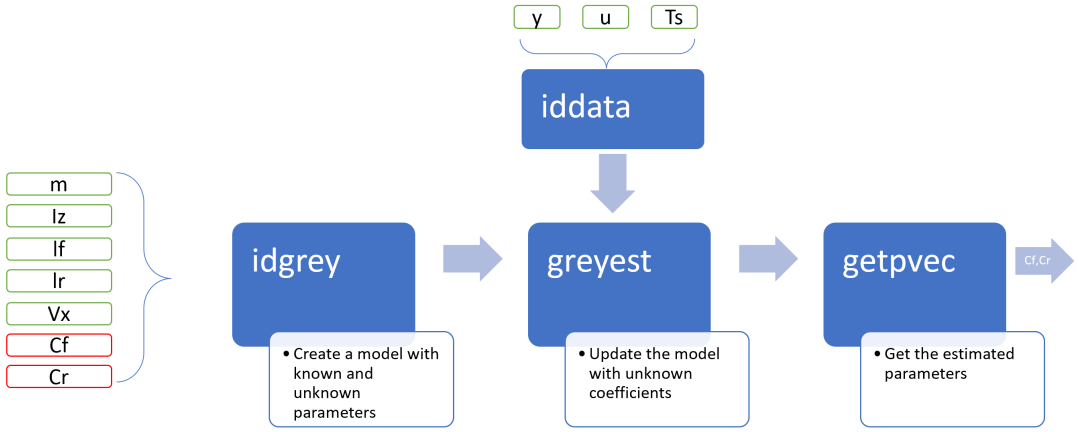


Figure 4.8: Sample code of grey-box model estimation

A first attempt with this estimation technique gives the result in figure 4.9. The actual value associated to both the cornering stiffness coefficients was 20000 and the initial estimation value was 10000. The estimator found a value of 63 for $C_{\alpha f}$ and a value of 600000 for $C_{\alpha f}$. From the graphical representation of the simulated response, it can be noticed that using the lateral acceleration does not help in the estimation. This is due to the fact that, applying a sine sweep, the lateral acceleration changes sign very high frequencies, assuming values not too distant from its noise amplitude. It is then chosen to focus only on the yaw rate.

A further consideration must be done on the obtained results; although the estimated

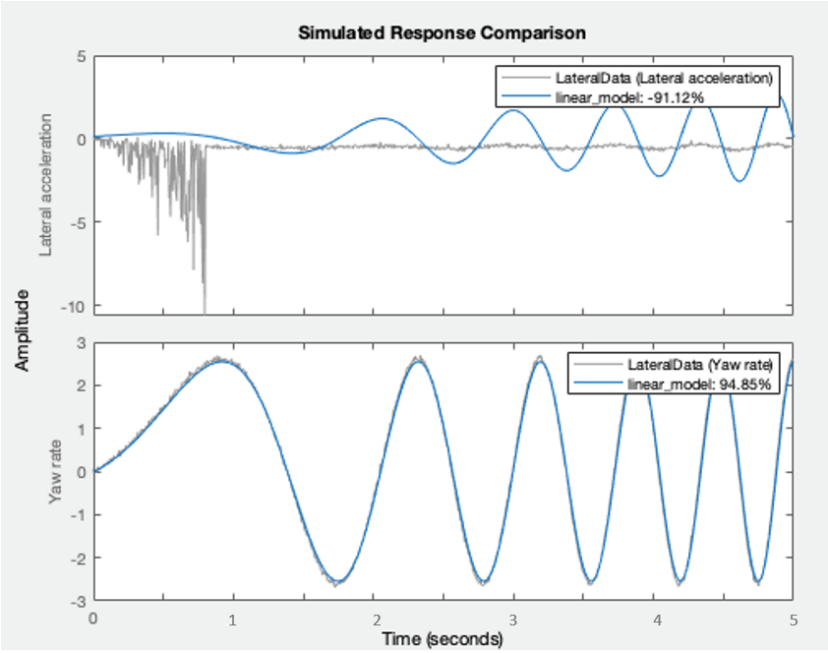


Figure 4.9: Result of estimation using *greyest*

coefficients are very far from those which were used for data collection, the coincidence of the simulated response is very high. The only hint which could let understand that the estimation is completely wrong is the huge difference between the two estimated

coefficients. The two values should actually be very similar, given that the four tires are identical.
The second estimation method is tested. *Parameter Estimator App* uses a Simulink model of the plant to match input and output data. All the known parameters must be set, while a starting value or a range of expected values must be indicated for the unknown parameters. In this case measurements of the simulated plant are collected with a sharper sine sweep, going from 0.1Hz to 5Hz in 10s.
It must be remarked that this app performs a high number of iterations on the Simulink model of the plant. From first attempts it was noticed that this procedure can be rather time consuming, especially when the computer used for estimation is not extremely powerful. For this reason, it is better to define the simplest possible model, which proved to be a state space model implemented with elementary blocks. To get an idea, the block scheme of the model is shown in figure 4.10.

The app provides several optimization methods: nonlinear least squares, gradient

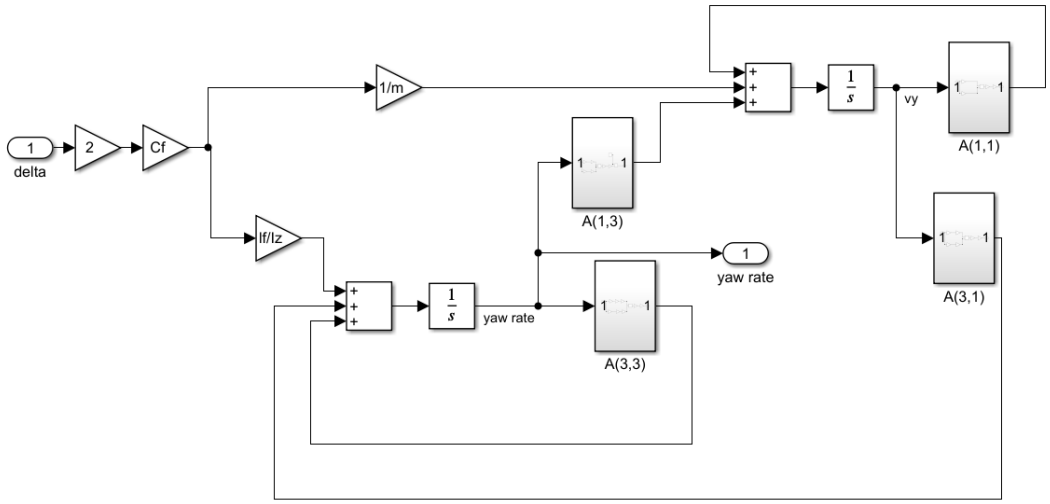


Figure 4.10: Simulink model for *Parameter Estimator App*

descendent, pattern search and simplex search. Some tests are conducted with all the optimization methods, revealing that the best optimization method for this problem is simplex search. Furthermore, it was recorded that it is better to perform a first estimation imposing $C_{\alpha f} = C_{\alpha r}$, and then using the found value as a starting point for a second estimation with different coefficients. Using these settings, the result in figure 4.11 is obtained. Experimental data had been collected using $C_{\alpha f} = 20000\frac{N}{rad}$ $C_{\alpha r} = 30000\frac{N}{rad}$, so the result $C_{\alpha f} = C_{\alpha r} = 23923\frac{N}{rad}$ seems to be sufficiently valid.
The procedure is then repeated once again using real data collected with the vehicle travelling at a speed of 1m/s.
The first part of the simulation is neglected, since the vehicle starts from rest and velocity is not constant in the first part. From figure 4.12 it can be seen that while the steering input signal has a constant amplitude, the amplitude of the yaw rate decreases with time. This can be due to two reasons: the cornering stiffness is very small or the actual input does not correspond to the given one. In fact, the input used for estimation is the one set in Simulink, as there is no hardware available to directly measure the orientation of the tires. It can be possible that the servo cannot react

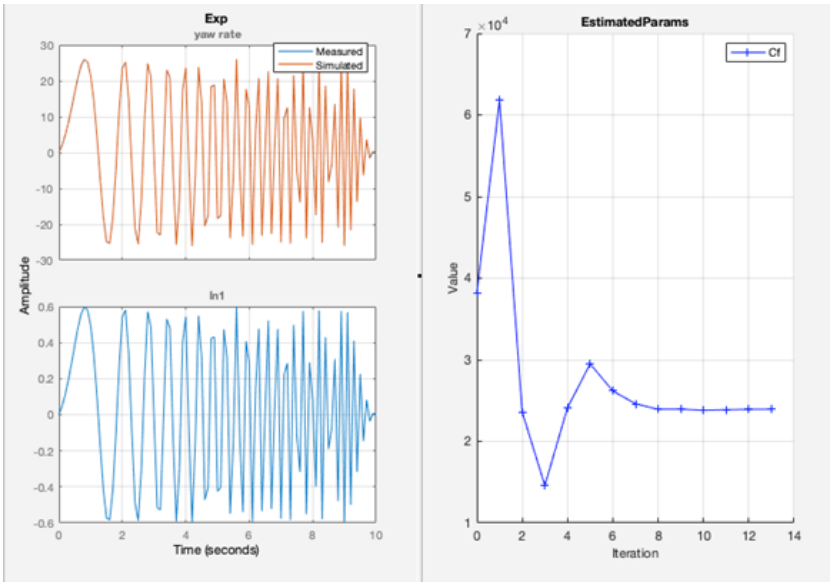


Figure 4.11: Estimation of $C_{\alpha f} = C_{\alpha r}$ using simplex search

fast enough to high frequency signals. Anyway, assuming input data are coherent and considering $C_{\alpha f} = C_{\alpha r}$, it results $C_{\alpha f} = C_{\alpha r} = 12.4\frac{N}{rad}$. This result is in the same order of magnitude of the one found in [51], where values close to $70\frac{N}{rad}$ where found using Pacejka's Formula. Further investigation could be done using a servo which guarantees the possibility of working at very high frequency, until then the obtained result can be used.

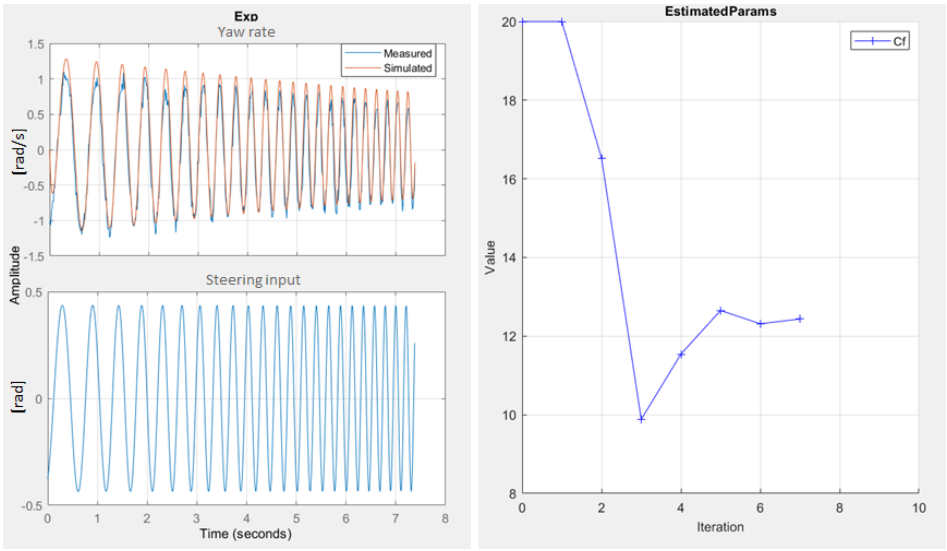


Figure 4.12: Estimation of $C_{\alpha f} = C_{\alpha r}$ using real data

4.2 Kalman Filter

The implementation of a Kalman filter allows to obtain an estimate of the lateral velocity and the yaw angle directly using the measurement of the lateral acceleration and the yaw rate. In this way, integration of data affected by noise and/or bias can be

avoided.
The Kalman filtering problem for a LTI system is formulated as follows [52]:

$$\begin{cases} \dot{x} = Ax(t) + Bu(t) + v_1(t) \\ y(t) = Cx(t) + v_2(t) \end{cases} \tag{4.5}$$

with
$v_1(t)$=process noise, $v_1(t) \sim WN(0, V1)$
$v_2(t)$=measurement noise, $v_2(t) \sim WN(0, V2)$.
In this case

$$x = \begin{bmatrix} v_y \\ \psi \\ \dot{\psi} \end{bmatrix}$$

$$A = \begin{bmatrix} -\dfrac{2C_{\alpha f} + 2C_{\alpha r}}{mV_x} & 0 & -V_x - \dfrac{2C_{\alpha f} l_f - 2C_{\alpha r} l_r}{mV_x} \\ 0 & 0 & 1 \\ -\dfrac{2C_{\alpha f} l_f - 2C_{\alpha r} l_r}{I_z V_x} & 0 & -\dfrac{2C_{\alpha f} l_f^2 + 2C_{\alpha r} l_r^2}{I_z V_x} \end{bmatrix} B = \begin{bmatrix} \dfrac{2C_{\alpha f}}{m} \\ 0 \\ \dfrac{2C_{\alpha f} l_f}{Iz} \end{bmatrix}.$$

Considering the IMU measures linear accelerations and angular velocities,

$$y = Cx(t) + Du(t) + v_2(t), \tag{4.6}$$

where

$$y = \begin{bmatrix} a_y \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} & A(1,:) & \\ 0 & 0 & 1 \end{bmatrix} x + \begin{bmatrix} B(1,:) \\ 0 \end{bmatrix} u + v_2(t) \tag{4.7}$$

Since the Kalman Filter is defined for a LTI system without a feedthrough matrix D , and keeping in mind that the system input is exactly known, a different output can be defined to feed the Kalman block:

$$y_{Kalman} = y - D \cdot u(t) = Cx(t) \tag{4.8}$$

The Simulink scheme associated with this formulation is reported in figure 4.13.

The system is expressed in discrete time and the EKF algorithm is applied following equations (1.33) (1.34). Through a trial-and-error procedure, matrix $Q^d = diag([100.10.1)]$ is defined, while R^d is chosen as a diagonal matrix having as elements the standard deviation of IMU data for a_y and $\dot{\psi}$ collected with the IMU still ($R^d = diag([0.0060.002])$).

The superposition of simulated values and EKF estimated values are plotted in figure 4.14, respectively in blue and red. It can be noticed that the sensor noise and the sampling time affect the estimation of the yaw angle and the yaw rate. Anyway, these estimates are really good if compared to those that would be obtained by integration (figure 4.15).

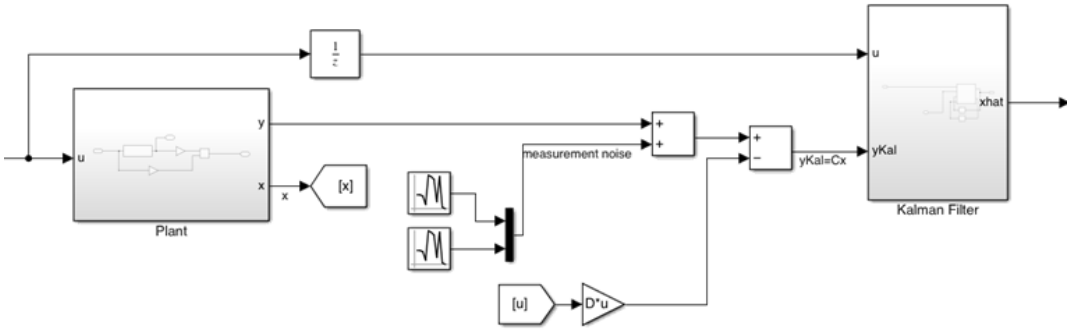


Figure 4.13: Kalman filter block scheme

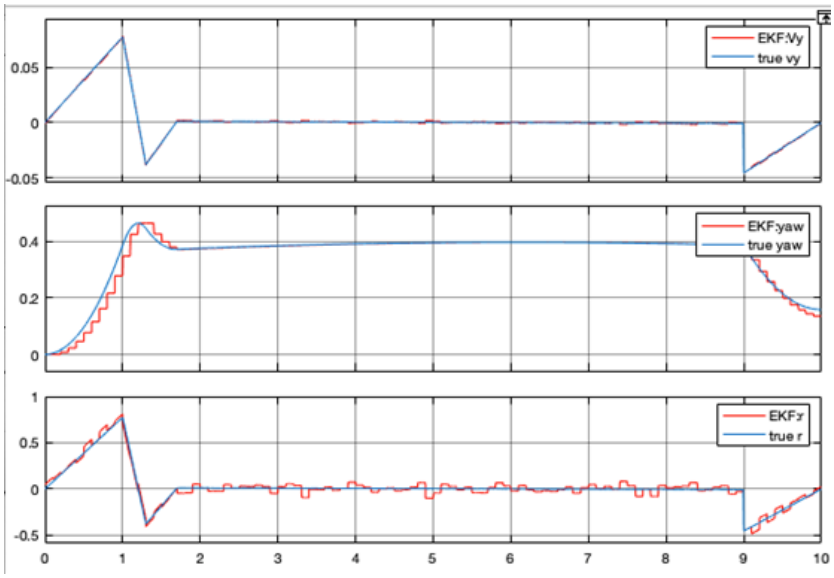


Figure 4.14: Kalman filter block scheme

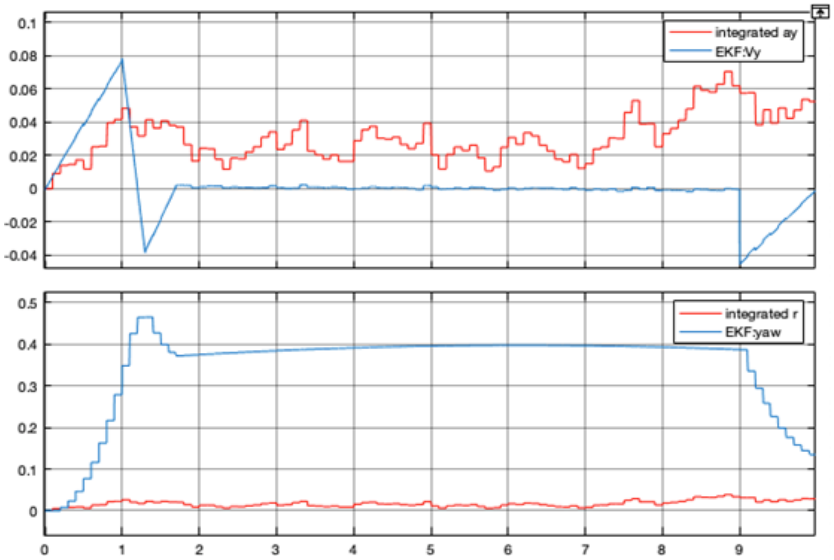


Figure 4.15: Kalman filter block scheme

4.3 Lateral Control

The control strategy selected for the vehicle's lateral dynamics is based on Stanley control law:

$$\delta(t) = \begin{cases} \frac{\pi}{2} + \psi(t) + arctan\frac{ke(t)}{v(t)+k_{soft}} & if\ \delta_{min} < \frac{\pi}{2} + \psi(t) + arctan\frac{ke(t)}{v(t)+k_{soft}} < \delta_{max} \\ \delta_{max} & if\ \frac{\pi}{2} + \psi(t) + arctan\frac{ke(t)}{v(t)+k_{soft}} \geq \delta_{max} \\ \delta_{min} & if\ \frac{\pi}{2} + \psi(t) + arctan\frac{ke(t)}{v(t)+k_{soft}} \leq \delta_{min} \end{cases} \tag{4.9}$$

where the minimum and maximum steering input are respectively 0 and π. The obtained angle must be converted in degrees to be given as input to the servo, as it will be extensively explained in the next chapter.
Another difference with respect to equation (1.22) is the presence of a softening constant k_{soft}. This constant can be tuned according to the vehicles' longitudinal speed, which in standard conditions will be kept under 2m/.

Chapter 5

Actuation

The working loop of an AV closes with actuators, which transform into action the command computed in the control section.
As it was said several times the actuation line is composed by a servo and a DC motor. The first is directly controlled by Raspberry Pi and the second needs a driver to be controlled.
Figure 5.1 represents the complete wiring of these components and can be useful to understand their working principle, which will be extensively explained in the next sections. 5.1

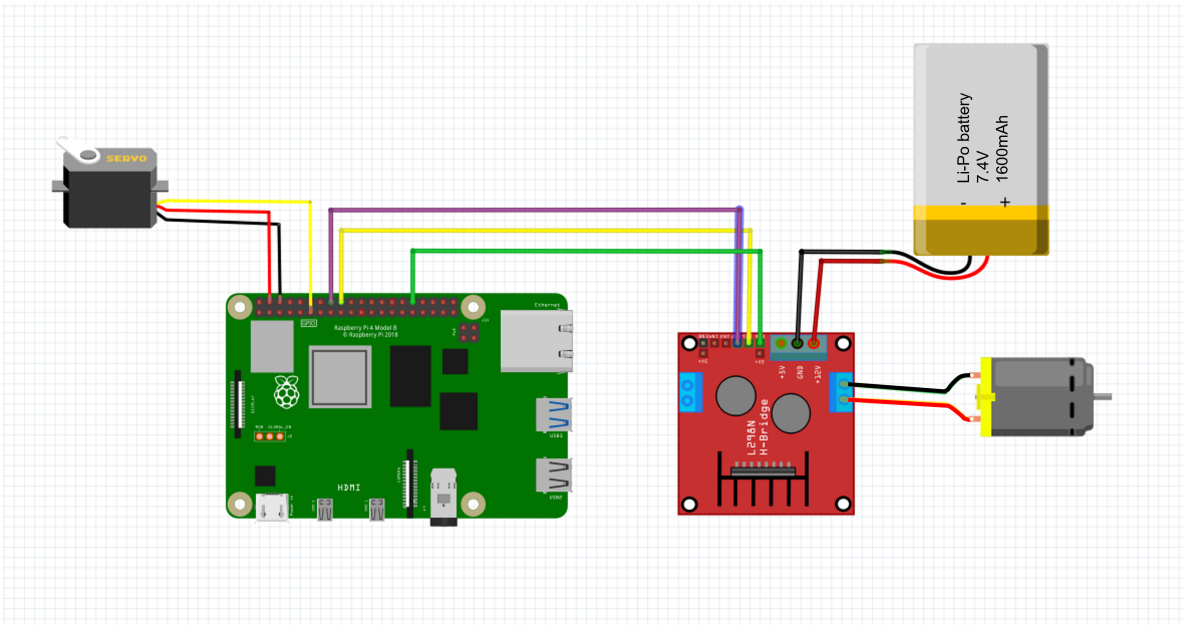


Figure 5.1: Wiring of actuation line

5.1 Steering Actuation

The actuation of the steering command is entrusted to a micro servo (figure 5.2). This device can be directly controlled by Raspberry Pi GPIO, and its dimension is such that it can fit into the case of the previous RC servo preserving the already present kinematic chain.

The reduction gears contained inside the case are very similar to those of the original RC part and work in an analogue way.
The main difference between this servo and the RC one is the presence of three wires

Figure 5.2: Micro servo

instead of 5. Typically, they are:

- Red wire: it indicates the supply voltage which must be in the range between 4.6V and 6V
- Black wire: ground voltage
- Yellow (or white) wire: PWM input signal.

As it can be imagined, this means that the electronic which ensures the right working of the servo is contained inside the device, so there is no need for a device similar to the RC receiver.
Following these specifications and referring to pin enumeration in figure 2.9, the following connections are made:

- The red wire is connected to pin 2
- The black wire is connected to pin 3
- The yellow wire is connected to GPIO 17, which corresponds to pin 11.

Generally, there exist servo libraries which allow to control these devices indicating the position of the output shaft in degree. In this case there is a MATLAB function called *servo* and a corresponding Simulink block (figure 5.3) which develop this function.
The input must range between 0 and 180, which indicate the degrees of rotation of

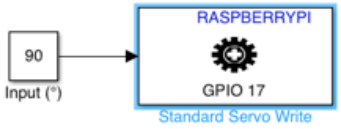


Figure 5.3: Simulink servo block

the shaft with respect to a fixed position. With this in mind, the servo is connected to its kinematic chain with wheels pointing forward and an input of 90°.
Consequently to the mechanical configuration, the wheels point left when the input is

0 and right when the input is 180, reaching the maximum degree of rotation in both senses. It must be noticed that due to the shape of the kinematic chain, the rotation of the servo's shaft does not correspond to the rotation of the wheels, which instead occurs in 1:2 scale. This ratio is observed experimentally and must be considered just an approximation.
Figure 5.4 depicts the three most important states of the vehicle wheels which correspond to an input of 0,90 and 180.
It must be noticed that MathWorks' function and block does not consider the possi-

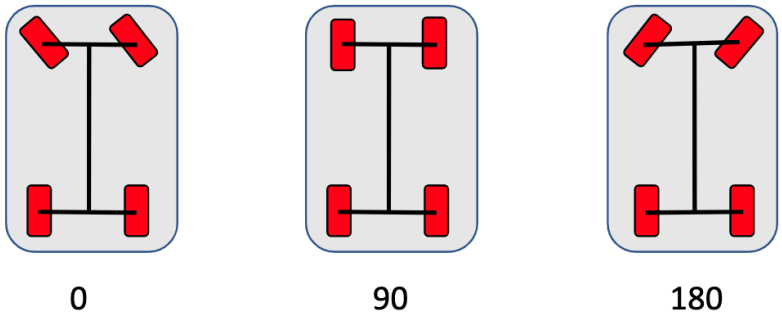


Figure 5.4: Wheels position corresponding to servo input in degree

bility to get an input outside the 0-180 range. Hence the signal entering the block must be saturated to avoid errors.

5.2 Control of DC motor

As per the DC motor, all the technical information about its working principle is explained in the hardware section. What is still missing is a description of the way this motor is controlled through Raspberry Pi.
It has been already said that there is the need of a power driver to interface the motor with the SBC. This occurs because Raspberry Pi is not built to control devices through variations of voltage and neither it is designed to provide high intensity currents. A device which is made for these exact purposes is a DC motor driver.
Motor drivers are usually classified considering three characteristics:

- Working current
- Working voltage
- Peak current.

In order to choose a proper driver these characteristics of the motor to be controlled must be known in advance.
The voltage required by the motor can be easily detected just looking at the Li-Po battery which powers the RC car. Its nominal voltage is 7.4V, then it can be expected that it can power the motors in a neighbourhood of 2V of that value.
As far as current is concerned, it is known from the information on the RC receiver that it can manage up to 60A. Anyway, as seen in the hardware section, it is likely that

this value is reached only when starting the DC motors.
To know the operating current of the motor, it is connected to a multimeter and a voltage going from 0 to 9.4V is supplied. Using the latter value, the absorbed current amounts to approximately 0.7A.
On the basis of these data, a module which can drive DC motors in a wider range

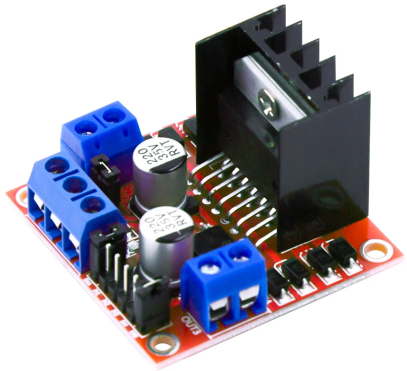


Figure 5.5: L298N DC motor driver

than 6.4-9.4V, with a peak current of at least 60A, should be found. Anyway, since the reliability of these data is poor and driver which can manage such high currents are rather expensive, it has been chosen to use a less powerful driver which can work with average voltages and currents.
The module chosen is a L298N dual H-Bridge motor driver (figure 5.5), which can drive DC motors with voltages between 5 and 35V and a peak current of 2A. Thanks to the dual H-bridge, it would be able to control speed and direction of a couple of DC motors, so, in the case in which it is chosen to add the second motor in the autonomous architecture, there will be no need for another driver.
The choice of this driver must be seen as an intermediate step to start controlling the motor and verify its working range. It is very likely that it will not be sufficient to start the motor, especially at low voltages, but after it is externally put into rotation, it will provide enough current to let it operate in normal conditions.
The wiring of this module is depicted in figure 5.1:

- Enable-GPIO12 of Raspberry Pi
- IN1-GPIO24 of Raspberry Pi
- IN2-GPIO23 of Raspberry Pi
- Supply voltage-Li-Po battery +
- Ground voltage-Li-Po battery -
- OUT1-DC motor +
- OUT2-DC motor -.

With such a configuration, motor speed depends on the intensity of the PWM at the Enable pin, which once again can be controlled using a MATLAB function, *writePWMVoltage* and a corresponding Simulink block. In this case the input signal must be indicated in a continuous range from 0 to 1.
The sense of rotation is regulated according to table 5.1.

Table 5.1: Working principle of L298N module

Verse	**IN1**	**IN1**
FORWARD	HIGH	LOW
REVERSE	LOW	HIGH

5.2.1 Characterization of DC Motor

When choosing a motor for a project, it is often possible to examine its datasheet, which provides useful information on the motor behaviour under some defined conditions. In the case of the RC car under analysis, all parts came together in a unique component and no technical information was provided about each item. The absence of these data sets an obstacle in the definition of an effective control strategy. For this reason, it was chosen to perform an input-output characterization of the DC motor.
The vehicle was placed on a 20m track and let drive forward, changing the PWM intensity and measuring the time interval necessary to drive through the track. For each input the procedure was repeated a few times to reduce errors due to measurement uncertainty.
The table in figure 5.6 collects the results of the first set of measurements using different colours to indicate different input. On the last column there is the computation of linear velocity computed as space over time. After computing the mean velocity corresponding to the same input, values distant from the mean of over two standard deviations were marked as outliers (red) and excluded from the computation of the mean.
The final result of this computation is shown in figure 5.7.
As it can be noticed, while the possible input ranges from 0 to 1, the applied input starts only from 0.6. The reason behind this choice is that when a lower input is given, the power given to the motor is not sufficient to let it start moving and therefore is meaningless to characterize the motor in that region.
This does not mean that the motor cannot work with voltage PWM between 0 and 0.6 but that it will work only if it is already spinning. It is likely that using a more powerful driver, the motor could start also at lower voltage values, but this is not really relevant for the purpose of this project.
Another issue that must be considered is that the plane on which the tests are conducted might not be actually plane. For this reason, another set of measurements is collected following the traveling the path the other way. Comparing results in figure 5.8 with those obtained in the first test with the same input, it is clear that the road is sloped. From these measurements it is possible to compute the slope of the road and scale the

PWM intensity	Space(m)	Time(s)	Velocity(m/s)
0.6	10	6,95	1,43884892
0.6	20	16,34	1,22399021
0.6	20	15,54	1,28700129
0.6	20	16,18	1,23609394
0.7	10	5,58	1,7921147
0.7	20	13,52	1,47928994
0.7	20	13,7	1,45985401
0.8	20	12,07	1,65700083
0.8	20	12,46	1,60513644
0.8	20	12,33	1,62206002
0.9	20	11,06	1,80831826
0.9	20	11,68	1,71232877
0.9	20	11,87	1,68491997
0.9	20	10,2	1,96078431
0.9	20	10,31	1,93986421
1	20	9,97	2,00601805
1	20	10,21	1,95886386
1	20	10,42	1,9193858
1	20	9,66	2,07039337
1	20	9,18	2,17864924
1	20	9,54	2,09643606

Figure 5.6: Characterization of DC motor: first set of measurements

PWM intensity	Velocity(m/s)	Mean time(s)
0,6	1,24902848	16,02
0,7	1,46957198	13,61
0,8	1,62806576	12,2866667
0,9	1,82017041	11,024
1	2,11515956	9,712

Figure 5.7: Characterization of DC motor: mean values of the first set of measurements

velocities in figure 5.7. Using two simple formulas for the inclined plane

$$v = v_0 + at \tag{5.1}$$

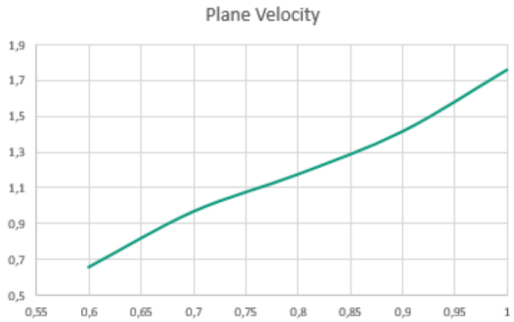
$$a = a_0 + mg\sin\theta \tag{5.2}$$

the estimated slope is of 0.215°. Exploiting this information, the velocity corresponding to a planar road is computed (figure 5.9).

The obtained curve is fitted in MATLAB using *polyfit* with polynomial degrees going from 1 to 4. Considering that when the motor is switched off the velocity is 0, also the point (0;0) is considered. From figure 5.10 it seems reasonable to approximate the obtained curve to a polynomial of degree 1.

PWM intensity	Space(m)	Time(s)	Velocity(m/s)
1	20	15,07	1,32714001
1	20	12,88	1,55279503
1	20	15,27	1,30975769

Figure 5.8: Test in the other way with input 1



PWM	Plane Velocity (m/s)
0.6	0,656363538
0.7	0,966065745
0.8	1,17351666
0.9	1,412334064
1	1,755861902

Figure 5.9: DC motor characterization: planar velocity

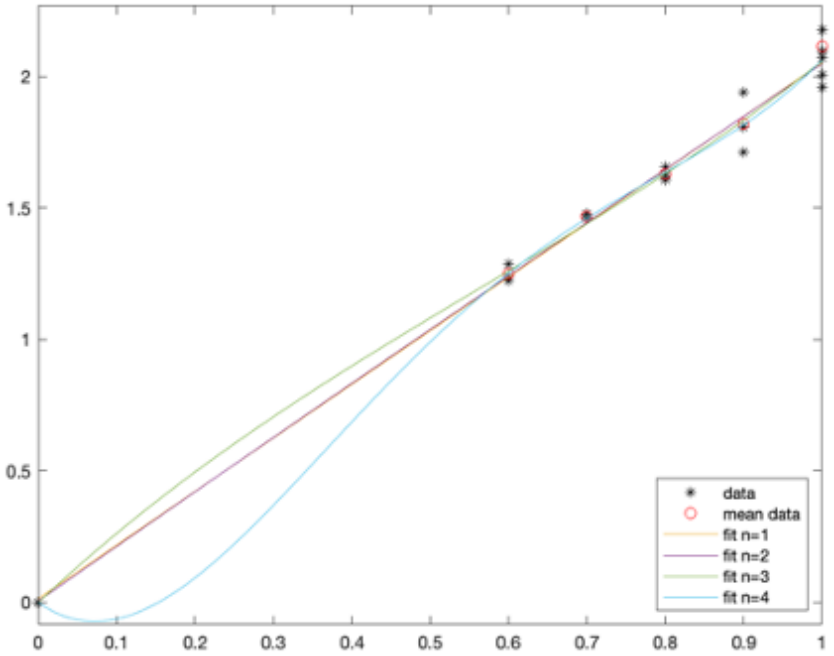


Figure 5.10: Characterization of DC motor: polynomial fitting

Chapter 6

Results

The assessment of the work done is made through two stages of testing and collection of results. First of all, the capabilities of the actuation pipeline are tested, loading a steering profile, and setting a fixed longitudinal velocity.
Once no doubts are left on the correct working of actuators, the perception and control algorithms are tested. Together, they provide the steering action in real-time while the longitudinal velocity is kept constant.

6.1 Pre-loaded steering profile

The creation of a steering profile is the first step to take in order to test the reaction of the servo motor. To do so, a path is created on Driving Scenario Designer App (figure 6.1) for a 1:1 scale vehicle. The simulated vehicle has length 3.3m and width 3m, with a wheelbase of 2.3m. The road width is set to 6m.
The easiest way to get a steering profile matching this track, is to define the vehicle's

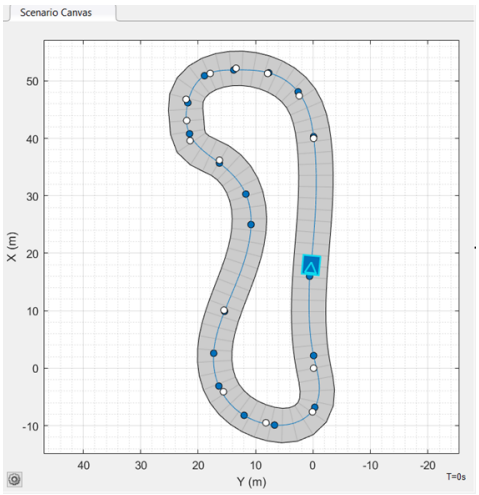


Figure 6.1: Path created on Driving Scenario Designer

waypoints and then let a Pure Pursuit controller decide the best steering action to be taken. To this purpose, the control scheme in figure 6.2 is used, where the vehicle is modeled as 3DOF body with dual track and the longitudinal velocity is set to 8m/s.
The only parameter to be set is the *lookahead distance ld*. The latter is tuned through

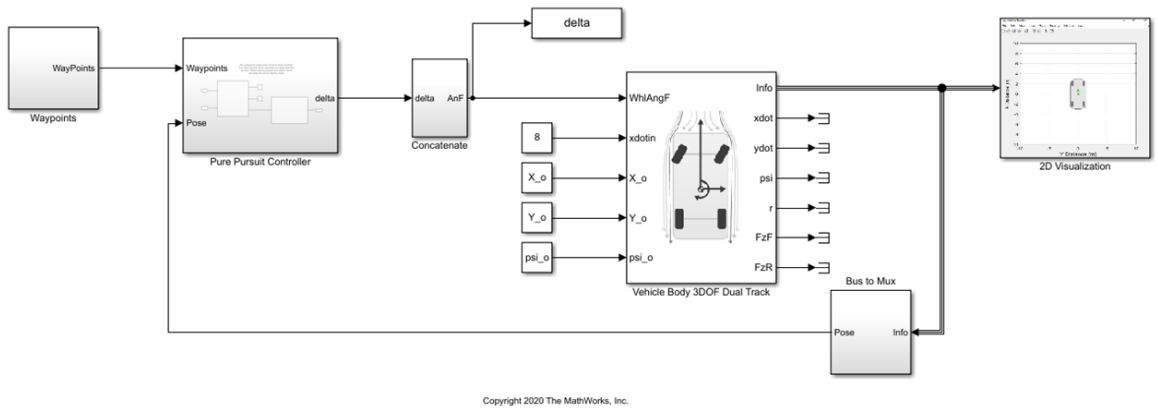


Figure 6.2: Pure Pursuit Control scheme implemented in Simulink

a trial-and-error procedure, which is summarized in figure 6.3. A small *lookahead distance* of 1.5m leads to a jagged path which too often takes the vehicle off-track, while with $ld = 6m$ the vehicle does not follow well the first curve. The best trade-off is found for $ld = 3.6m$.

After setting all the parameters, the simulation is started and the steering angles com-

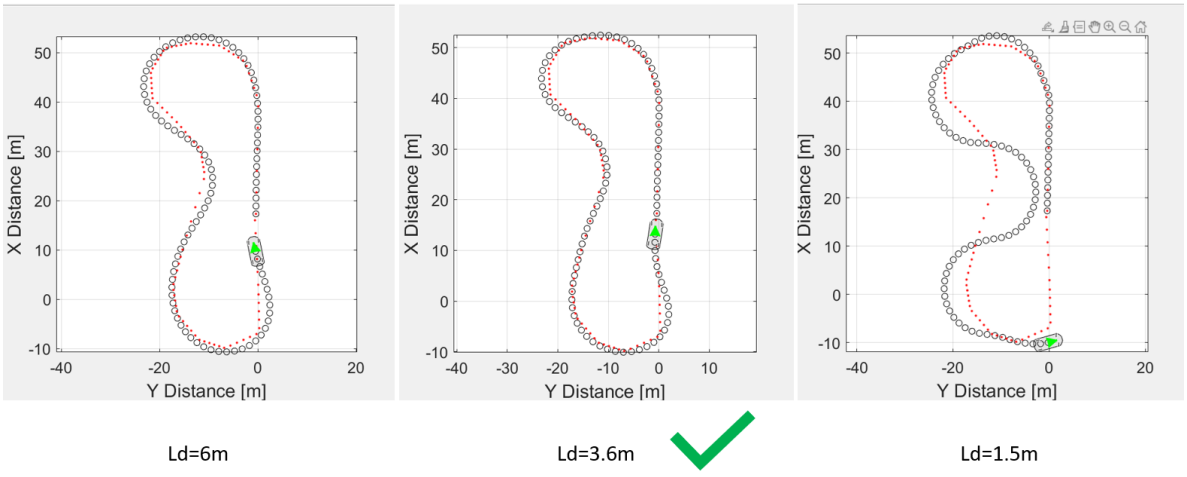


Figure 6.3: Comparison between different *lookahead* distances

puted in Simulink are collected and saved in a timeseries, which contains both steering angle and time data. A graphical representation of the obtained steering profile is shown in figure 6.4.

This profile is then applied to the 1:10 scale model, after a small acceleration phase on a straight line, with the aim of reaching a constant speed.

Figure 6.5 qualitatively shows the vehicle's trajectory compared to the expected waypoints. It can be noticed that the shape of the trajectory resembles the desired one, but it is very inaccurate. Two main causes are identified; first, the road surface had a slight slope towards the negative Y direction. This led to a small acceleration in that direction and a small deceleration in the opposite one. Secondly, it must be considered that the steering profile was obtained for a 1:1 scaled vehicle which above all has a

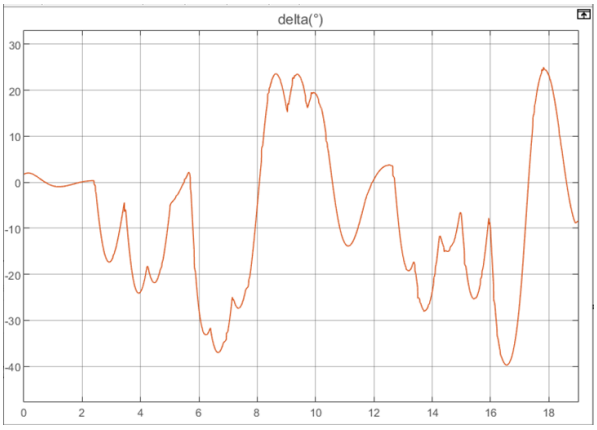


Figure 6.4: Steering profile

different dynamic behaviour, especially in terms of friction.
The positive aspect of this test is the reaction of the servo motor to the input variation, which is a good starting point for more challenging tests.

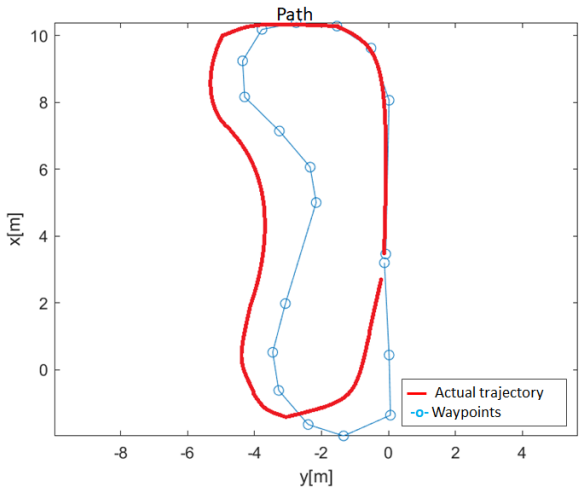


Figure 6.5: Actual trajectory Vs planned trajectory

6.2 Steering command from perception and constant longitudinal velocity

The second stage of testing aims at validating the combined working of the perception and control part. To this end both the camera and the IMU are switched on and a Lateral Controller Stanley strategy is implemented. Figure 6.6 describes the behaviour of the whole system.

The lane detection algorithm provides real-time estimation of road curvature k, lateral

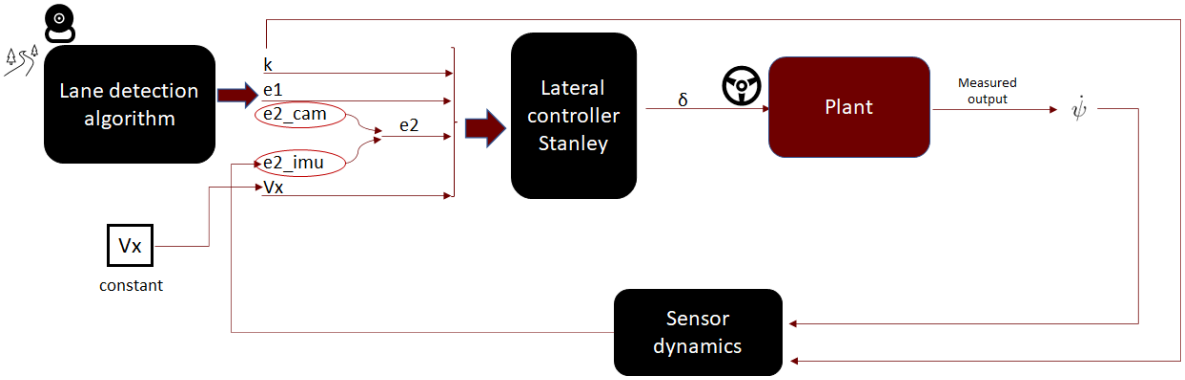


Figure 6.6: Perception and Lateral Control

deviation e_1 and heading error $e_{2_{CAM}}$. At the same time, the IMU measures the yaw rate and exploits the information about curvature to compute the derivative of the heading error $\dot{e}_{2_{IMU}}$. Both the estimates of the heading error are fused together to obtain a final estimate e_2. The controller, which exploits a kinematic model of the plant, uses the error data and the velocity input (which once again is kept constant) to compute the steering angle.

Camera parameters are shown in figures 6.7 and 6.8 and defined as follows:

- Distance ahead the sensor considered for lane detection: 4m
- Space to each side considered for lane detection: 2.3m
- Camera height from the ground: 0.3m
- Camera inclination with respect to the vertical axis (pitch): 10.5°

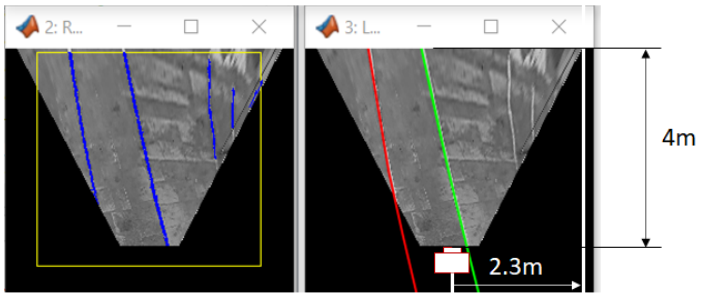


Figure 6.7: Camera parameters: distance ahead of sensor and space to each side

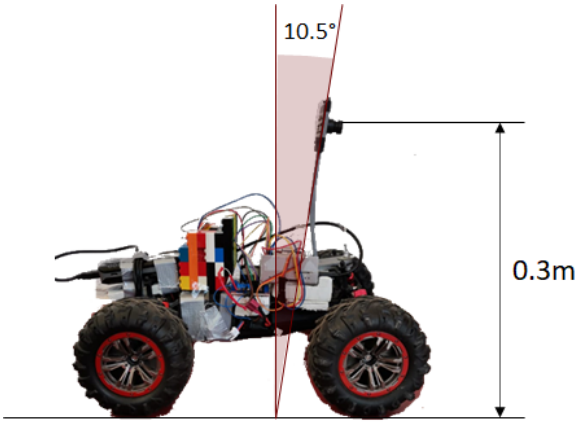


Figure 6.8: Camera parameters: camera height and pitch

The choice of these parameters is made after a previous testing phase which highlighted a few issues in Vision. In particular, the identification of high curvature lanes proved to be particularly difficult, as can be seen in figure 6.9, where the considered

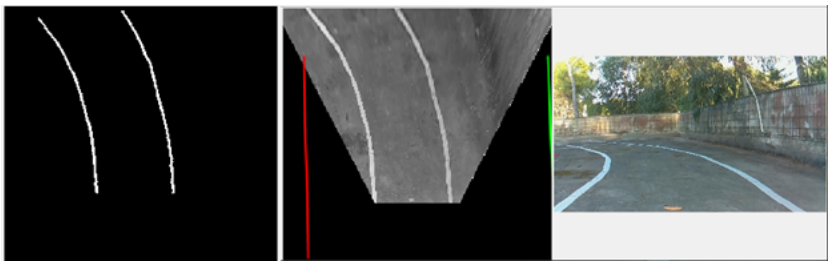


Figure 6.9: Detection of a curvy road

distance ahead the sensor was 8m. Despite the good results of colour extraction, the lane is not well detected. This is the reason why it was chosen to set a distance of 4m instead, since a smaller part of the road tends to appear much less curvy, and it is easier to detect.
Another issue is the stability of the camera support. While placing the camera higher provides a wider field of view, it also leads to greater oscillations due to road asperities. It mainly causes the variation of the pitch angle, which translates in a bad Bird's Eye View transformation. A height of 30cm is a good trade-off which allows a more than sufficient width of the useful field of view and a small range of oscillations, especially when the vehicle's speed is kept relatively slow.
In this preliminary test, the lane detection algorithm proved robust to light variations due to tree shadows and small parts of road of lighter colour.
Alongside, also the tuning of the Stanley Lateral controller has been carried out. After a few attempts at a speed of 1m/s and with Stanley gain $k = 2$, the steering action appeared too sharp. For this reason, it was decided to add a softening constant $k_{soft} = 3$ to the control law, which is suitable in low- speed cases like this.
The tuning of the algorithm is made through several simulations and tested in different conditions. Figure 6.10 and table 6.1 summarize the guidelines followed in this phase. Setting the velocity $v = 1m/s$, $k = 2$, $k_{soft} = 3$, the control algorithm was tested in all possible combinations of positive and negative lateral deviations and heading error,

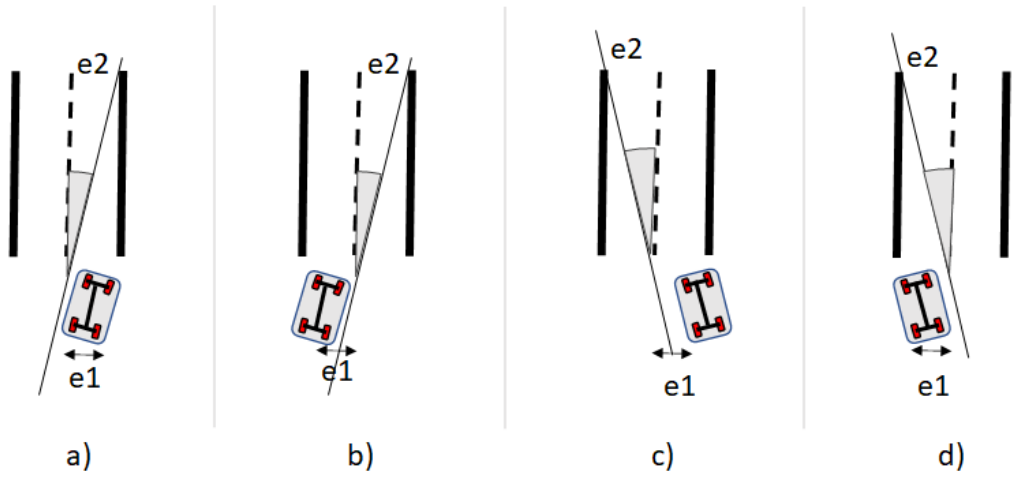


Figure 6.10: Testing conditions of Stanley Lateral Controller

Table 6.1: Testing results of Stanley Lateral Controller

CASE	a)	b)	c)	d)
Heading error(°)	-10°	-10°	10°	10°
Lateral deviation(m)	0.2m	-0.2m	0.2m	-0.2m
Servo input (°)	-15.7°	-4.3°	4.3°	-15.7°

providing reasonable results.
Since the various part of the system are set and validated separately, a test of the behaviour of the whole system depicted in figure 6.6 was performed. The vehicle was placed in a 20m track with two small curves, one left and one right and let drive autonomously.
The output of the perception pipeline is composed by four windows updated at each loop to show the computations made by the algorithm (figure 6.11).

Figure 6.11: Perception output

These figures are very useful in the tuning phase to verify the correct working of all parts.
Heading error and lateral deviation are well estimated and the frame, used by the cam-

era to analyse the path, corresponds to the one set, also proving the accuracy of the calibration procedure. Anyway, real-time testing reveals that the perception and control loop is too slow. This means that images are processed with a certain delay and, at the moment in which the command input is provided, the configuration of the road has already changed. Once the vehicle is off-track, the lane can be too far from the chosen frame and thus it never returns to the right path.
It is known that printing figures or strings represents an issue in terms of loop time. For this reason, the print of figures was suppressed, and the loop time was measured. The result of this comparison is showed in table 6.2, where the mean loop time computed

Table 6.2: Loop duration

LOOP TYPE	LOOP TIME(s)
With figures	0.29
Without figures	0.12

over 300 loops is showed. It is made evident that suppressing figures, the loop time is reduced of about 2.4 times.
The whole algorithm was tested in this condition. The performance was good enough to let the vehicle drive autonomously along the path and deal with both curves successfully.
A more challenging test is performed with sharper curves to set the limits of the algorithm (figure 6.12).
Figure 6.13 reports the vehicle's behaviour while driving at a speed of 0.75m/s using

Figure 6.12: Vehicle testing on a curvy track

Driving Scenario Designer App, while figure 6.14 shows its lateral deviation along the path, which never overcame 15cm, thus meaning staying 10cm far from the closest lane mark. At this speed, the vehicle could go through curves of about 8m of curvature, while raising the speed to 1m/s, the maximum radius that can be dealt with is 15m.

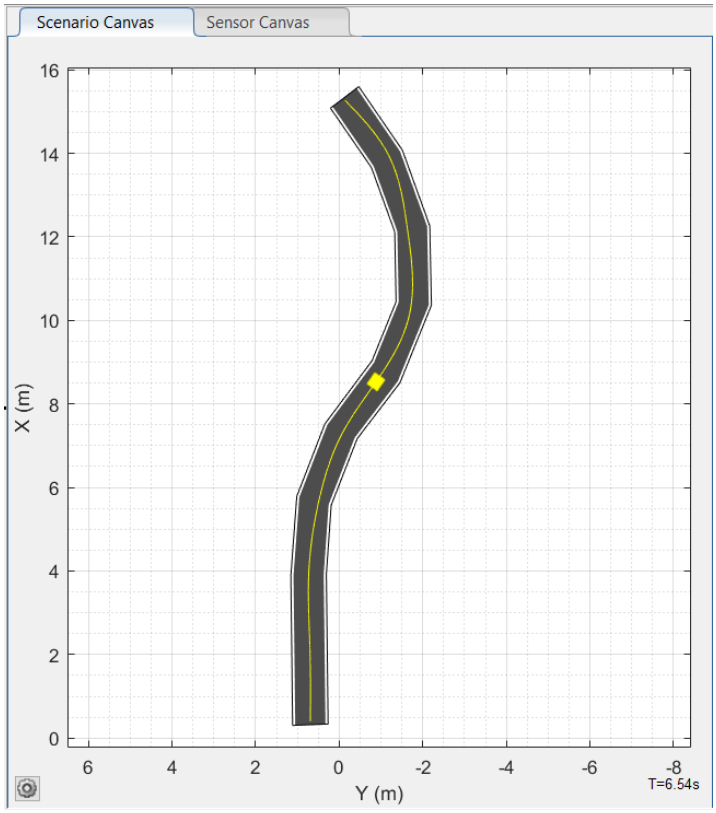


Figure 6.13: Vehicle's behaviour in Driving Scenario Designer App

The maximum speed that could be reached is 2m/s but it can be managed only on roads with very small curves.

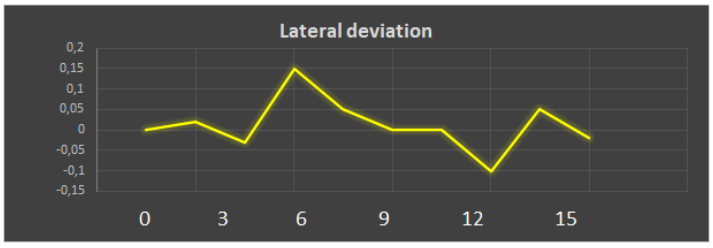


Figure 6.14: Vehicle's lateral deviation

Chapter 7

Conclusions and Future Works

This thesis work explored the complete process of building an autonomous vehicle from a 1:10 RC car, starting from the choice of its components till testing the final prototype on a real track. After a preliminary analysis of the working principle of a RC model, the choice of hardware necessary for the completion of all the tasks of perception, control and actuation has been described. Particular focus was placed on the selection of the on-board SBC, Raspberry Pi 4B, which can manage all sensors and actuators by means of a dedicated MATLAB and Simulink Support Package.

Subsequently, the working principle of the perception pipeline has been explained. A monocamera was used to perform lane detection and extract variables such as road curvature, and vehicle's lateral deviation and heading error, while an IMU allowed the knowledge of linear accelerations and angular velocities.

A mathematical description of the model was provided, exploiting grey-box estimation techniques and CAD modeling to identify all the parameters of the plant. The latter was then used to implement a Kalman filter and a Stanley lateral controller for the vehicle.

After a complete analysis of the actuation system, composed by a servo and a DC motor, the vehicle performance was tested on a 20m curvy track, where the vehicle showed the capability of driving autonomously under different light conditions and in the presence of shadows due to three leaves or of lighter marks on the road surface. Setting a longitudinal speed of 0.75m/s the vehicle successfully recognized curves of up to 8m of radius in both direction, also providing a good estimation of heading error

and lateral deviation. In this conditions it efficiently went trough the whole 80cm wide track, always staying at least 10cm far from the closest lane marker. The same result was achieved increasing the velocity to 1m/s for radii of up to 15m and at 2m/s for less curvy roads.

In conclusion, it can be stated that the initial targets of the project have been successfully met, leading to the creation of a platform with autonomous capabilities that can be used to test complex algorithms in a safe and cheap environment.

Possible developments of this project are countless but a few interesting improvements can be indicated for future works. First, a longitudinal controller can be implemented such that vehicle speed is regulated depending on road curvature. It could be useful to make the computer vision algorithm independent from the other parts of the control loop, in order not to lose information which could be provided at higher frequency. This could also be done exploiting a more powerful SBC dedicated to image processing, which could as well provide the hardware necessary to perform road signal detection. The vehicle can, and it likely will, be used to test path planning and control algorithms intended for full scale cars. All the bases are set to implement a Model Predictive controller and, with a grater effort, also neural networks could be used for control purposes. Finally, one or more twin vehicles can be built to test V2V communication on different real world scenarios reproduced in small scale.

List of Figures

1.1 Newest MIT RACECAR platform [17] . . . 12
1.2 Architecture of an autonomous system [23] . . . 13
1.3 Sensors and their range in an autonomous vehicle [24] . . . 13
1.4 Ideal detection from a 3D LiDAR[23] . . . 14
1.5 Ultrasonic sensor HC-SR04 . . . 15
1.6 GPS working principle [31] . . . 17
1.7 Mapping from World coordinates to pixel coordinates [34] . . . 17
1.8 Stereo camera [36] . . . 18
1.9 Stereo camera working principle [32] . . . 18
1.10 ISO 885-2011 reference frame [37] . . . 19
1.11 Vehicle top-view . . . 19
1.12 Bicycle model . . . 20
1.13 Tire deformation and slip angle . . . 21
1.14 Dynamical model . . . 22
1.15 Pure pursuit parameters . . . 23
1.16 Pure pursuit-arc construction . . . 24
1.17 Relation between steering angle and curve radius . . . 24
1.18 Effects of different ld on the trajectory . . . 25
1.19 Effect of Stanley controller on large heading error . . . 26

2.1 RC car scales . . . 29
2.2 RC car . . . 30
2.3 RC car without top cover . . . 31
2.4 Steering mechanism . . . 31
2.5 Working principle of a DC motor . . . 32
2.6 Brushed DC motor structure . . . 32
2.7 RC servo structure . . . 33
2.8 Scaled Autonomous Vehicle . . . 35
2.9 Raspberry Pi 4B GPIO pinout [43] . . . 36
2.10 Working principle of RC car . . . 36
2.11 Working principle of car guided by keyboard . . . 37
2.12 Working principle of autonomous car . . . 37

3.1 3D design of camera support . . . 41
3.2 Pinhole camera model [44] . . . 41
3.3 Radial distortion [34] . . . 43
3.4 Camera Calibrator App . . . 43
3.5 Radial distortion correction . . . 44

3.6 ROI, BEV and image binarization 44
3.7 Lane detection and inverse perspective transformation 45
3.8 Transition from vehicle point of view to external point of view 46
3.9 Lane detection output 46
3.10 LSM9DS1 sensor and Simulink block 47
3.11 IMU support 48
3.12 Gyroscope calibration offset correction 49
3.13 Accelerometer calibration 49
3.14 Correction of hard iron offset 50
3.15 Fixed and mobile reference frame 51
3.16 Computation of longitudinal velocity using integration 51
3.17 Application of Low-Pass Filter 52
3.18 Photoresistor 52
3.19 Photoresistor circuit 53

4.1 3D representation of the chassis 57
4.2 3D representation of car wheel 57
4.3 3D representation of power bank 57
4.4 3D representation of IMU Lego house 58
4.5 3D representation of the ensemble 59
4.6 Grey box model identification 60
4.7 Data collection model 60
4.8 Sample code of grey-box model estimation 61
4.9 Result of estimation using *greyest* 61
4.10 Simulink model for *Parameter Estimator App* 62
4.11 Estimation of $C_{\alpha f} = C_{\alpha r}$ using simplex search 63
4.12 Estimation of $C_{\alpha f} = C_{\alpha r}$ using real data 63
4.13 Kalman filter block scheme 65
4.14 Kalman filter block scheme 65
4.15 Kalman filter block scheme 65

5.1 Wiring of actuation line 67
5.2 Micro servo 68
5.3 Simulink servo block 68
5.4 Wheels position corresponding to servo input in degree 69
5.5 L298N DC motor driver 70
5.6 Characterization of DC motor: first set of measurements 72
5.7 Characterization of DC motor: mean values of the first set of measurements 72
5.8 Test in the other way with input 1 73
5.9 DC motor characterization: planar velocity 73
5.10 Characterization of DC motor: polynomial fitting 73

6.1 Path created on Driving Scenario Designer 75
6.2 Pure Pursuit Control scheme implemented in Simulink 76
6.3 Comparison between different *lookahead* distances 76
6.4 Steering profile 77
6.5 Actual trajectory Vs planned trajectory 77
6.6 Perception and Lateral Control 78

6.7 Camera parameters: distance ahead of sensor and space to each side . . 78
6.8 Camera parameters: camera height and pitch 79
6.9 Detection of a curvy road 79
6.10 Testing conditions of Stanley Lateral Controller 80
6.11 Perception output 80
6.12 Vehicle testing on a curvy track 81
6.13 Vehicle's behaviour in Driving Scenario Designer App 82
6.14 Vehicle's lateral deviation 82

List of Tables

2.1 RC car components 30
2.2 Scaled AV components 34

3.1 Camera technical specifications 40

4.1 Mechanical characteristics of the car's parts 58
4.2 Physical properties 59

5.1 Working principle of L298N module 71

6.1 Testing results of Stanley Lateral Controller 80
6.2 Loop duration 81

References

[1] Adam Ziebinski et al. "Review of advanced driver assistance systems (ADAS)". In: *AIP Conference Proceedings*. Vol. 1906. 1. AIP Publishing LLC. 2017, p. 120002.

[2] Peter Davidson and Anabelle Spinoulas. "Autonomous vehicles: what could this mean for the future of transport". In: *Australian Institute of Traffic Planning and Management (AITPM) National Conference, Brisbane, Queensland*. 2015.

[3] Daniel J Fagnant and Kara Kockelman. "Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations". In: *Transportation Research Part A: Policy and Practice* 77 (2015), pp. 167–181.

[4] Weber M. *Where to? a history of autonomous vehicles*. URL: `https://computerhistory.org/blog/where-to-a-history-of-autonomous-vehicles/`. (accessed: 25.08.2021).

[5] Robert E Fenton and Robert J Mayhan. "Automated highway studies at the Ohio State University-an overview". In: *IEEE transactions on Vehicular Technology* 40.1 (1991), pp. 100–113.

[6] Mark Peplow. *Robots rev up for Grand Challenge*. 2005.

[7] E. Blasch, A. Lakhotia, and G. Seetharaman. "Unmanned vehicles come of age: The DARPA grand challenge". In: *Computer* 39.12 (Dec. 2006), pp. 26–29. ISSN: 1558-0814. DOI: 10.1109/MC.2006.447.

[8] J Markoff. "Google cars drive themselves, in traffic". In: *The New York Times* 10 (Jan. 2010).

[9] Shantanu Ingle and Madhuri Phute. "Tesla autopilot: semi autonomous driving, an uptick for future autonomy". In: *International Research Journal of Engineering and Technology* 3.9 (2016), pp. 369–372.

[10] SAE International. *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*. Vol. J3016. 2016.

[11] Michael Taylor. "The level 3 audi a8 will almost be the most important car in the world". In: *Forbes. com* (2017).

[12] Todd Litman. *Autonomous vehicle implementation predictions*. Victoria Transport Policy Institute Victoria, Canada, 2017.

[13] WuLing Huang et al. "Autonomous vehicles testing methods review". In: *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2016, pp. 163–168.

[14] Michael G Bechtel et al. "Deeppicar: A low-cost deep neural network-based autonomous car". In: *2018 IEEE 24th international conference on embedded and real-time computing systems and applications (RTCSA)*. IEEE. 2018, pp. 11–21.

[15] Matthew O'Kelly et al. "F1/10: An open-source autonomous cyber-physical platform". In: *arXiv preprint arXiv:1901.08567* (2019).

[16] *Autonomous RC Cars*. URL: `https://www.mw.tum.de/en/ftm/main-research/intelligent-vehicle-systems/autonomous-rc-cars/`. (accessed: 25.08.2021).

[17] *MIT RACECAR*. URL: `https://racecar.mit.edu/`. (accessed: 25.08.2021).

[18] *About Donkey*. URL: `https://docs.donkeycar.com/`. (accessed: 25.08.2021).

[19] Edoardo Pagot, Mattia Piccinini, and Francesco Biral. "Real-time optimal control of an autonomous RC car with minimum-time maneuvers and a novel kineto-dynamical model". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 2390–2396.

[20] Alexander Liniger and John Lygeros. "Real-time control for autonomous racing based on viability theory". In: *IEEE Transactions on Control Systems Technology* 27.2 (2017), pp. 464–478.

[21] Myeon-gyun Cho. "A study on the obstacle recognition for autonomous driving RC car using lidar and thermal infrared camera". In: *2019 Eleventh International Conference on Ubiquitous and Future Networks (ICUFN)*. IEEE. 2019, pp. 544–546.

[22] Eugenio Alcalá et al. "Autonomous racing using linear parameter varying-model predictive control (lpv-mpc)". In: *Control Engineering Practice* 95 (2020), p. 104270.

[23] Scott Drew Pendleton et al. "Perception, planning, control, and coordination for autonomous vehicles". In: *Machines* 5.1 (2017), p. 6.

[24] *Sensor Integration*. URL: `https://novatel.com/industries/autonomous-vehicles`. (accessed: 25.08.2021).

[25] Juraj Kabzan et al. "Amz driverless: The full autonomous racing system". In: *Journal of Field Robotics* 37.7 (2020), pp. 1267–1294.

[26] Shahian-Jahromi Babak et al. "Control of autonomous ground vehicles: a brief technical review". In: *IOP Conference Series: Materials Science and Engineering*. Vol. 224. 1. IOP Publishing. 2017, p. 012029.

[27] Wan-Joo Park et al. "Parking space detection using ultrasonic sensor in parking assistance system". In: *2008 IEEE Intelligent Vehicles Symposium*. 2008, pp. 1039–1044. DOI: `10.1109/IVS.2008.4621296`.

[28] Minseok Ok, Sungsuk Ok, and Jahng Hyon Park. "Estimation of Vehicle Attitude, Acceleration and Angular Velocity Using Convolutional Neural Network and Dual Extended Kalman Filter". In: *Sensors* 21.4 (2021), p. 1282.

[29] Hui Guo and Huajie Hong. "Research on filtering algorithm of MEMS gyroscope based on information fusion". In: *Sensors* 19.16 (2019), p. 3552.

[30] M Bersani et al. "Vehicle state estimation based on kalman filters". In: *2019 AEIT International Conference of Electrical and Electronic Technologies for Automotive (AEIT AUTOMOTIVE)*. IEEE. 2019, pp. 1–6.

[31] Rabi Shrestha and Basant Awasthi. *How Nepal can be benefitted from different GNSS systems available worldwide*. Jan. 2021. DOI: 10.13140/RG.2.2.21231.05282.

[32] Abdelmoghit Zaarane et al. "Distance measurement system for autonomous vehicles using stereo camera". In: *Array* 5 (2020), p. 100016.

[33] Z. Zhang. "A flexible new technique for camera calibration". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.11 (2000), pp. 1330–1334. DOI: 10.1109/34.888718.

[34] MATLAB. *What Is Camera Calibration*. (accessed: 25.08.2021).

[35] Francisco Bonin-Font, Alberto Ortiz, and Gabriel Oliver. "Visual navigation for mobile robots: A survey". In: *Journal of intelligent and robotic systems* 53.3 (2008), pp. 263–296.

[36] *ZED Mini*. URL: https://store.stereolabs.com/products/zed-mini?_ga=2.161714364.1473817532.1629275287-70125954.1629275287. (accessed: 25.08.2021).

[37] Moad Kissai et al. "Adaptive Robust Vehicle Motion Control for Future Over-Actuated Vehicles". In: *Machines* 7 (Apr. 2019), p. 26. DOI: 10.3390/machines7020026.

[38] Stefano Feraco et al. "A local trajectory planning and control method for autonomous vehicles based on the RRT algorithm". In: *2020 AEIT International Conference of Electrical and Electronic Technologies for Automotive (AEIT AUTOMOTIVE)*. 2020, pp. 1–6. DOI: 10.23919/AEITAUTOMOTIVE50086.2020.9307439.

[39] R Craig Coulter. *Implementation of the pure pursuit path tracking algorithm*. Tech. rep. Carnegie-Mellon UNIV Pittsburgh PA Robotics INST, 1992.

[40] Gabriel M Hoffmann et al. "Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing". In: *2007 American Control Conference*. IEEE. 2007, pp. 2296–2301.

[41] *Extended Kalman filter*. URL: https://en.wikipedia.org/wiki/Extended_Kalman_filter. (accessed: 28.09.2021).

[42] *Servo (radio control)*. URL: https://en.wikipedia.org/wiki/Servo_(radio_control). (accessed: 19.09.2021).

[43] *Raspberry Pi Documentation*. URL: https://www.raspberrypi.org/documentation/computers/os.html. (accessed: 26.09.2021).

[44] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2nd ed. Cambridge University Press, 2004. DOI: 10.1017/CBO9780511811685.

[45] *findParabolicLaneBoundaries*. URL: https://it.mathworks.com/help/driving/ref/findparaboliclaneboundaries.html. (accessed: 30.09.2021).

[46] Iuri Frosio, Federico Pedersini, and N Alberto Borghese. "Autocalibration of MEMS accelerometers". In: *IEEE Transactions on Instrumentation and Measurement* 58.6 (2008), pp. 2034–2041.

[47] Joshua Hrisko. *Calibration of an Inertial Measurement Unit (IMU) with Raspberry Pi*. 2020. URL: https://makersportal.com/blog/calibration-of-an-inertial-measurement-unit-with-raspberry-pi. (accessed: 30.09.2021).

[48] Hans Pacejka. *Tire and vehicle dynamics*. Elsevier, 2005.

[49] *Estimate Coefficients of ODEs to Fit Given Solution*. URL: https://it.mathworks.com/help/ident/ug/estimating-coefficients-of-odes-to-fit-given-solution.html. (accessed: 27.09.2021).

[50] *Parameter Estimator*. URL: https://it.mathworks.com/help/sldo/ref/parameterestimator-app.html. (accessed: 27.09.2021).

[51] Eugenio Alcalá et al. "Autonomous racing using linear parameter varying-model predictive control (lpv-mpc)". In: *Control Engineering Practice* 95 (2020), p. 104270.

[52] Karl Johan Åström and Richard M Murray. *Feedback systems*. Princeton university press, 2010.