tecnun
Universidad
de Navarra

Self-Driving Car Autonomous System Overview

- Industrial Electronics Engineering - Bachelors' Thesis -

Author:
Daniel Casado Herráez
Thesis Director:
Javier Díaz Dorronsoro, PhD
Thesis Supervisor:
Andoni Medina, MSc

San Sebastián - Donostia, June 2020

"When something is important enough, you do it
even if the odds are not in your favor."

- *Elon Musk* -

To my Grandfather, Family, Friends & to my supervisor Javier Díaz

1. Contents

1.1. Index

1.2. Figures

1.1. Tables

1.2. Algorithms

2. Abstract

Research has made possible a continuous development of autonomous vehicles during the past decade. This project will provide an overview of the main technologies involved in autonomous driving, mentioning specific features that will make it suitable for the Formula Student Driverless competition. First, the sensors that capture data from the environment will be studied: LIDAR, camera, radar, GPS, IMU, odometry and their combination using sensor fusion. Second, object identification and localization will be explained with practical examples of classical methods (color thresholding and descriptor extraction), as well as modern techniques using convolutional neural networks. Third, the control components of the car will be analyzed, regarding the car model, path generation and optimization, and the vehicle controller. Finally, Formula Student driverless car examples will be presented with the goal of comparing the studied components with real life cases.

3. Introduction

3.1. Motivation

My main motivations to overview the technologies in an autonomous car have been: 1) the many companies in a continuous research and development environment and 2) the many technological fields that take part in a self-driving vehicle. Moreover, 3) I took an autonomous cars course during my exchange semester at the University of Michigan, leading to my becoming more interested in the topic. Furthermore, 4) I wanted to contribute my study to the Formula Student team as a there are plans to take part in the Driverless Vehicle competition class in the future.

1) On the one hand, even though autonomous vehicles have been around for many years now, it has been last decade that the race towards a fully functional self-driving vehicle has become tighter. Worldwide known companies such as Tesla, Waymo and Zoox have been leading the market with their own innovative solutions.

2) On the other hand, autonomous cars can be considered robots, and they cover a wide variety of fields ranging from image processing to controller design. This makes it an interesting topic for further specialization in one of those fields. Diving into each of these will help me further decide which path to take after finishing my master studies.

3) Regarding my time at the University of Michigan, the autonomous cars course was a graduate course where many of the key points in autonomous driving where explained. However, I did not get to understand how the components worked as a whole, and how they could be implemented. This made me become eager to deepen my understanding on this global vision of the car.

4) Finally, another of my main motivations in learning about unmanned vehicles was providing the Formula Student team from Tecnun-University of Navarra a background guide on the topic. This would help them in in the future so that they can segment their work and develop their own system for the vehicle. At first, the goal of the project was to start implementing self-driving techniques into it, however, I was lacking a general vision of how each of the components worked together.

In a nutshell, this overview should serve as an overall study of autonomous vehicles and their technologies, serving as a summary to the Formula Student team by introducing each of the concepts that are convenient to understand to develop a fully autonomous vehicle. Practical examples will be implemented to facilitate this understanding of the topic.

3.2. Objectives and Scope

The main goal of this project is to gather existing information of all the different components that make the autonomous component of a car. This will make possible studying each of the parts separately and knowing the available options that have already been developed.

Another objective is developing a summary that will serve the Formula Student team from Tecnun-University of Navarra when developing their self-driving model for the competition.

As for the scope of the project, it will include:

- Terms & definitions before each section for a better understanding of certain keywords
- Study of the control techniques used for autonomous driving:
    - Car models
    - Path planners
    - Controllers
- Study of the perception techniques used for autonomous driving:
    - Lane detection
    - Object detection
        - Implementation of classical methods
        - Implementation of modern convolutional neural networks
- Study of the sensors necessary for autonomous driving
- Recommend possible features for the Formula Student autonomous driving challenge.
- Review existing autonomous self-driving Formula Student solutions.
- Useful references to dive deeper into each topic.

And it will not include:

- In depth explanation and development of each self-driving method
- Implementation of autonomous driving algorithms in simulators or real life
- Available market models of physical components such as sensors
- A self-built Convolutional Neural Network
- Controller development from scratch
- Specific information about the Formula Student Driverless Vehicle challenge

3.3. Introduction to Autonomous Vehicles

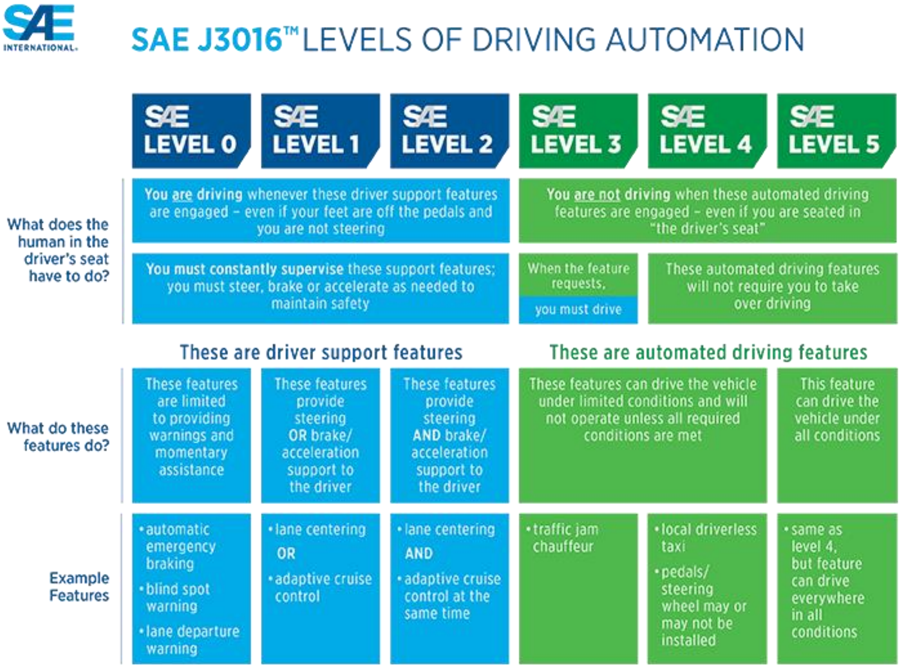
Autonomous vehicles have been researched since 1920, when they were first called 'phatom autos' [1]. These cars were impressive technological breakthroughs for the time, being remotely controlled through Morse keys and a radio signal. People saw the invention as the future of driving safety.

The Defense Advanced Research Projects Agency launched its first prototypes in the 1980s, driving for 600 meters. This same agency launched the 'DARPA Challenge' in 2004, that boosted the innovation from universities around this field. The challenge goal was to drive autonomously for 240km in the Mojave Desert region, and the winner would get $1 million prize. None of the participants managed to achieve the objective. A year later, five vehicles completed the 2005 Challenge of 212km off-road drive [2].

More inventions have become popular over time, and new companies have joined the challenge of creating a fully autonomous commercial vehicle. The most popular nowadays include Tesla, Waymo, Zoox and Aptiv. As well as branches of automotive companies such as Ford and Mercedes.

In 2017, Formula Student Germany introduced the new Driverless Vehicle class to their competition, allowing the teams to use a previously-built car adapted to autonomous driving. Currently 75 times have joined with top performing teams like TUfast from the Technical University of Munich and AMZ from ETH Zürich.

In addition, unmanned cars are grouped into five different levels of automation defined by the Society of Automotive Engineers (SAE), shown in *Figure 3-1*. The autonomous vehicles studied in this project will be between Level 3 and Level 5.



SAE J3016™ LEVELS OF DRIVING AUTOMATION

	SAE LEVEL 0	SAE LEVEL 1	SAE LEVEL 2	SAE LEVEL 3	SAE LEVEL 4	SAE LEVEL 5
What does the human in the driver's seat have to do?	**You are driving** whenever these driver support features are engaged – even if your feet are off the pedals and you are not steering			**You are not driving** when these automated driving features are engaged – even if you are seated in "the driver's seat"		
	You must constantly supervise these support features; you must steer, brake or accelerate as needed to maintain safety			When the feature requests, you must drive	These automated driving features will not require you to take over driving	
	These are driver support features			These are automated driving features		
What do these features do?	These features are limited to providing warnings and momentary assistance	These features provide steering **OR** brake/ acceleration support to the driver	These features provide steering **AND** brake/ acceleration support to the driver	These features can drive the vehicle under limited conditions and will not operate unless all required conditions are met		This feature can drive the vehicle under all conditions
Example Features	• automatic emergency braking • blind spot warning • lane departure warning	• lane centering **OR** • adaptive cruise control	• lane centering **AND** • adaptive cruise control at the same time	• traffic jam chauffeur	• local driverless taxi • pedals/ steering wheel may or may not be installed	• same as level 4, but feature can drive everywhere in all conditions

Figure 3-1: SAE Levels of Automation [1]

[1] *https://www.sae.org/news/press-room/2018/12/sae-international-releases-updated-visual-chart-for-its-%E2%80%9Clevels-of-driving-automation%E2%80%9D-standard-for-self-driving-vehicles*

To achieve autonomy in a car, a cyclic process illustrated in *Figure 3-2* needs to be carried out. First, the sensors (*Chapter 0*) will capture the surroundings and the vehicle state, then perception and localization (*Chapter 6*) will find the vehicle position with respect to its obstacles so that path planning and motion planning (*Chapter 7*) can take place. The desired route will be computed and sent to the car controller (*Chapter 7.2.*) This will generate the outputs that go to the actuators so that the car can respond to the environment. All these topics except the actuators will be introduced in the project.

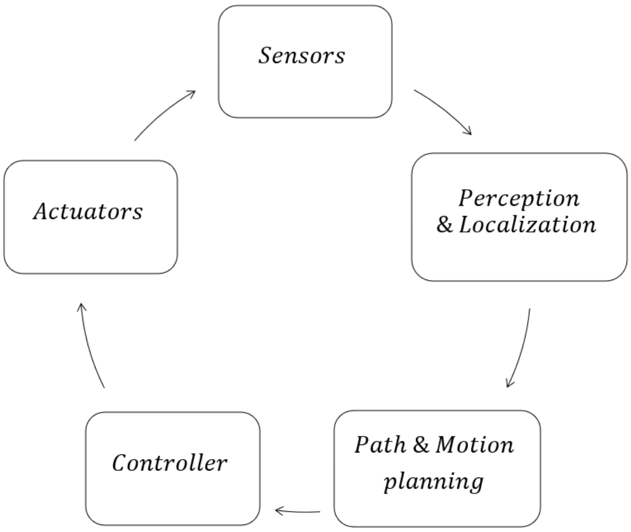


Figure 3-2: Autonomous driving steps

4. Project timeline and Gantt chart

The project was divided in four different work packages spanning from the 18th of February to the 14th of June of 2020:

WP1: Control

- Car model: 24th February – 1st March 6th -12th April
- Path generation: 2nd – 29th March
- Controllers: 18th February – 1st March; 30th March – 12th April

WP2: Perception

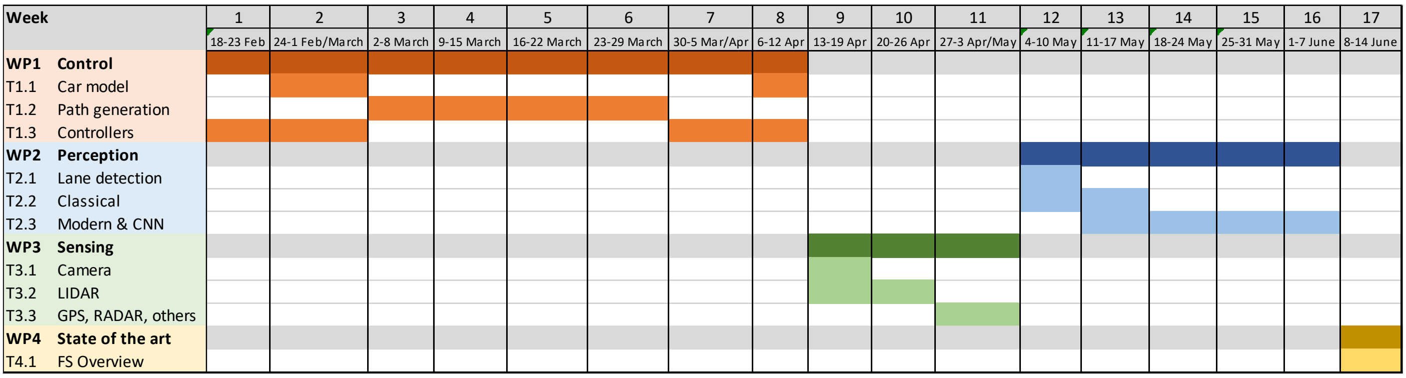
- Lane detection: 4th – 10th May
- Classical object detection: 4th – 17th May
- Modern object detection: 11th May – 7th June

WP3: Sensing

- Camera: 13th - 19th April
- LIDAR: 13th -26th April
- GPS, RADAR, and other sensors: 27th April – 3rd May

WP4: Introduction and other Formula Student cases

- Formula Student existing Autonomous Vehicles: 8th – 14th June
- Abstract, introduction, motivation: 8th – 14th June



Week		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
		18-23 Feb	24-1 Feb/March	2-8 March	9-15 March	16-22 March	23-29 March	30-5 Mar/Apr	6-12 Apr	13-19 Apr	20-26 Apr	27-3 Apr/May	4-10 May	11-17 May	18-24 May	25-31 May	1-7 June	8-14 June
WP1	**Control**	■	■	■	■	■	■	■	■									
T1.1	Car model		■						■									
T1.2	Path generation			■	■	■	■											
T1.3	Controllers	■	■					■	■									
WP2	**Perception**												■	■	■	■	■	
T2.1	Lane detection												■					
T2.2	Classical												■	■				
T2.3	Modern & CNN													■	■	■	■	
WP3	**Sensing**									■	■	■						
T3.1	Camera									■								
T3.2	LIDAR									■	■							
T3.3	GPS, RADAR, others											■						
WP4	**State of the art**																	■
T4.1	FS Overview																	■

5. Sensing

The first step for autonomy is being able to sense the surroundings. This data is essential to position the vehicle and detect obstacles. There is a wide variety of sensors that can be used for this purpose. The most used ones will be studied: LIDAR, camera, radar, GPS, IMU and odometry encoders.

5.1. LIDAR

Terms & Definitions:

K-Nearest Neighbor: K-Nearest Neighbor is a simple statistical algorithm useful for classification problems. It stores all the available points, and classifies new ones based on a predefined criterion, which is usually the distance. This distance between the new case and the K nearest neighbors will be measured, and the new case will be assigned to the class of the closest neighbors. A graphical example is shown in *Figure 5-1*.

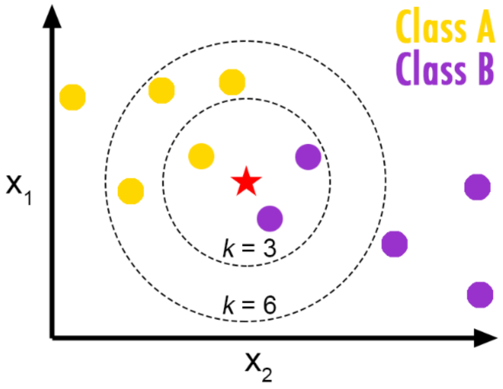


Figure 5-1: K-NN example[2]

5.1.1. LIDAR sensor

Light Detection and Ranging devices are very powerful tools for autonomous driving by providing a point map of the surroundings that contain distance data. Subsequently, LIDAR technology should conveniently be used in the Formula Student vehicle.

The main principle of LIDARs is to compute the depth of the environment by measuring the time difference between the emission of a light signal, and the receival of the reflected wave. This leads the engineer to think that it must be formed by two components, the light source, and the photodetector, that will determine the range of the sensor.

In practical terms, wavelengths of 1550nm and 905nm are commercialized, the former offers a greater distance measurement, more power, but higher costs, complexity, and the laser utilized might damage human sight, while the latter will have decent ranges at relatively low prices.

Another important parameter to consider is the Field of View (FOV) of the LIDAR. Which will be highly horizontal for 2D scanners, or horizontal and vertical for 3D sensors. Usually, the more directional the LIDAR, the larger the range. In other words, a sensor with a very small FOV will have a greater range than one that maps all the surroundings in a smaller radius. This is called the range-FOV tradeoff *Figure 5-2*.

[2] *https://medium.com/@equipintelligence/k-nearest-neighbor-classifier-knn-machine-learning-algorithms-ed62feb86582*

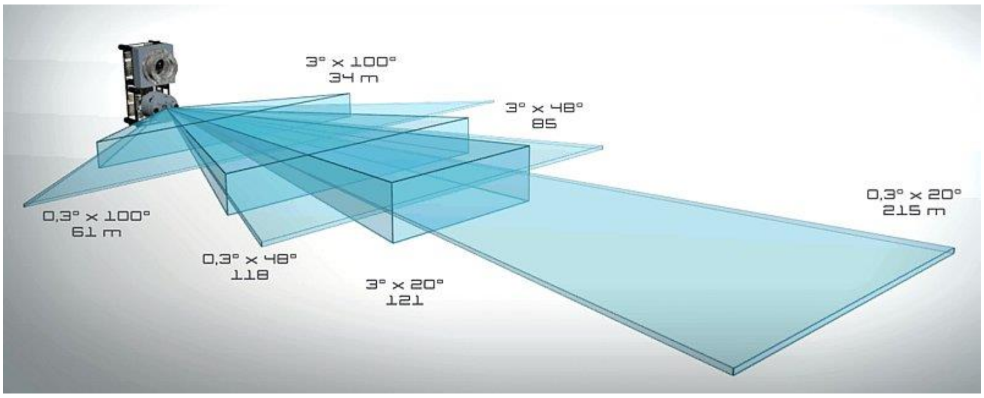


Figure 5-2: LIDAR Field of View [3]

Finally, the rotation frequency is also to be considered, as if the car is moving at high speeds, the variation between one laser scan and the next one will become larger.

The main question in the LIDAR section revolves around what does the sensor outputs, and how it can be used for object detection. In the context of a Formula Student racecar, how the LIDAR is relevant for cone detection. *Figure 5-3* shows the steps to extract and process the points from the environment to detect objects. An open source library with various features that will run through most of the steps of the diagram is PCL, from PointClouds.org [3].

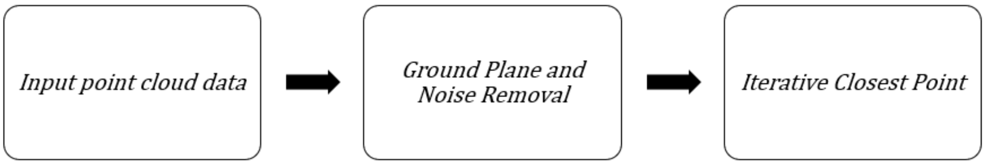


Figure 5-3: LIDAR point cloud processing steps

5.1.2. Point clouds

The LIDAR sensor output will contain the information of the surroundings related to their location in the form of point clouds (*Figure 5-4*). Let us take the example used by Tomáš Trafina in his bachelors thesis [4], where he studies the Velodyne VLP-16 scanner. In section 3.3 of his paper, he presents the output data packets of the LIDAR, showing that the main point cloud data obtained are, the timestamp, distance to the object, azimuth angle, vertical angle (each channel represents and angle with a 2º difference) and the intensity (*Figure 5-5*).

[3] *http://grauonline.de/wordpress/?page_id=1233*

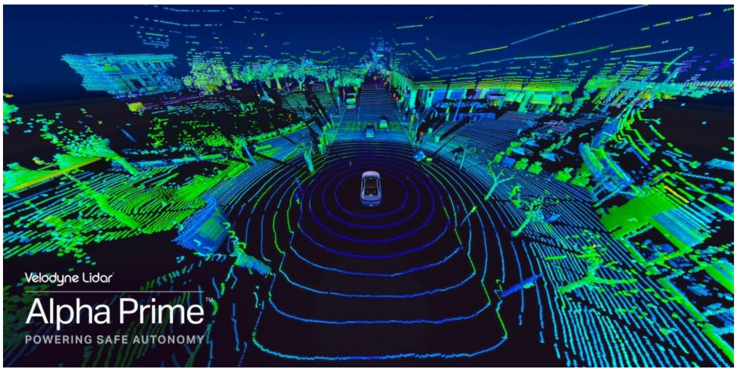


Figure 5-4: LIDAR point cloud example [4]

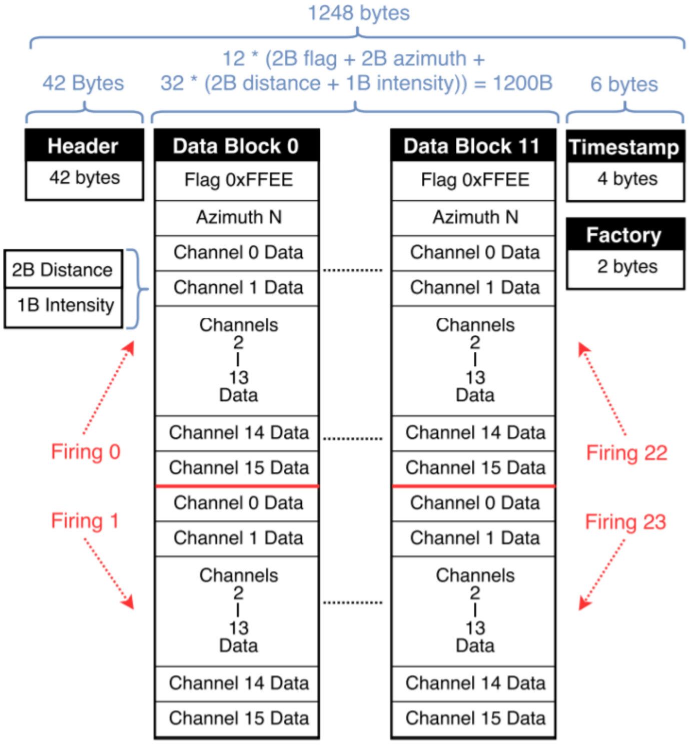


Figure 5-5: VLP-16 LIDAR output data [4]

After having the point set, the coordinates are based on the distance, azimuth, and vertical angle, so it is convenient to transform data to cartesian coordinates before processing (*Figure 5-6*).

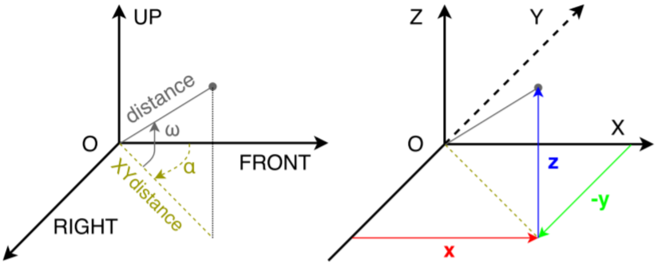


Figure 5-6: LIDAR coordinates vs Cartesian coordinates [4]

[4] *https://velodynelidar.com/products/alpha-prime/*

Without considering the center offset between the light transmitter and receiver, the following transformation needs to be applied:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} r \cdot \cos(\alpha) \cdot \cos(\omega) \\ r \cdot \sin(\alpha) \cdot \cos(\omega) \\ r \cdot \sin(\alpha) \end{bmatrix} \tag{1}$$

This will return the xyz local coordinates of the points in the set.

Accordingly, in order to collect all the data points in a single structure, a simple approach is to group them in a list or matrix such that:

$$P = [p_1 \quad \cdots \quad p_n] = \begin{bmatrix} x_1 & \cdots & x_n \\ y_1 & \cdots & y_n \\ z_1 & \cdots & z_n \end{bmatrix} \tag{2}$$

5.1.3. Ground plane and noise removal

The second step in point cloud management is removing the ground plane and noise so that it does not interfere with distance to object measurements.

There is a wide variety of algorithms that will help solve this problem, and Matlab Machine Learning Toolbox can be used for experimentation using K-Nearest Neighbors, Decision Trees etc. A paper published by Spanish researchers [5] briefly explains and compares these methods.

A fast method for ground plane detection called fast segmentation was published by Himmelsbach in 2010 [6]. This algorithm has been used by AMZ in their autonomous Formula Student car.

The main idea behind it is to segment 3D space into 2D segments of equal size. The LIDAR points will be compared to these reference lines. If they are below them, they are labelled as ground points, and will be deleted from the point cloud. If they are above it, they are non-ground points and they are kept (*Figure 5-7*).

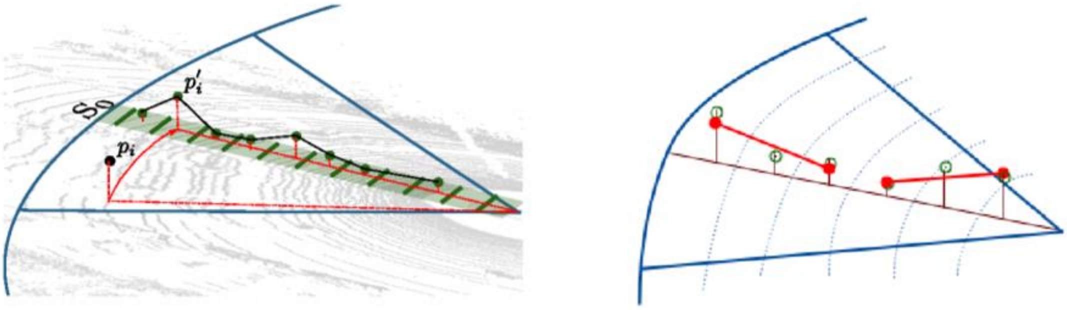


Figure 5-7: Fast segmentation for ground removal [6]

5.1.4. Iterative Closest Point (ICP)

Iterative Closest Point has the main goal to align two point clouds (*Figure 5-8*). This can be used either to align the depth map computed by the stereo cameras and the LIDAR point cloud, or for the vehicle's localization aligning two point clouds at different timesteps. Variations of the Iterative Closest Point exist, such as weighted ICP [7] and Trimmed ICP [8].

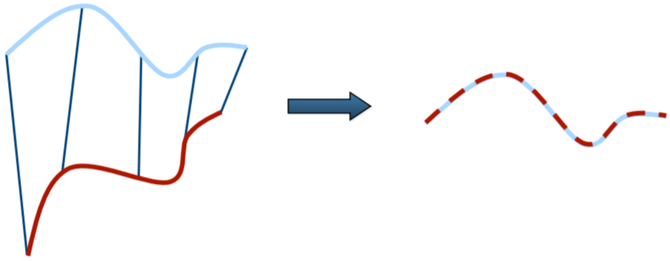


Figure 5-8: ICP alignment [5]

The Iterative Closest Point algorithm has the following steps, listed in the slides 'Introduction to Mobile Robotics: Iterative Closest Point' by professors at Mason University [9].

1) Initialize error between point clouds to infinite.
2) Calculate correspondence of the points:

The correspondence can be calculated by using closest point search algorithms like K-Nearest Neighbor. The Euclidean distance between point clouds is:

$$d(p_1,p_2) = ||p_1 - p_2|| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} \quad (3)$$

3) Calculate rotation and translation matrices via SVD.

Singular Value Decomposition (SVD) in linear algebra is a way of representing a matrix $W \in \mathcal{R}^{nxn}$ such that $W = U \cdot \sum \cdot V^T$, with $U, V \in \mathcal{R}^{nxn}$, and $\sum \in \mathcal{R}^{nxn}$ being the diagonal matrix with the singular values of W.

Let $\mathcal{P} = \{p_1, p_2 \dots, p_n\}$ and $Q = \{q_1, q \dots, q_n\}$ be the two sets of points in $\mathcal{R}^{3x3}$, and it is assumed that both have the same number of points, N.

The optimization problem can be formulated as:

$$\begin{aligned} &\underset{R,t}{\text{minimize}} \quad \frac{1}{N} \cdot \sum_{i=1}^{N} ||R \cdot p_i + t - q_i||^2 \\ &\textit{subject to} \\ &\qquad i = 1,2, \dots, N \end{aligned} \quad (4)$$

The center of mass of each point set is defined as:

$$\begin{aligned} \mu_p &= \frac{1}{N} \cdot \sum_{i=1}^{N} p_i \\ \mu_q &= \frac{1}{N} \cdot \sum_{i=1}^{N} q_i \end{aligned} \quad (5)$$

[5] *http://ais.informatik.uni-freiburg.de/teaching/ss11/robotics/slides/17-icp.pdf*

Before calculating the transformation, in order to shift all the points to a zero center of mass, the following subtraction is done to each of the points in the set:

$$\begin{aligned} \mathcal{P}' &= \{p_1 - \mu_p\} = \{p_1{}'\} \\ Q' &= \{q_1 - \mu_q\} = \{q_1{}'\} \end{aligned} \tag{6}$$

The matrix $W \in \mathcal{R}^{3x3}$, called the cross covariance matrix, can be formed:

$$W = \sum_{i=1}^{N} p_i \cdot q_i^T \tag{7}$$

Finally, the SVD decomposition of W is represented as:

$$W = U \cdot \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{bmatrix} \cdot V^T \tag{8}$$

$U, V \in \mathcal{R}^{3x3}$, and $\sigma_1, \sigma_2, \sigma_3$ are the singular values of W.

If $rank(W) = 3$, the solution to the optimization problem is unique and given by:

$$\begin{aligned} R &= U \cdot V^T \\ t &= \mu_p - R \cdot \mu_q \end{aligned} \tag{9}$$

Where $R \in \mathcal{R}^{3x3}$ and $t \in \mathcal{R}^{3x1}$ are the rotation and translation matrices.

4) Apply the transformations.

At step k:

$$\mathcal{P}_{k+1} = R_k \cdot \mathcal{P}_k + t_k \tag{10}$$

5) Compute error.

After the Singular Value Decomposition, the error is given by:

$$\sum_{i=1}^{N} (\left||p_1'|\right|^2 + \left||p_2'|\right|^2) - 2 \cdot (\sigma_1 + \sigma_2 + \sigma_3) \tag{11}$$

6) If error is greater than the threshold, restart from 2).

5.2. Camera

Cameras are a crucial sensor for autonomous vehicles, as they will capture the surroundings to detect objects, people, and obstacles. In the Formula Student racecar, cameras will be important to identify cones from the track and measure their distance.

Class notes from the University of Michigan make a distinction between two camera types based on the use of lenses (*Figure 5-9*): pinhole cameras and lensed cameras. 1) Pinhole cameras are based on focusing the light on a single point and directing this to the camera film. However, this limits the light levels that can be measured. 2) Cameras with lenses will focus the light on a point in the plane converging at a focal distance f. The main problem that must be addressed here is the concept of distortion (*Figure 5-10*). Therefore, it is necessary to calibrate the camera and obtain the camera matrix, that will reduce distortion when multiplying it by the image pixels.

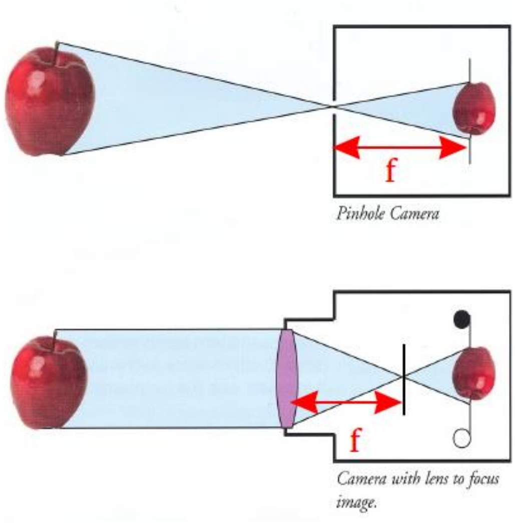


Figure 5-9: Pinhole camera (top) and lensed camera (bottom) [6]

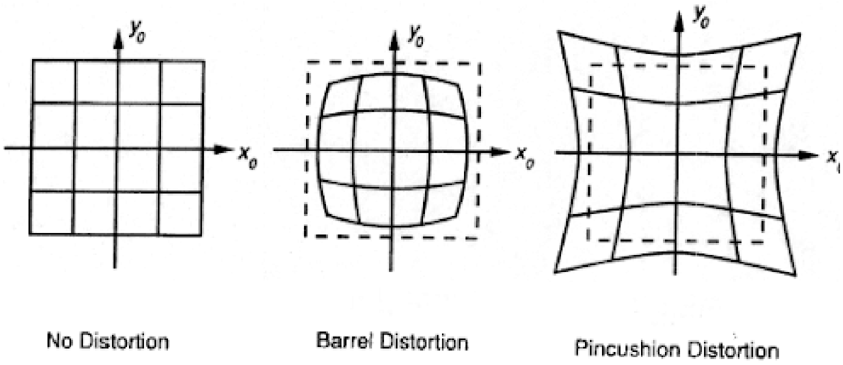


Figure 5-10: Types of distortion [7]

[6] *https://momofilmfest.com/depth-of-field-the-basics-explained/*

[7] *http://www.baspsoftware.org/radcor_files/hs100.html*

Once the camera is calibrated, the next issue to be solved is what is called "The Toy City Problem". It is based on the following hypothesis: if a small camera is placed in a toy city, everything will look huge. There is no quantitative way of knowing the measurements of the toy and the real city. Someone might have first thought that the idea of assigning a number of pixels to be a certain distance, but this is tricky, because a small car placed closer to the autonomous car will look bigger than a truck far away from it.

In order to obtain depth information, more than one image from a single object is necessary. There are two ways of tackling this problem. The first one is by using a monocular camera. Costs are reduced because less hardware is needed, but it might be harder to implement. One well-known option for monocular mapping is Structure from Motion (SFM), in which many pictures of an object taken at different perspectives build a depth map of that object (*Figure 5-11*). Additionally, and based on SFM, researchers from KTH in Sweden have demonstrated how to implement monocular vision for Simultaneous Localization and Mapping (SLAM) [10].

The second way of getting depth information is using two cameras separated a certain distance from each other, and stereo vision.

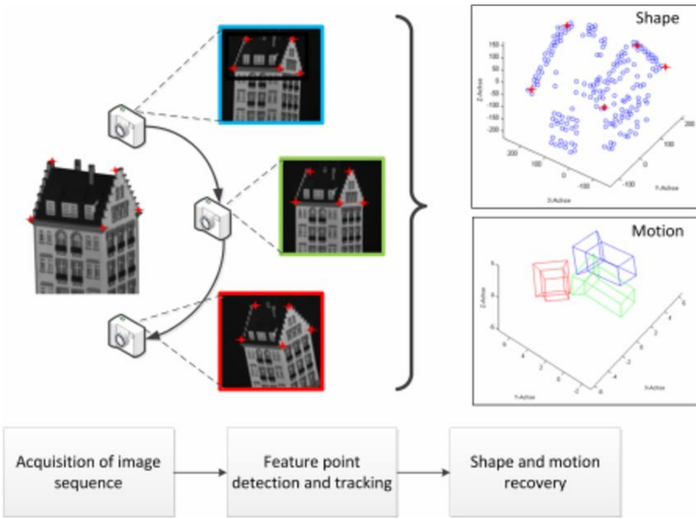

Shape
Motion
Acquisition of image sequence
Feature point detection and tracking
Shape and motion recovery

Figure 5-11: Structure from motion (SFM) [8]

5.2.1. Stereo Vision

Stereo vision comes into place when triangulation can be used for distance measurement. Stereo is used naturally by the human eyes. The main goal is to estimate the position of P given two observations O_1, O_2 (*Figure 5-12*). For this, two steps are required, to find the correspondence between images, and to use these observations to compute the distance. Due to the fact that only a small glimpse of the topic will be explained, the main sources refer to class notes from different universities [11] [12].

[8] *D. Aufderheide, W. Krybus, G. Edwards, " Inertial-Aided Sequential 3D Metric Surface Reconstruction from Monocular Image Streams" University of Bolton Conferences,*
Research and Innovation Conference, 2013.

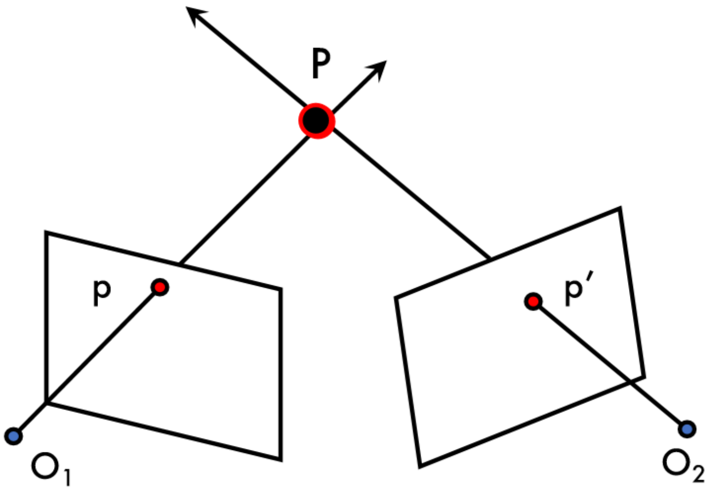


Figure 5-12: Stereo vision diagram [9]

It is convenient to make sure that the image planes are parallel, either by setting the cameras in a parallel manner, or by applying rectification. This will make the correspondence and triangulation problems easier (*Figure 5-13*).

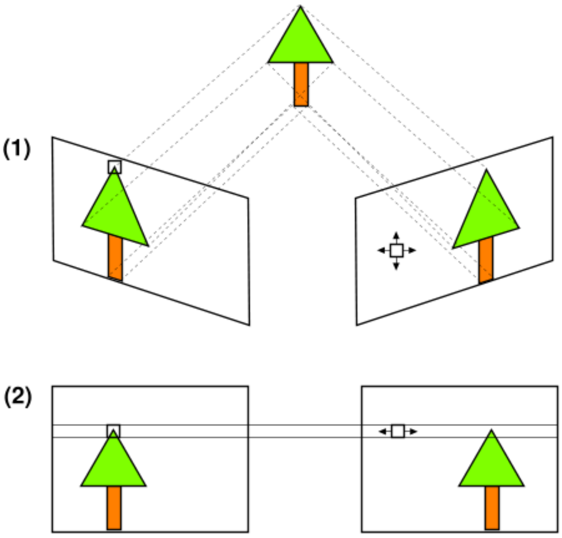


Figure 5-13: Epipolar planes make stereo easier [10]

[9] *University of Michigan EECS498: 'Self-Driving Cars: Perception and Control' Slides*

[10] *University of Michigan EECS498: 'Self-Driving Cars: Perception and Control' Slides*

5.2.1.1. Correspondence

The first stage before stereo image triangulation can be done is to identify the correspondence between the left and right images. Features will be matched, and the distance in both images will be compared so that the depth can be computed. Extracting the features of both images is the main challenge in this step. This difficulty comes from:

- Occlusions: Objects hiding other objects may interfere with the vision of the cameras. One object might be seen by one of the cameras while remain hidden to the other one.
- Fore shortening: Related to perspective distortion, in which depending on the way an object is observed, it may vary its shape.
- Baseline trade-off: The higher the $\frac{Baseline}{Distance\ to\ object}$ ratio, the higher the error in depth estimation.
- Homogeneous regions and repetitive patterns: Areas in the images that look almost the same are difficult to match. For example, two completely blue skies.

In order to detect the features, the goal is to identify interesting regions from images. These regions are called descriptors, and they can be extracted in three different ways:

1) Image Patches

This first method of extracting descriptors is based on identifying patch interest regions from images (*Figure 5-14*). Illumination and pose normalization can be performed to make the descriptors as generic as possible. However, image patches do not perform well when there are variations inside the same class, or when the pose is very different.

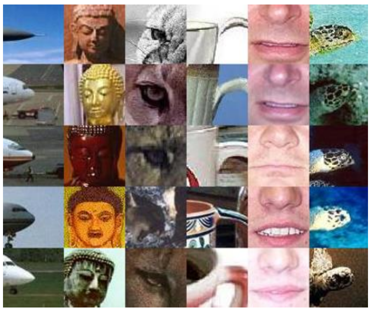


Figure 5-14: Examples of image patches [11]

2) Filters

Similar to the techniques used for lane detection in Javier Gonzalez's bachelors' thesis [13], it is possible to use edge detection for descriptor identification. An efficient way of doing this is by using Laplacian of Gaussian (LoG) [14] where derivatives of the pixels are computed to obtain blobs from edges smaller than a given size. Descriptor extraction using filters is a good way of dealing with pose changes and variations inside the same class.

[11] *University of Michigan EECS498: 'Self-Driving Cars: Perception and Control' Slides*

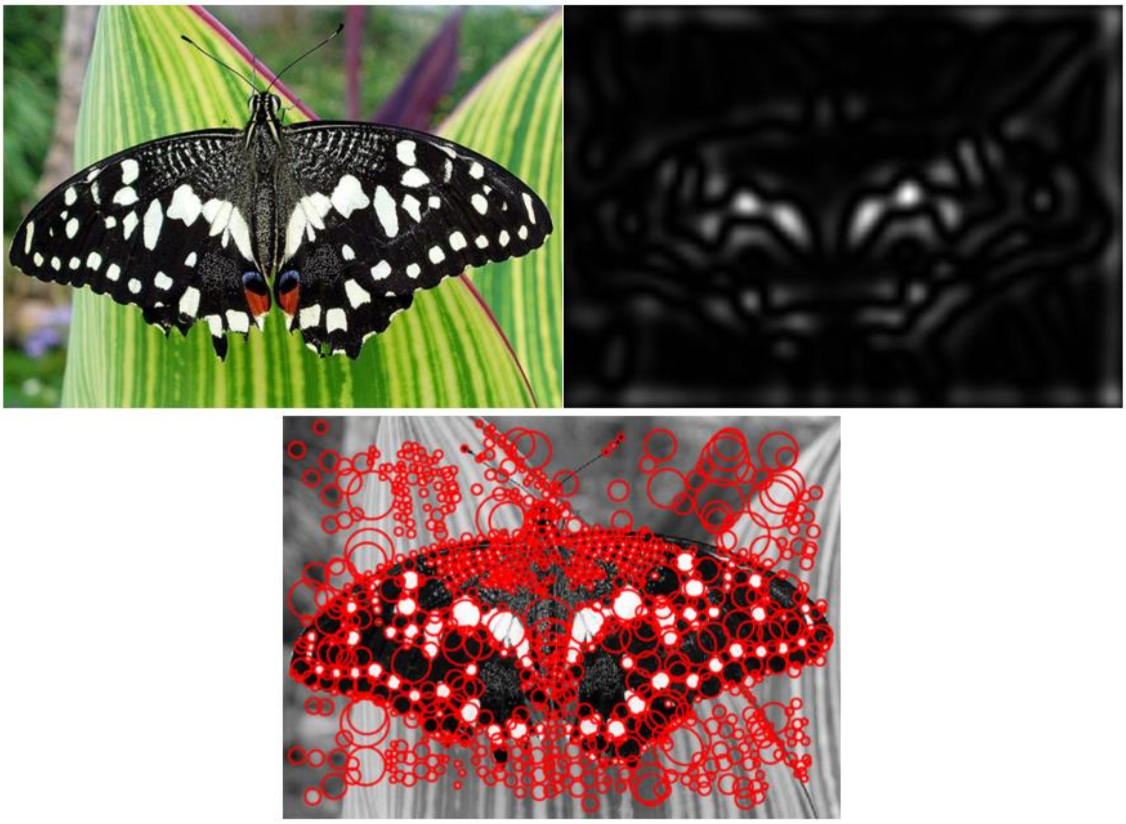


Figure 5-15: Descriptor extraction using filters. Original image (top left), Application of LoG (Top right) and image with descriptors (bottom) [12]

3) Descriptors

The best performing method for feature detection is by using descriptors that identify the most important points in the image. A popular descriptor algorithm is SIFT (Scale-Invariant Feature Transform), patented by David Lowe in 2004 [15]. It extracts scale and position invariant descriptors. Other alternatives to this exist such as ORB, BRIEF or SURF [16].

SIFT descriptors are robust and are not noticeably affected by pose variations, rotations, illumination, scale, and intra-class variability.

The SIFT approach is defined by first, creating a scale space of images and using Gaussian blur to find local extrema. Secondly, finding the histograms in windows of the gradient directions. And finally, creating a feature vector out of these histograms. A similar process is observed in *Figure 5-16*.

[12] *https://cvgl.stanford.edu/teaching/cs231a_winter1415/lecture/lecture10_detector_descriptors_2015_notes.pdf*

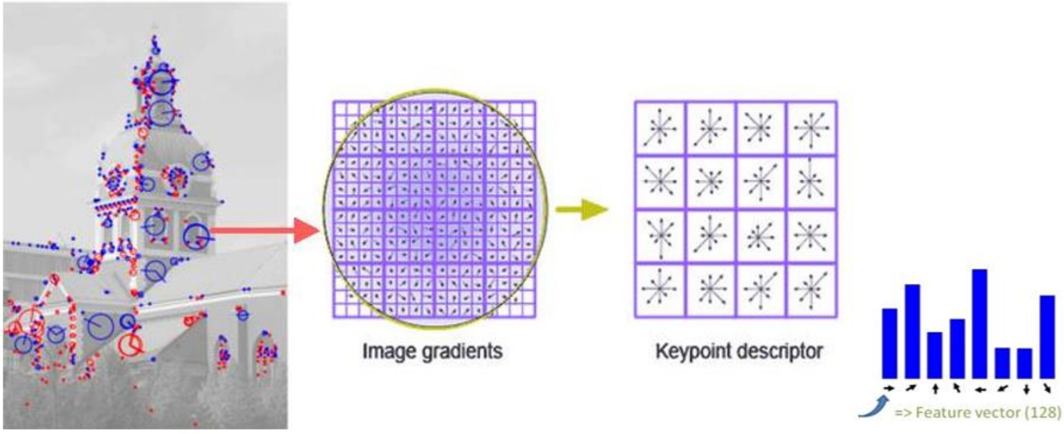


Figure 5-16: SIFT Descriptors [13]

5.2.1.2. Triangulation

Once the features have been found and matched in both images, the disparity d is computed by subtracting the point from the left image to the right image.

$$d = x - x' = \frac{B \cdot f}{z} \tag{12}$$

Knowing d, the baseline between the two cameras, and their focal length, the distance to the object z can be obtained. A visual representation is shown in *Figure 5-17*.

Furthermore, it can be observed that disparity is inversely proportional to depth, and the value z, distance between the cameras and the point P, is function of the baseline. This must be taken care of, since if the baseline is too small, the accuracy at longer distances will be noticeably reduced.

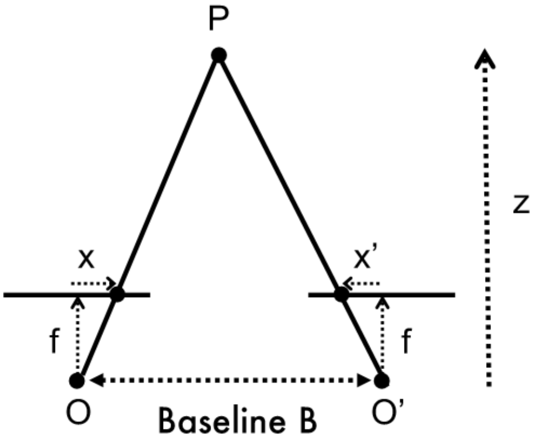


Figure 5-17: Stereo vision triangulation [14]

[13] *https://www.codeproject.com/Articles/619039/Bag-of-Features-Descriptor-on-SIFT-Features-with-O*

[14] *University of Michigan EECS498: 'Self-Driving Cars: Perception and Control' Slides*

5.2.1.3. Color

After overviewing the main concepts in cameras for autonomous driving, it is convenient to understand the characteristics of color and color spaces, since it is one of the core fields in image processing and feature extraction.

The color spectrum is obtained by the reflected light, the different wavelengths will result in a different color. Chromatic light can be characterized depending on:

- Radiance: Energy from the light source.
- Luminance: Perceived amount of light.
- Brightness: Achromatic intensity of light.

Human eye contains three receptors, that identify the three primary colors Red, Green and Blue, given by a certain wavelength (*Figure 5-18*). The colors can be defined by:

- Brightness: Achromatic intensity of light.
- Hue: Dominant wavelength in a mixture of light waves. It is represented in the radial scale between 0 and 360º where every color in the spectrum is located
- Saturation: Amount of white light mixed with hue. In other words, the saturation indicates how the color brightness refers to the brightness of the pixel

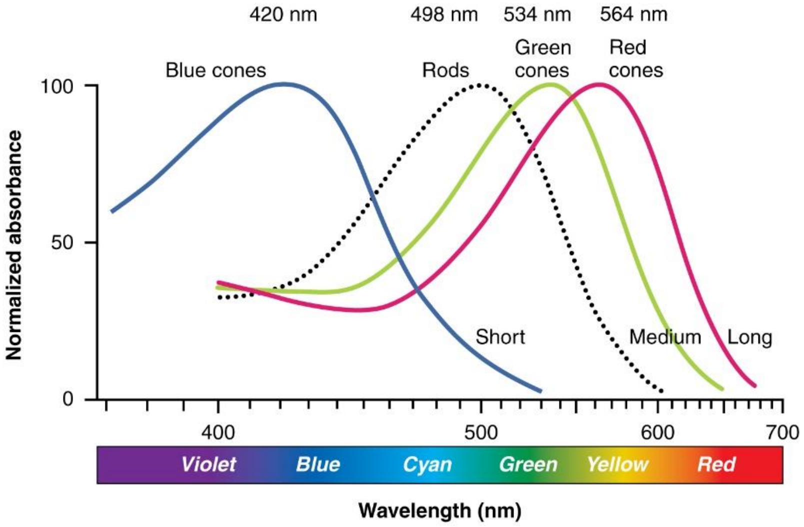


Figure 5-18: Visible light color wavelengths [15]

However, the combination of these cannot generate all visible colors, and that is why other color spaces have been tested and compared for image processing [17]. Some common examples are presented:

- HSI space (Hue Saturation Intensity, or HSB, or HSL): This model decouples intensity from color information. The main advantage it offers is when it comes to filtering. Using RGB, the program would need to filter all three levels for intensity, while in HSI only the Intensity layer would need to be filtered, improving efficiency.
- HSV space (Hue Saturation Value): HSV considers the value of the color. This refers to the darkness or lightness of a color.

[15] *https://www.iluminet.com/que-es-efecto-purkinje/*

- LAB space: LAB has been created as a good approximation to human vision. L refers to the lightness value, representing the darkest black at L = 0, and the brightest white at L = 100. A and B are the color channels. A refers to the green-red component while B refers to blue-yellow component. A=B=0 correspond to neutral gray a* = 0 and b* = 0.

Figure 5-20 shows the three channels of the color space in a road. The best channel or combination can be selected for lane detection or object identification.

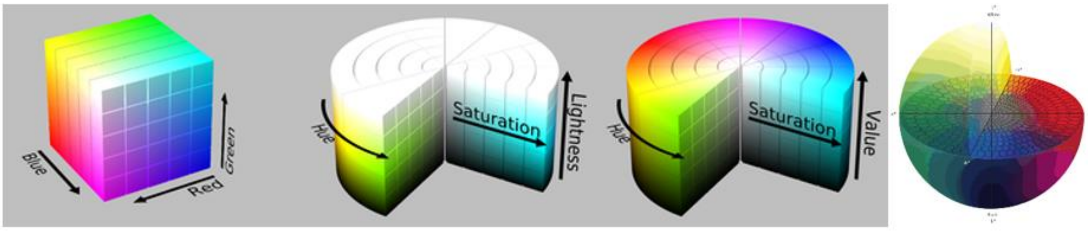


Figure 5-19: Color spaces. From left to right: RGB, HSL, HSV [16]*, LAB* [17]

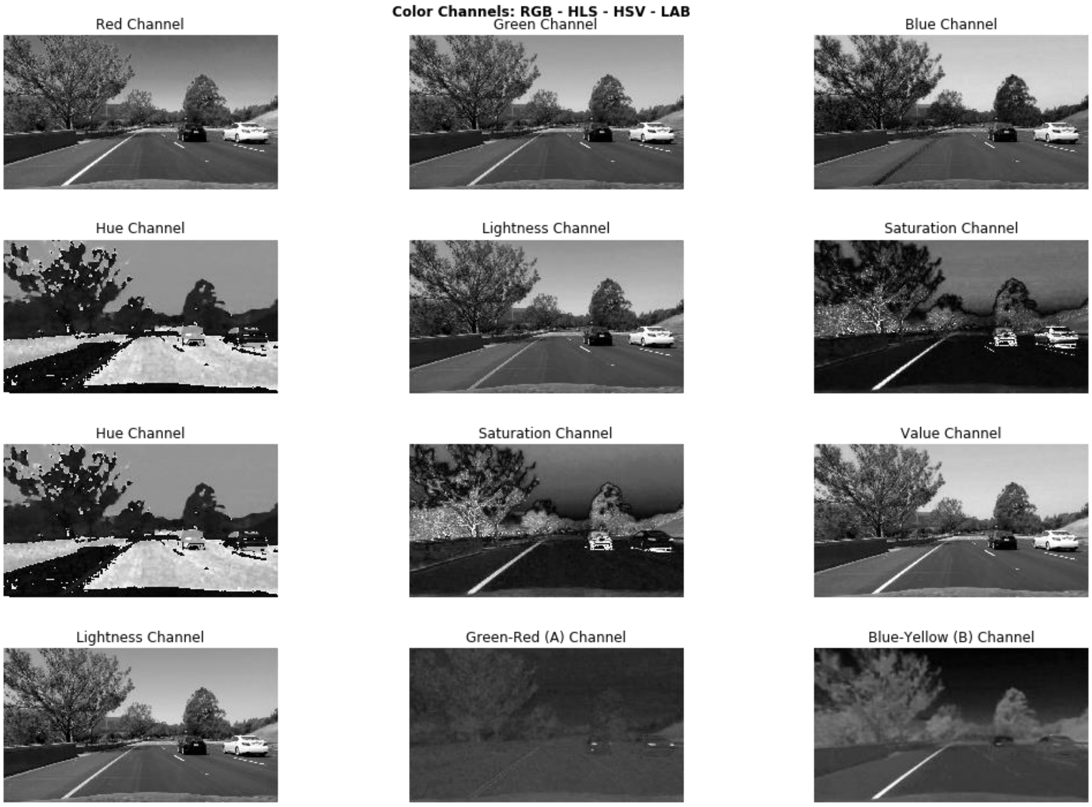


Figure 5-20: Color spaces applied to a road picture [18]

[16] *https://en.wikipedia.org/wiki/HSL_and_HSV*

[17] *https://www.xrite.com/blog/tolerancing-part-3*

[18] *https://es.slideshare.net/Yhat/finding-lanes-for-selfdriving-cars-pydata-berlin-jul-2017-ross-kippenbrock-of-yhat*

5.3. Radar

The radar is a radio frequency ranging sensor that works with a similar principle as the LIDAR, measuring the time difference between a signal sent and received. This signal will be electromagnetic waves in the microwave range instead of light rays. It also measures the frequency of the received wave to identify the object.

The main advantages of using radar technology are that it is not affected by atmospheric conditions, such as rain, snow, or dust because of the wavelength not being in the visible spectrum. Another beneficial point is range, which can vary depending on the application. In addition, the radar can easily measure the motion of surrounding objects. This last feature is especially useful when it comes to road driving, where other traveling vehicles must be considered.

Figure 5-21 shows the range profile of a radar modelled in a report by Ansys HFSS High Frequency Electromagnetic Field simulator [18]. It can be observed how the closest objects, the trucks and the light posts, are detected with a higher signal return, while the ones far in the distance are hardly noticeable.

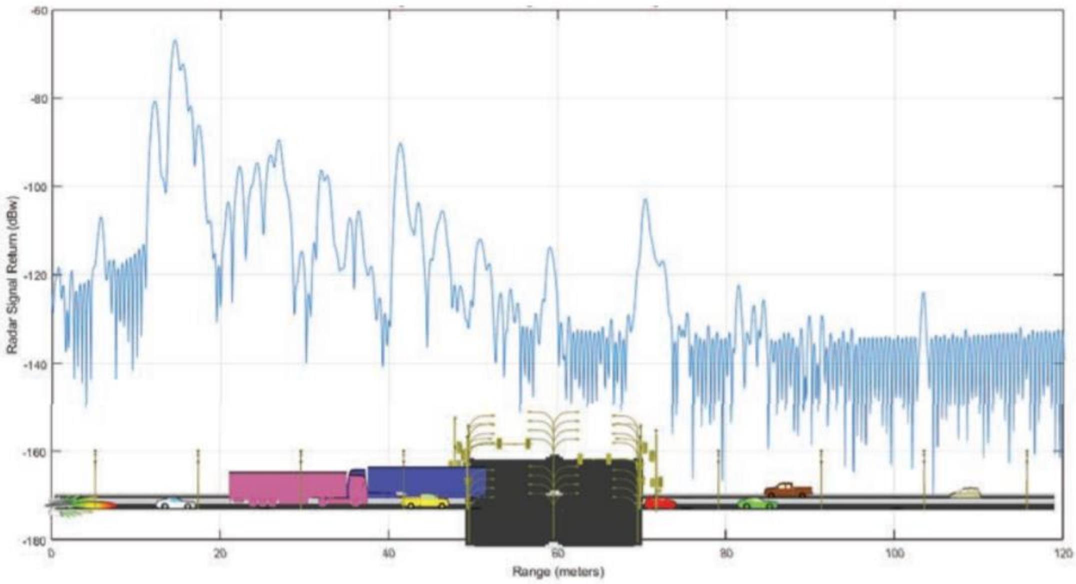


Figure 5-21: Radar signal in an intersection [18]

On the flip side, the signal accuracy of the radar tends to be low, so the best option is to use it in combination with other sensing technologies.

All in all, as in the Formula Student competition there are no moving obstacles, and good precision is required for racetrack following, radar should be only considered for future use in more advanced driving methodologies.

5.4. GPS/GNSS

Terms & Definitions:

Coordinate Systems: As mentioned in Meng's article aiming for a robust vehicle localization approach using sensor fusion [19], it is necessary to identify four coordinate systems to find the localization of a car:

1) Earth-Centered-Earth-Fixed (ECEF) coordinate system (*Figure 5-22, left*): Origin at the Earth center, positive Z axis points to the north pole, X axis goes along the Greenwich meridian, and Y is perpendicular and obtained with the right-hand rule.

2) Global coordinate system: It is defined either as North-East-Down (NED), where X points North, Y points East and Z points down, or North-East-Up (NEU), with X pointing East, Y pointing North and Z pointing up.

3) Body coordinate system (*Figure 5-22, right*): Depends on the body's coordinate system. In the case of the autonomous car, as it was defined by the Society of Automotive Engineers, X points forward, Y points right and Z points down.

4) Sensor coordinate system: The reference system in each of the sensors. Calibration is done so that the sensor and body coordinate systems coincide.

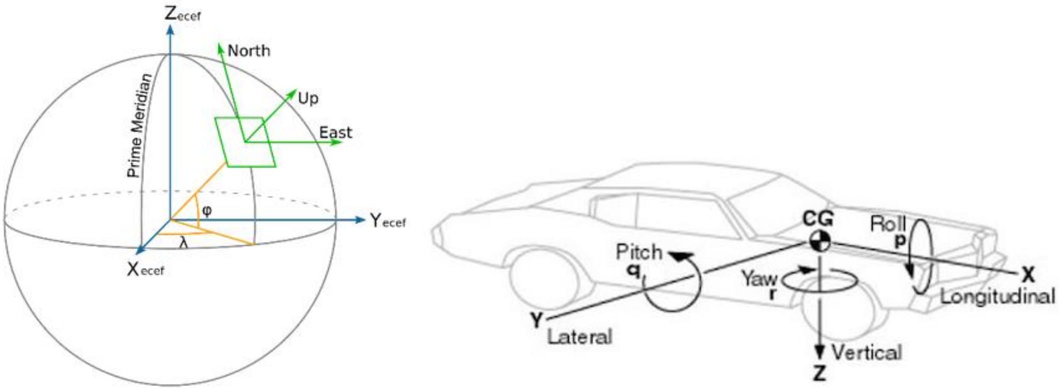


Figure 5-22: Earth reference frame [19] (left) and SAE vehicle coordinates (right) [20]

Global Positioning Systems give position and velocity information. The GPS system is divided into three segments: space, control, and end-user. The first one corresponds to a constellation of 24 satellites that move in fixed orbits around the Earth, each transmitting signals that will be decoded by the GPS receiver module. The control segment refers to the ground stations that communicate and control the satellites. They use radar to make sure the satellite is well positioned. And finally, the user segment, in which the GPS receiver identifies any visible satellites, through which it locates itself in the Earth. The kind of message that will be read contains information about the satellites, location, as well as the time difference between the receiver and each satellite.

[19] *http://www.dirsig.org/docs/new/coordinates.html*

[20] *Hashem Zamanian, Farid Javidpour , 'Dynamic Modeling and Simulation of 4-Wheel Skid-Steering Mobile Robot with Considering Tires Longitudinal and Lateral Slips' Department of Engineering, KAR University, Qazvin Branch, Iran*

Global Positioning System devices have noticeably improved their accuracy since they were first invented, centimeter precision can be obtained nowadays [20]. Most importantly, they don't have accumulative error, as they measure absolute position. However, update frecuencies are low, and GPS efectiveness depends on the number of satellites visible.

GNSS modules (Global Navigation Satellite System) are an alternative to GPS. They receive signals from more satellites, including the 24 from the GPS system, and are therefore more efficient in terms of finding available satellites.

The output data of the GPS/GNSS unit will usually be represented in ECEF coordinate system, latitude, longitude, and altitude. This must be converted to global coordinates. A precise formula for this, considering the curvature of the Earth, is given using the World Geodesic System (WGS 84, from 1984) [21].

However, for small distances the Earth might be considered round, leading to a simplified version of the coordinate transform:

$$\begin{aligned} X &= R \cdot \cos(\varphi) \cdot \cos(\lambda) \\ Y &= R \cdot \cos(\varphi) \cdot \sin(\lambda) \\ Z &= R \cdot \sin(\varphi) \end{aligned} \tag{13}$$

Where R is the Earth radius, latitude is denoted by φ, longitude by λ and altitude h.

5.5. IMU

Inertial Measurement Units (*Figure 5-23*) are electronic devices composed by accelerometers, gyroscopes and sometimes magnetometers. IMUs give information about the orientation and motion of the vehicle. As opposed to the GPS, IMUs have a high accuracy and update frequency. However, they are considered dead reckoning sensors, as each state depends on relative measurements and are computed upon previous states. This causes accumulative error. They can be used by the vehicle when no GPS signal is being received, and as a way of measuring the car acceleration for the control system.

It is important to consider the IMU coordinate system, which after calibration, must coincide with the car coordinate system. To transform it to global coordinates, the rotation matrices along the car's x axis (roll) and the z axis (yaw) are found using Euler angles:

$$R_{Roll} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos{(\varphi_r)} & \sin(\varphi_r) \\ 0 & -\sin(\varphi_r) & -\cos{(\varphi_r)} \end{bmatrix} \tag{14}$$

$$R_{Yaw} = \begin{bmatrix} \cos{(\varphi_y)} & \sin{(\varphi_y)} & 0 \\ \sin{(\varphi_y)} & -\cos{(\varphi_y)} & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{15}$$

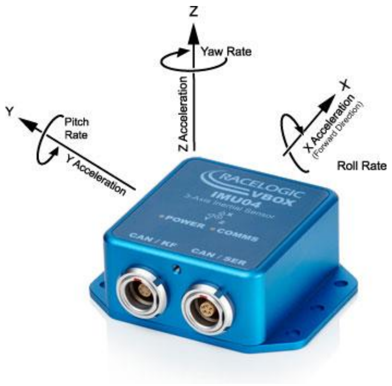


Figure 5-23: Inertial Measurement Unit (IMU) [21]

[21] *https://www.vboxautomotive.co.uk/index.php/es/products/modules/inertial-measurement-unit*

5.6. Odometry sensors

Odometry sensors are one of the most popular sensors in robotics used for many applications, as they estimate the change in position over time. They can be divided into incremental and absolute encoders.

The first ones will measure the position of the wheel attached to it by counting the pulses it generates. Incremental encoders are built with two or less lines with grooves, used to measure speed or detect the direction of motion. Nevertheless, incremental encoders generate error over time.

Absolute encoders will capture the exact wheel position when it is required. They are built using more groove channels than for the incremental encoders, and they detect the wheel position in a bit array, resulting in a very accurate measurement. However, neither of these sensors will measure wheel slippage which can increase the accumulative error values.

A common type of odometry encoders are optical encoders (*Figure 5-24*), based on light detection. A photo-diode emits light that goes through a disk with gaps at the same distance from each other. This light is detected by a photosensor, and wheel position is measured based on the number of times a light ray has been detected.

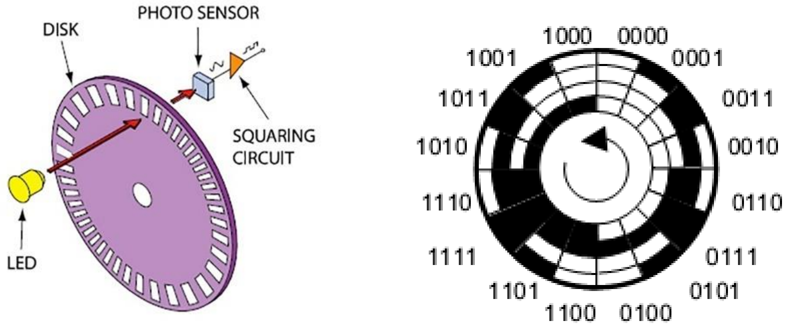


Figure 5-24: Optical incremental encoder [22] *(left) and absolute encoder (right)* [23]

[22] *https://www.analogictips.com/rotary-encoders-part-1-optical-encoders/*

[23] *http://www.incb.com.mx/index.php/component/content/article?id=3299:curso-de-electronica-electronica-digital-parte-1-cur5001s*

5.7. Sensor Fusion

Individual sensor implementation is a useful way of studying their performance and configuring them. However, as it has been observed, each sensor has specific advantages and disadvantages. Combining them (*Figure 5-25*) will provide a robust way of managing data. A brief explanation will be given about sensor fusion based on MATLAB provided resources [22].

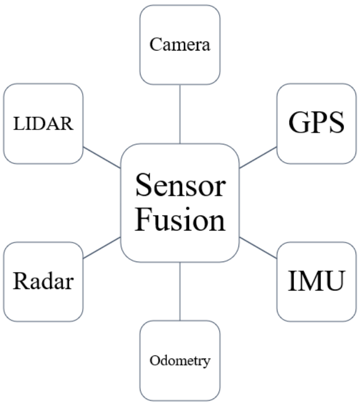


Figure 5-25: Generic sensor fusion diagram

There are four main ways this will help:

1) Increase the quality of the data: The noise and perturbances of each sensor will be compensated by the other sensors. As a simple example, if an accelerometer and a magnetometer are used together, the output could be averaged resulting in a cleaner output signal.
2) Increase reliability: Failure of a sensor may occur during operation; this will reduce the system performance, but the other sensors will still be outputting data.
3) Increase coverage area: Instead of averaging the output of the sensors, a coherent system map can be generated, and the area covered by each of the sensors can be complementary.
4) Reduce uncertainty: By comparing the different sensors, uncertainty will be reduced noticeably, increasing accuracy.

It is important to keep in mind that the main goal of sensor fusion in self-driving cars is to estimate the state of the vehicle. Article published by Federico Castanedo in Bilbao 'A Review of Data Fusion Techniques' [23] summarizes the existing data fusion techniques. Section 4 of the paper summarizes the most used ones: 1) Maximum Likelihood and Maximum Posterior, based on probabilistic theory, 2) Kalman Filtering, one of the most used approaches and has variations such as Extended Kalman Filter (EKF) and Unscented Kalman Filter (UKF). It is also popular in autonomous cars. 3) Particle Filter, that builds a density function based on random particle positioning, and 4) Covariance Consistency methods.

Something special about Kalman Filters is that they can incorporate the vehicle model. Therefore, they have two input states: the predictions made using the model and the measured position, e.g. use an accelerometer to measure motion and predict future states, and the recorded state updates e.g. GPS detects the actual position and makes corrections to reduce the error with the predicted state.

It is important to remember that the required coordinate transformations need to be performed for a good estimation between all the sensors.

6. Perception

Perception in self-driving vehicles has a similar meaning as it does for humans. It refers to the ability of being aware of the environment to interact with it. By using the sensors integrated in the car, it will be able to position itself and identify obstacles such as cars and people. This section will be used to describe two main types of perception problems that must be solved, road lane detection, and object detection. Simultaneous Localization and Mapping (SLAM) will also be covered briefly in this chapter.

6.1. Lane Detection

One of the most important tasks performed by a car's driver is lane detection. This is essential to follow the lane and give the vehicle the appropriate steering angle. Furthermore, even though it might not be of direct application to a Formula Student racecar, it is convenient to understand how it works, and it might be useful for other autonomous model car competitions.

Terms & Definitions:

Canny algorithm: Canny is an edge detection algorithm in which the image is first passed through a Gaussian filter for noise reduction, represented in *Figure 6-1 left*, and then a first derivative Gaussian filter is applied, shown in *Figure 6-1 right*.

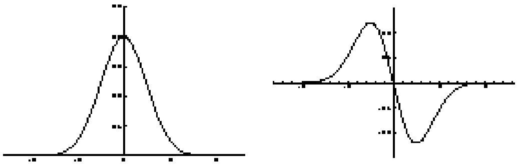


Figure 6-1: Gaussian filter (left) and Gaussian derivative (right) [24]

Gaussian Blur: Gaussian Blur is the process of applying a Gaussian filter (*Figure 6-1 left*) to the image. The result will be a very similar image with blurred edges.

A lane detection method introduced by Javier Gonzalez's bachelor's thesis [13] is summarized, with some additional suggestions.

After camera calibration has been done, the undistorted image will be obtained. The region of interest is extracted to discard data that does not contain information about the lane (*Figure 6-2*). The next step in Javier's Thesis would first transform the image to a top view of the lane, to perform edge detection. Afterwards the top view is obtained by applying a mattrix tranformation to the original image (*Figure 6-3*).

[24] *http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MARBLE/low/edges/canny.htm*

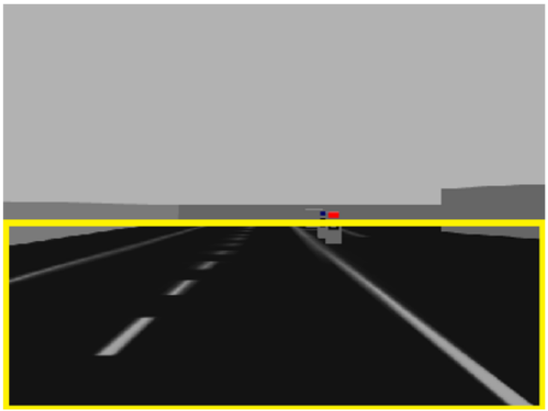


Figure 6-2: Region of Interest (ROI) [13]

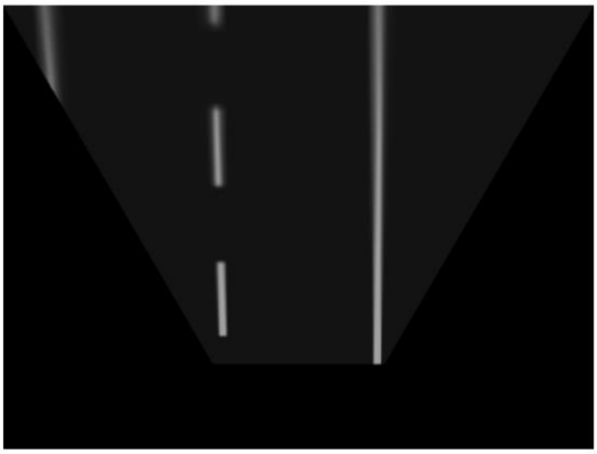


Figure 6-3: Top view of the road [13]

Thirdly, the aerial view image is converted into binary, noise is reduced using gaussian blur, and the edges are extracted. However, in real world binary thresholding might not be an efficient method as the variety of colors in a single image will be very large. Color thresholding can be done in several ways, while at the same time, transforming the picture to different color spaces.

Procedure from article 'Lane Detection for Autonomous Vehicles', by Sertap Kamçi et al. [24] gets the image through a conversion to HSV before Gaussian Blur and grayscale methods for the Canny edge detection algorithm to perform better. Testing should be done in order to obtain the best performance with the use of different color spaces (HSV, HLS…).

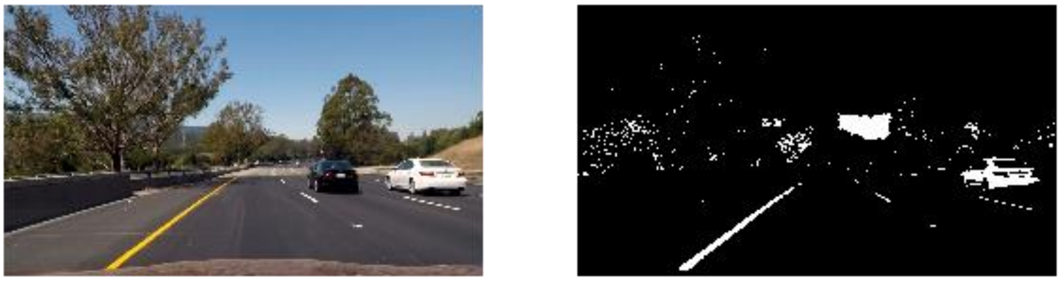


Figure 6-4: Image after removing distortion (left) and after applying HLS Color Space thresholds (right) [24]

Afterwards, lane detection is done with windowing techniques. Windows are placed along the lane and the centroid is computed. This centroid will indicate the point where the lane is located. Finally, a polynomial is fitted within the points that will indicate the lane function.

Several GitHub repositories exist that will facilitate the implementation and understanding of lane detection. A useful one is 'CarND-Advanced Lane Finder' [25] that will go from camera calibration to drawing the lane in the picture.

6.2. Object detection

The object detection process will find the objects in the image and classify them into classes. A wide range of research has been done in this field and it is convenient to divide the methods into the classical procedure, and the modern procedure.

Terms & Definitions:

Batch: A batch of images is defined as a group of images into which the training set is divided. The pictures are selected randomly. This will lead to the convolutional neural network learning generic features, instead of modifying its weights for each specific image.

6.2.1. Classical Object Detection

Traditional methods are algorithms and conditions applied directly over the image. Their main advantage is their simplicity of implementation. Two common classical methods are by color thresholding and using descriptors.

6.2.1.1. Color thresholding

Strictly referring to FS cone track detection, after extracting the Region of Interest just like in lane detection, and focusing only on the track, the simplest way of identifying the cones in the image can be by simply applying a color thresholder to the camera image.

MATLAB offers *colorThresholder* [26] feature as an interesting tool to experiment in different color spaces. Thresholds could be obtained manually to match the cones in the track. A sample image has been used for demonstration purposes of a segment of the Formula Student racetrack (*Figure 6-5, top*). It can be observed how after setting the RGB thresholds, the background can be erased to just keep the location of the cones in the track (*Figure 6-5, bottom*). However, the thresholds will vary depending on the environment conditions. For example, if the light is low, other thresholds would be required in order to successfully detect the cones. This results in a non-generic method for cone detection prone to failure not combined with other techniques.

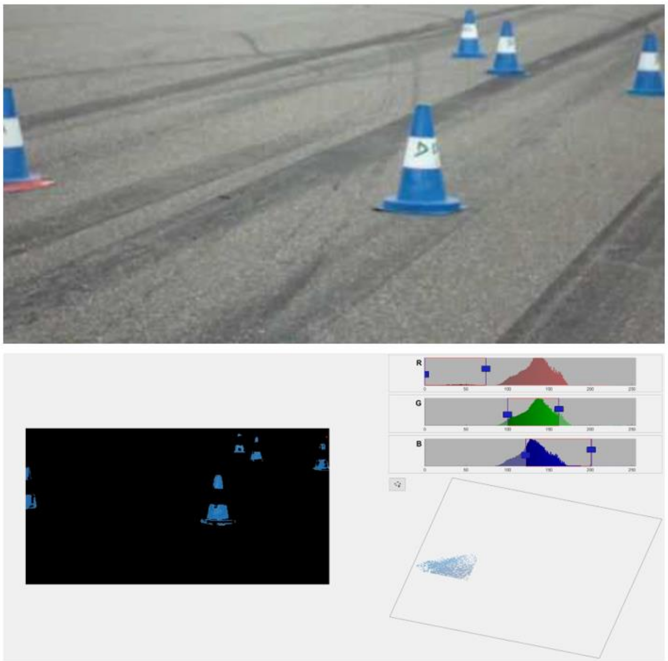


Figure 6-5: MATLAB color thresholder (bottom) applied to an image (top)

6.2.1.2. Descriptors

The second classical object detection method is the use of descriptors. This is handled by firstly, feeding the algorithm an image of the object to be detected, from which the feature descriptors will be extracted. If multiple images of the same object in different conditions are used, accuracy can be improved as more descriptors will be available for comparison. Secondly, these features are matched with the camera image for the objects to be detected.

Accuracy and efficiency will be influenced by the type of descriptor that is being used. Multiple comparisons have been done with the goal of identifying the most appropriate one for each application. The article published by Ebrahim Karami et. al [16] compares them based on rotation, intensity, scaling and distortion. They conclude that for their input images, ORB descriptors lead to the fastest algorithm, while SIFT descriptors are the most accurate.

With the purpose of observing descriptor extraction and matching, a basic program in Python has been coded using the OpenCV library [27] . ORB features are extracted, and matched using Brute-Force algorithm [28], which will take each descriptor of the image, compare it with all the descriptors from the camera image, and depending on its similarity, a distance value is given. The smallest one is selected and matched with the original descriptor.

The code is presented in *APPENDIX I: ORB feature extraction*. Image is read with the OpenCV function *cv2.imread()*, and then the ORB descriptors are extracted using the *orb.detectAndCompute()* function. A limited number of a 100 is set for a clear visualization. Afterwards, the key points are drawn in the image, shown in *Figure 6-6*.

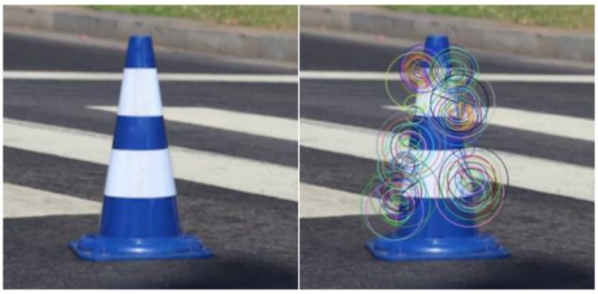


Figure 6-6: ORB descriptors in a blue cone picture [25]

In order to see feature matching between the sample cone and track cones with certain variation, the Brute-Force OpenCV algorithm was added to the code for 500 descriptors. Shown *APPENDIX II: BruteForce matching*. The results are represented in *Figure 6-7*.

[25] *https://www.shutterstock.com/es/image-photo/blue-plastic-cone-on-crosswalk-asphalt-1387480928*

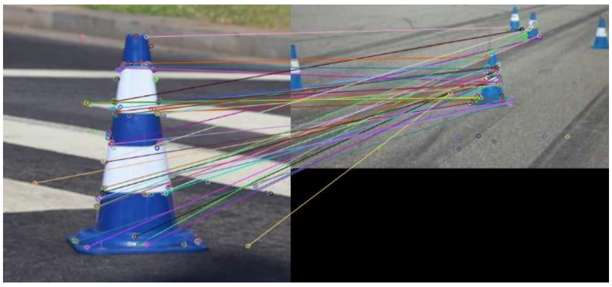


Figure 6-7: BruteForce algorithm applied to two cone images

Even though the cones had different white stripes, and only one cone picture was used for training the algorithm, it can be observed that the matches are qualitatively good. Moreover, no specific instruction was given to the algorithm to match the original image cone with each of the road cones, which caused crossed matches in the image. This points out that it could be a good starting point for cone detection in the Formula Student vehicle.

A more precise approach for multiple object identification is proposed by Stefan Zickler and Alexei Efros at Carnegie Mellon University [29], where multiple objects in the same image are detected using a variation of SIFT descriptors. This firstly obtains the descriptors in the camera image and groups them into clusters. Then the centroid of each cluster is computed and classified as a detected object.

6.2.2. Modern Object Detection: Convolutional Neural Networks

The development of computer vision techniques has led to improvements in the classical object detection methods over time. One of the most well-known approaches nowadays for image analysis and object classification are Convolutional Neural Networks (CNNs). In this section, an introduction to Neural Networks and CNNs will be given, different networks for object classification will be overviewed and one will be selected for a sample implementation.

6.2.2.1. Neural Networks and CNNs

In line with Professor's Mehlign, Artificial Neural Network professor at Gothenburg University [30], neural networks are computing models based on the biological net that makes up the brain. It is a collection of artificial neurons, called perceptrons, connected to each other through weighted links, through which information is sent and received.

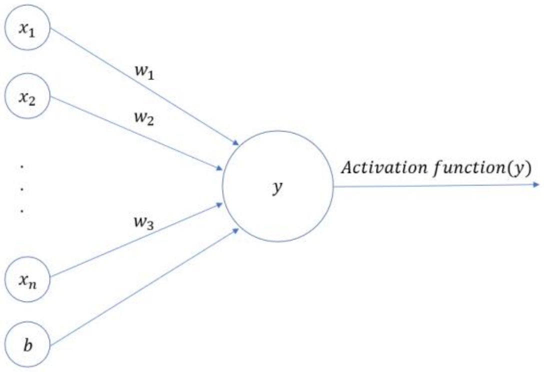


Figure 6-8: Neural Network Perceptron

Figure 6-8 shows a perceptron y with various inputs $x_1, x_2, x_3 \dots, x_n$ and the corresponding weights $w_1, w_2, w_3 \dots, w_n$. The weights define the importance of each input, and an activation function will determine the output of the perceptron. A linear activation function will be the weighted sum of the inputs, plus a bias, used to shift and adjust the function curve:

$$y = \sum_{i=1}^{n} (x_i \cdot w_i \, x_1 + b) \tag{16}$$

However, linear functions have great limitations in neural networks. When it comes to training (backpropagation), the derivative will be a constant with no relationship to the inputs, so the network cannot go back and adjust the weights to improve the prediction. As a result, multiple layers with linear activation functions can be combined into a single one, as the output will be a linear function of the input.

This issue can be solved by using non-linear activation functions, which can be different depending on the application. The most popular ones are presented in *Figure 6-9*.

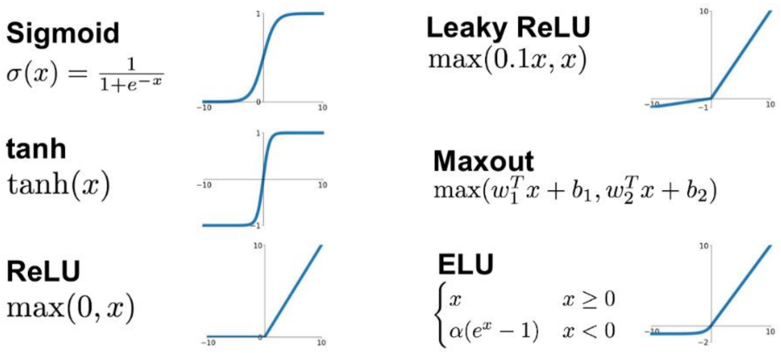


Figure 6-9: Popular neural network activation functions [26]

Convolutional Neural Networks are one variation of this Artificial Neural Network model that were introduced by Fukushima in late 1980s. He presented a model called "Neocognitron", with multilayer structure that could detect text patterns without the input position being a source of error [31]. Yann LeCun, and eminent name in the field, followed Fukushima and discovered backpropagation as an effective way of training a CNN to identify handwritten zip codes [32].

[26] *https://towardsdatascience.com/complete-guide-of-activation-functions-34076e95d044*

6.2.2.2. The training stage

In order to detect the correct class of the input object, training of the neural network is necessarily done. It consists of two steps. 1) Forward propagation, in which the image goes through the NN just like an input image would do, but initially the weights are random, and 2) backpropagation, explained at the deep learning book from Goodfellow et al. [33] where the information of the cost flows backwards to compute the gradient and adjust weights. Training is finished when the maximum number of iterations has been done, or when the cost function has been minimized to a certain threshold.

This loss function $J(\theta)$ can be defined as:

$$J(\theta) = \frac{1}{N}\sum_{i}^{N} loss(f(x^{(i)}, \theta), y^{(i)}) \tag{17}$$

Where θ is the parameter to be adjusted (for the weights: W_1, W_2 ...). $f(x^{(i)}, \theta)$ is the predicted output, $y^{(i)}$ is the actual output and N is the number of iterations, also called epochs.

Common loss functions are the cross-entropy function as well as mean-square error, both mentioned by Peter Sadowski in 'Notes about Backpropagation' [34].

Based on the learning stage, Neural Networks can be divided into three different approaches:

1) Reinforcement learning: The least related to the application of autonomous driving as it is based on a rewards/penalty policy. The network accuracy is improved every time a new success has occurred. i.e. it learns by making mistakes.
2) Supervised learning: Based on generating a dataset with the object already detected and labelled, input it to the NN to train it. The main downside of supervised learning is that the network will not get better performance than the images it has been trained for.
3) Transfer learning: It uses pre-trained models in which the first layers are very generic, and the last layers, the most specific ones to the application, are removed and retrained. This significantly reduces training time and resources. It is explained by Lisa Torrey and Jude Shavlik from the University of Wisconsin in their paper 'Transfer Learning' [35].

Training is done over a dataset of images. These should contain the object that is being trained for in a wide range of conditions different in lighting, perspective, distance, image quality, background… There are many generic datasets such as PASCAL VOC [36], ImageNet [37], COCO [38]. These are commonly used for CNN performance testing in object classification.

The most interesting set for the Formula Student Autonomous Racecar is FSOCO [39]. This dataset has been built by 45 FS teams for the purpose of training CNNs for cones from the AV competition. It contains around 360,000 images, and to increase this number, they are requesting at least 600 images to have access to the dataset.

When training, data is sometimes divided into three different groups mentioned in the citations by Jason Brownlee in his article 'What is the difference between Test and Validation datasets?' [40]:

1) Training set: The CNN will use images from the training set to adjust the weights.
2) Validation set: Instead of training the weights, it verifies with other pictures that the training is causing an improvement of accuracy in the neural network.
3) Testing set: The last set of images would be the one used by the user to visualize the results.

A key point to consider in CNN training is overfitting, shown in *Figure 6-10*. It means that the more iterations performed during training, the less accurate it becomes after going over a certain limit. The validation results will start to underperform when overfitting starts to show up.

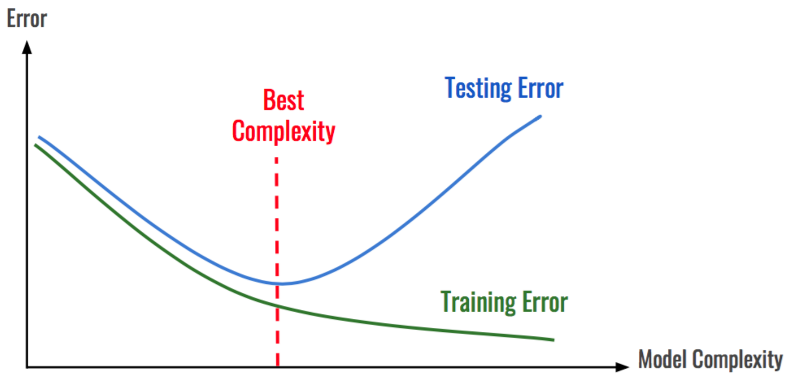


Figure 6-10: Overfitting in terms of model complexity referring to the number of iterations performed [27]

A simple technique to avoid this was presented in the 'Journal of Machine Learning' in 2014 by N. Srivastava called 'Dropout' [41]. A certain probability of participation is given to each perceptron, in the paper a Bernoulli probability distribution is used, and dropout rates between 0.5 and 1 are recommended. This will cause some random perceptrons to be inactive during some epochs, and active during others, resulting in a thinned network that is less prone to overfitting and takes less time to train. *Figure 6-11* shows the comparison between the neural net before and after dropout.

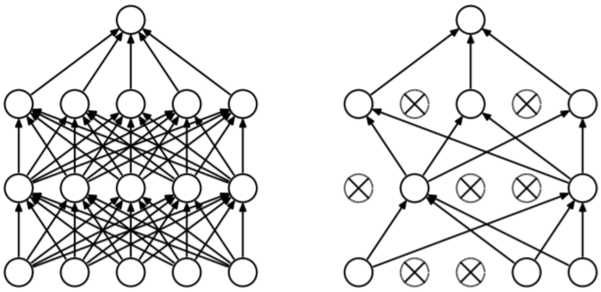


Figure 6-11: Node dropout in a neural network [41]

[27] *https://hackernoon.com/memorizing-is-not-learning-6-tricks-to-prevent-overfitting-in-machine-learning-820b091dc42*

6.2.2.3. Evaluation metrics

The evaluation of a dataset is done based on the validation dataset. It is useful to understand how the precision of a CNN is measured to select one that meets its purpose. There are two main ways of measuring precision depending on the type of neural network. They are mentioned in a Medium article by Jonathan Hui [42].

The first one is mean average precision (AP) for classification CNNs. Networks trained on PASCAL VOC use this metric. The formula in this case is simple, averaging the precision obtained for each of the classes it has been trained for.

$$AP = \frac{1}{N}\sum_{i=1}^{N} AP_i \qquad (18)$$

The average precision for each class will be:

$$AP_i = \frac{True\ positives\ of\ class\ i}{True\ positives\ of\ class\ i + False\ positives\ of\ class\ i} \qquad (19)$$

The second one is mean average precision (AP) for object detection and localization CNNs. This will mean that the network predicts bounding boxes for the objects it is detecting. COCO datasets are measured using this method. There are a few steps to take to get the AP value.

Firstly, the predicted box is compared to a ground truth bounding box from the validation set using Intersection over Union (IoU) visualized in *Figure 6-12*. The formula for this would be:

$$IoU = \frac{Predicted\ Bounding\ Box\ \cap Ground\ truth\ Bounding\ Box}{Predicted\ Bounding\ Box\ \cup Ground\ truth\ Bounding\ Box} \qquad (20)$$

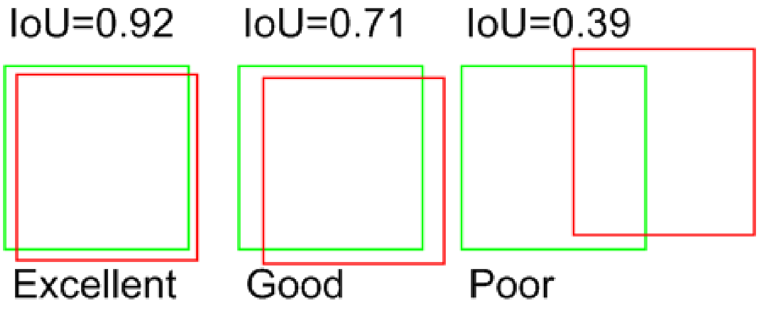


Figure 6-12: Intersection over union examples [28]

Secondly, the IoU is compared to a threshold. This border will be increased by five in each evaluation step, measuring from AP_{50} to AP_{95}. After all those evaluations have been done, the average of them is taken to measure the final value of the AP. Note that mean average precision can also be written as mAP.

$$mAP = \frac{1}{10}(mAP_{50} + mAP_{55} + mAP_{60} + \cdots + mAP_{95}) \qquad (21)$$

[28] *http://www.interstellarengine.com/ai/Intersection-Over-Union.html*

Moreover, as it might be seen in some research papers, object detection and localization CNNs are measured for small, medium and larder objects using this same method. The results will be represented as AP_S, AP_M, and AP_L respectively.

As a result of an IoU evaluation metric, the results will usually be lower than for the conventional method, as values up to 95% precision are taken into consideration.

6.2.2.4. Convolutional Neural Network Architecture

CNNs are built of numerous layers, and even though there are some that might be specific to each model, convolution, activation, pooling, and fully connected layers are found in in most of the networks.

6.2.2.4.1. Convolution layer

Convolution layers in simple words, apply filters to the original image to break it into smaller pieces. They are composed by several feature maps that perform the same operation of applying a specific filter (Kernel) over the image coming from the previous layer. Each feature map will look for a single thing in different locations, for example in lower levels it could search for a curved line, while in higher levels it can be looking for a cat or a face. Therefore, multiple feature maps will lead to multiple feature identification.

A mathematical representation is given by LeCun in his article 'Convolutional Networks and Applications in Vision' [43]. The output $Y^{(l)}$ of layer l will have $N^{(l)}$ feature maps. The i^{th} feature map denoted by $Y_i^{(l)}$ is given by:

$$Y_i^{(l)} = \sum_{i=1}^{N^{(l-1)}} K_{i,j}^{(l)} * Y_i^{(l-1)} + B_i^l \qquad (22)$$

$(*)$ is the convolution operator, $K_{i,j}^{(l)}$ is the Kernel with i being the index of the previous feature map (or layer as in the case of *Figure 6-13*, where there is no bias) and j the index of the current feature map.

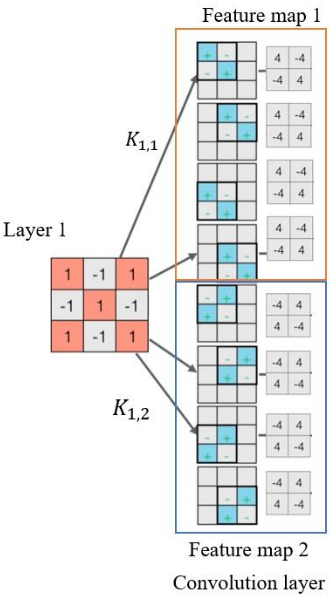


Figure 6-13: Convolution layer feature map examples [29]

[29] *'A friendly introduction to Convolutional Neural Networks and Image Recognition', YouTube, Luis Serrano, https://www.youtube.com/watch?v=2-Ol7ZB0MmU*

6.2.2.4.2. Activation layer

The activation layer, also called nonlinear layer, performs an elementwise operation of the activation function in the input image. It will determine which of the neurons of the next layer will fire. Instead of setting up the output of each perceptron to apply the nonlinear function, a layer is created that performs this step. It is mathematically represented as:

$$Y^{(l)} = f(Y^{(l-1)}) \tag{23}$$

The dimensions of the output and the input are identical.

6.2.2.4.3. Pooling layer

After the features have been extracted using convolution layers, they must be used for classification. However, doing this directly would lead to very large processing power, as well as overfitting. It is illustrated in an example from Stanford Deep Learning tutorial website [44]. An image of size 96x96 pixels with 400 features over 8x8 inputs, would lead to an output size for each convolution of (96−8+1)·(96−8+1)=7921, this can be computed with the formula mentioned in the CNN Stanford course [45]:

$$Output\ size = \frac{W - K + 2P}{S} + 1 \tag{24}$$

Where W is the image size, 96, K is the filter size, 8, S is the stride, or number of pixels the filter moves at each step, 1, and P is the padding, external image adjustments for the filter to fit correctly, 0 in this case.

The resulting vector is of 7921·400=3,168,400 features per sample image.

Subsequently, the pooling layer, also called subsampling layer, will perform the downsampling of the inputs. Moreover, in case pooling is done horizontally, translational invariance will be added to the features. Mathematically, the output volume size will be $m_1^{(l)} x\, m_2^{(l)} x\, m_3^{(l)}$:

$$\begin{aligned} m_1^{(l)} &= m_1^{(l-1)} \\ m_2^{(l)} &= \frac{(m_2^{(l-1)} - F^{(l)})}{S^{(l)}} + 1 \\ m_3^{(l)} &= \frac{(m_3^{(l-1)} - F^{(l)})}{S^{(l)}} + 1 \end{aligned} \tag{25}$$

F is the pooling filter size and S is the stride.

Two pooling types are used for this size reduction, average pooling will average the values of the convolution layer inside the filter, while max pooling will take the greatest value.

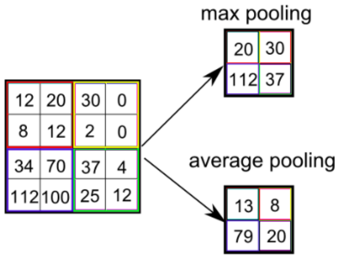


Figure 6-14: Max pooling and average pooling example [30]

6.2.2.4.4. Fully connected layer

The goal of the fully connected layers in a CNN is to process and classify the results from the previous layers. They are formed as a multilayer perceptron with two or three layers. The output will be a vector whose elements are the probabilities of the input image belonging to each of the labels the networks has been trained for.

6.2.2.5. Existing Convolutional Neural Network

After studying the working principle of convolutional neural networks, a table of available categories for Neural Networks is presented with their main features and superiorities (*Table 1*).

A basic categorization is done based on the stages, where a single stage points to CNNs that are faster and easier to implement, as they directly establish the bounding box for the desired object, while double step consists of the region of interest (ROI) extraction first and then building the bounding box around the object. Another differentiation can be done between camera-based networks and LIDAR based, however, LIDAR data is not often very significant to detect the class of certain objects as mentioned in the comparison 'A Survey of Deep Learning Techniques for Autonomous Driving' by researchers from Transylvania University of Brasov [46]. This paper also mentions the existence of networks working with the combination of both and how it might take a lot of processing power.

[30] *https://www.quora.com/What-is-max-pooling-in-convolutional-neural-networks*

Table 1: Convolutional Neural Network examples

Classification		**CNN Name**	**Approach**	**Main features**	**Reference**
Camera based	**Two-Stage**	SPPNet	Last pooling layer replaced with spatial pyramid pooling layer to keep feature spatial information.	Variable input image size	[47]
		R-CNN	Three step operation and training: region proposal, feature extraction, and bounding boxes.	Popular network with good precision.	[48]
		YOLO	Image is divided into a grid, confidence score and probability are assigned.	Lots of variations: Fast YOLO (Up to 155 fps), YOLOv2, YOLOv3. Real Time, original YOLO up to 45 fps.	[49]
		Single-Shot Multibox detector (SSD)	Single feedforward pass of the input image is done. That produces a fixed-size collection of bounding boxes with different scales and confidence scores. Another final step will produce the final detections	Real time object detection and classification up to 46 fps.	[50]
	One-Stage	OverFeat	Sliding Window approach, one of the first single stage methods.	Classification, Localization & Detection	[51]
		CornerNet	Detect objects as paired key points. Two maps are generated, one for top corners and another one for bottom corners. Best matches are grouped to form the box for an object.	High performance. No need to predesign bounding boxes to fit into the objects.	[52]
		SqueezeNet	CNN with convolution layers that have in majority 1x1 filters instead of 3x3, it decreases the number of input channels to 3x3 filters and carries out downsampling.	Very low memory requirements. Model can be compressed down to 0.5MB. Useful for FPGAs	[53]

LIDAR	PointNet	Learns a set of optimization functions/criteria that select interesting or informative points of the point cloud and encodes the reason for their selection. The final fully connected layers of the network aggregate these learnt optimal values into the global descriptor for the entire figure.	Point cloud pattern extraction for object identification	[54]
LIDAR and Camera	RoarNet	Uses the monocular camera image to narrow down the search space, and performs detection within these regions	Creates 3D bounding boxes. It is publicly available and pretrained.	[55]

Table 1 offers some generic knowledge about some popular Convolutional Neural Networks. However, a good way of understanding the different steps to setup a neural network from start to finish is by putting it into practice.

The following section will address setting up the training environment and training the convolutional neural network YOLOv3 based on the official website support [49], and additional sources like articles from Medium [56].

6.2.2.6. Setting up the runtime environment

The runtime environment is a crucial component when training a neural network, as it will define the computational resources that are going to be used. The better the graphics processing unit (GPU), the less time the training process will take, but the more expensive it will be per hour of use. Two main alternatives will be examined, Amazon Web Services (AWS) and Google Collaboratory.

On the one hand, A popular cloud runtime environment used for Deep Learning is AWS [57], which offers a $100 initial credit for student accounts. The downside is that the account can't be upgraded to "Developer Account" unless a payment is done, and therefore the only instance available is "t2.micro" which corresponds to only 1GB of simple GPU memory.

An alternative to this has been Google Collaboratory notebooks, which offers for free the following specifications:

- Tesla K80 24GB GDDR5 GPU
- x2 Intel(R) Xeon(R) CPU @ 2.30GHz
- 13GB of RAM
- 34 GB disk space

The interface is very similar to a python Jupyter notebook, but it runs on the cloud and through Google Drive.

As no in-depth coding was going to be done during YOLOv3 implementation, and since it is completely free, Google Collab was the selected platform to train and test the CNN. On the flip side, there are were some downsides that it was necessary to overcome:

- Every time the notebook is closed, all the data runtime will be lost: Data should be downloaded to the computer right after generating it.
- Maximum runtime before the notebook gets disconnected is around 20 to 30 minutes: Users from Medium [56] have figured out a way to keep and "auto-clicker" and maintaining the page active.
- Unintuitive folder organization: It took a long time to familiarize to how the folders and organization units worked together with Google Drive.
- Google sets the user's Collab's limits: Google reserves the right to assign processing time to users of their choice.

If a balance was to be done between the pros and the cons of using Google Collaboratory as a totally free Deep Learning training platform, it would be that it is best option for very specific tasks, such as training and testing, but not for full development from scratch.

6.2.2.7. Selecting the Convolutional Neural Network

Table 1 presented many of the available options in the field of CNNs. As it might have caught the reader's attention, YOLO can run up to 155 frames per second, which makes it an attractive choice for autonomous racing. Another benefit of building perception upon YOLO is the available cone dataset, FSOCO [39], developed by FS teams such as AMZ and Munich MotorSport. Here, the data is labelled in a way that can be used to train YOLO CNNs and its variations with similar input requirements. Furthermore, the open source history of this network has increased significantly since its release.

6.2.2.8. You Only Look Once (YOLO)

YOLO was published by Joseph Redmon in 2016 [58], and it is built over his Darknet framework. Its main goal is to pursue object detection in a straightforward way running a single convolutional neural network.

The main difference of this CNN with respect to other object detectors is its approach. Instead of using a sliding window going through the image, it considers the image globally, dividing it into an SxS grid. After this, each cell of the grid predicts the x, y, w, h and confidence score C for each class, of a number B of bounding boxes. x, y are coordinates for the center of the bounding box, and w, h are its width and height. The best bounding box for each object is selected and placed from a set of predefined anchor boxes with various sizes. The confidence score is based on the Intersection over Union (IoU) between the predicted box and the ground truth box.

The network architecture has 24 convolutional layers with two fully connected layers that will predict the output probabilities and coordinates. The input image must be 448x448x3 pixels and the output will be of size $S \times S \times (B * 5 + C)$. For example, if the picture is divided in 7 cells, the bounding box limit is 2, and there are 20 classes, prediction output will be 7$\times$ $7 \times (2 * 5 + 20)$. The network architecture is shown in Figure 6-15: YOLO CNN architecture *Figure 6-15*.

The last important thing to consider about YOLO is its loss function. It can be divided into two parts, one for object classification and the other one for localization. An article by Lilian Weng [59] represents it in a more intuitive way than the YOLO publication paper.

$$\mathcal{L}_{loc} = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 + \left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2 \right] \qquad (26)$$

$$\mathcal{L}_{class} = \sum_{i=0}^{S^2} \sum_{j=0}^{B} \left(1_{ij}^{obj} + \lambda_{noobj}\left(1 - 1_{ij}^{obj}\right)\right)\left(C_i - \hat{C}_i\right)^2 + \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_i^{obj}\left(p_i(c) - p_i(c)\right)^2 \qquad (27)$$

$$\mathcal{L} = \mathcal{L}_{class} + \mathcal{L}_{loc} \qquad (28)$$

$\mathcal{L}$: Total loss function

$\mathcal{L}_{loc}$: Object localization loss function

$\mathcal{L}_{class}$: Object classification loss function

1_i^{obj}: An indicator of whether cell i contains an object. It will be 1 if there is an object in that cell, indicating that it is accounting for object classification.

1_{ij}^{obj}: Indicates if the j^{th} bounding box is responsible for that prediction. Therefore, if it is responsible, it will be one, and loss will be considered. If it was not responsible for the prediction, there would be no point considering it.

C_i: Confidence score of cell i,from the multiplication of the probability of containing an object and the intersection over union results.

$\hat{C}_i$: Predicted confidence score

c: Class contained in the set of classes $\mathcal{C}$.

$p_i(c)$: Conditional probability of cell i containing an object of class c:

λ_{noobj}: Importance given to object classification loss. A value of 0.5 is used in the paper.

λ_{coord}: Importance given to object localization loss. A value of 5 is used in the paper.

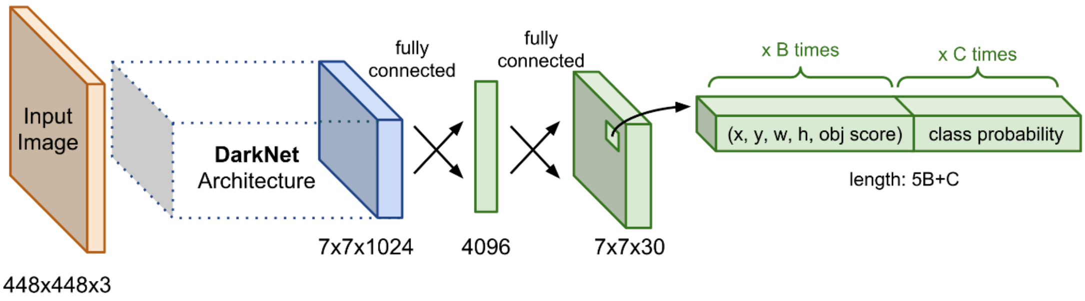


Figure 6-15: YOLO CNN architecture [31]

Between 2016 and 2018 other variations of YOLO have been released, such as YOLO-Lite, less computer demanding, [60]. Fast YOLO, for embedded computers, [61] and YOLO9000, which can detect up to 9000 classes [62]. An impressive alternative is YOLOv3-Tiny, which can get up to

[31] *https://lilianweng.github.io/lil-log/2018/12/27/object-detection-part-4.html#yolo-you-only-look-once*

33.1mAP running at 220 frames per second. It is recommended that the reader explores, tests, and verifies these alternatives more in detail to select the most attractive one depending on his hardware requirements.

One of the most recent versions has been explored and implemented for demonstration purposes, YOLOv3 [63].

According to the original paper, this variant is a more accurate approach than the original YOLO, however it is slightly slower than YOLOv2, presented together with YOLO9000 [62]. However, Joseph Redmon in his website [49] suggests that YOLOv2 has a 48.1mAP at 25ms (or 40 frames per second, 1fps = 1 Hz) , while the latter can perform up to 51.4mAP at 22ms (45 fps).

The main innovations to the v3 CNN are pointed out by Ayoosh Kathuria from Medium [64]:

1) It is now built over Darknet-53 feature extractor and adds another 53 convolutional layers, making a total of 106 convolutional layers, as opposite to the version 2 built over Darknet-19.
2) It takes decisions at three different scales. Applies 1x1 detection kernels on feature maps of three different sizes at three different places in the network, which subsequently adds three upsampling layers.
3) Makes predictions at three different scales, by downsampling the input image by 32, 16 and by 8. The predictions are made at layers 79, 91 and 106, as it can be observed in the orange layers from *Figure 6-16*.
4) Improved at detecting smaller objects but made worse at larger object predictions.
5) It now uses 9 anchor box models for bounding box prediction, and it predicts more bounding boxes per image.
6) $\mathcal{L}_{class}$ squared error has been replaced by cross-entropy loss function, which means predictions are made using logistic regression.

$$Cross\ entropy\ loss = -\sum_{c=1}^{N} y_{o,c} \log{(p_{o,c})} \tag{29}$$

N is the number of classes, $y_{o,c}$ binary indicator if the class c is the correct classification for observation o, and $p_{o,c}$ is the predicted probability that observation o is of class c.

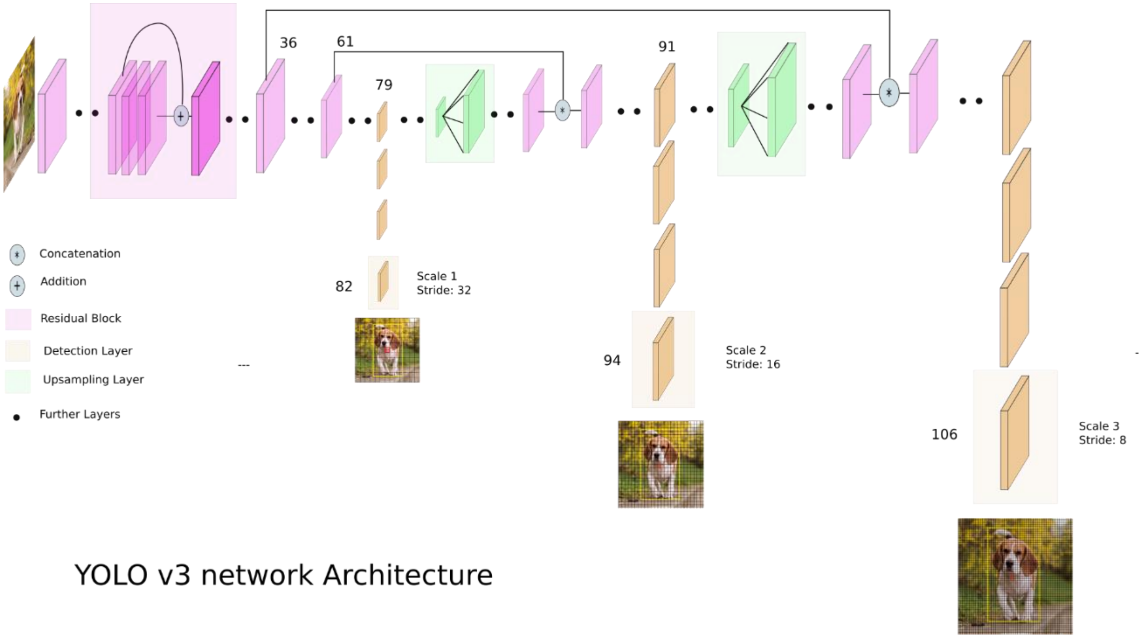


Figure 6-16: YOLOv3 CNN architecture [64]

6.2.2.9. Implementation of YOLOv3 in Google Collaboratory

Before following the guides provided by open-source programmers online, it is recommended that a simple introduction to CNN programming in Python with Keras is done. This gives the user a general overview of the neural network architecture and implementation steps. One of the most popular ones is the MNIST, that identifies handwritten digits and letters. A simple and explanatory tutorial is presented by user Manish Bhobé in Medium [65].

However, none of the Keras layer-by-layer programming will be used for YOLOv3, which comes already built. There are two main stages in order to use YOLO for object detection, getting the dataset for the class, and training the network with that data. Guidance from the original YOLO website [49] was followed, as well as the recommendations from GitHub [66] by a YOLO fork developer Alexey Bochkovskiy.

6.2.2.9.1. Creating a custom dataset

With the goal of proving that YOLOv3 can be trained for a custom class, and without having the FSOCO cone image set data available, balloon pictures were used instead. They were taken from "Google Open Images Dataset V6". Two thousand images containing balloons were downloaded for training, and 44 for validation. Their labels were adapted to YOLO format using code from the OIDv4 ToolKit [67]. The labels from the downloaded pictures had the format:

$\boldsymbol{Balloon}$	$\mathbf{553.6\ px}$	$\mathbf{484.565\ px}$	$\mathbf{753.28\ px}$	$\mathbf{680.4394\ px}$
$\boldsymbol{Name\ of\ the\ class}$	$\boldsymbol{x_{min}}$	$\boldsymbol{y_{min}}$	$\boldsymbol{x_{max}}$	$\boldsymbol{y_{max}}$

However, YOLO only accepts labels with the normalized coordinates, and a class identifier for each class it is being trained for. After using the toolkit, the result was the following:

$\mathbf{0}$	$\mathbf{0.4990}$	$\mathbf{0.3511}$	$\mathbf{0.9993}$	$\mathbf{0.2908}$
$\boldsymbol{Class\ ID}$	$\boldsymbol{\frac{x_{min}(px)}{width\ (px)}}$	$\boldsymbol{\frac{y_{min}(px)}{height\ (px)}}$	$\boldsymbol{\frac{x_{max}(px)}{width\ (px)}}$	$\boldsymbol{\frac{y_{max}(px)}{height\ (px)}}$

The files were then compressed into two different archives, one for training and one for testing.

6.2.2.9.2. Coding the train script

Steps followed for implementing YOLOv3 are shown in *Figure 6-17*. The process can be divided into ten stages. While its source code is shown in *APPENDIX III: YOLOv3 implementation*, some configuration parameters are explained for a better understanding of the network.

1) Clone repository: Darknet repository must be cloned from GitHub's original repo.
2) Define *imshow*(): Helper functions should be defined to plot the final results.
3) Upload the training set: The custom training set that has been created previously needs to be uploaded to the Google Collaboratory workspace so that it can be used and unzipped.
4) Python text file script: YOLOv3 requires as well "train.txt" and "test.txt" files that contains all the paths of the training and validation images. The script is provided by "The AI Guy" in GitHub [68] and it should be edited to match the Google Drive paths.
5) Edit "obj.data" and "obj.names": The file "obj.data" will contain the number of classes, and the path of the ".txt." files generated in step 4) so that the images can be used by the CNN for training and validation.
6) Change config file: This is one of the most important steps when setting up YOLOv3, as it is defining how the network will be trained. Each of the main parameters is explained in detail in *Table 2*. It is interesting to read through them to understand how training of YOLOv3 works.
7) Train: Training of the network is done in this step by running a command specific to YOLOv3. The path of the configuration file, "obj.data" and "obj.names" is needed for the training to be successful. With the goal of considerably reducing training time, transfer learning was used by starting from some given pre-trained weights available at the Darknet GitHub repository [66]. It took around six hours to train on 2000 images.
8) Upload validation data set: Like step 3), the validation set obtained from following the steps to create a custom dataset needs to be uploaded to the workspace to be unzipped.
9) Validate: The precision of the CNN was measured with the validation data set, and the medium average precision (mAP) results were obtained.
10) Test: Some practical tests were run using some random images to demonstrate the detections performed.

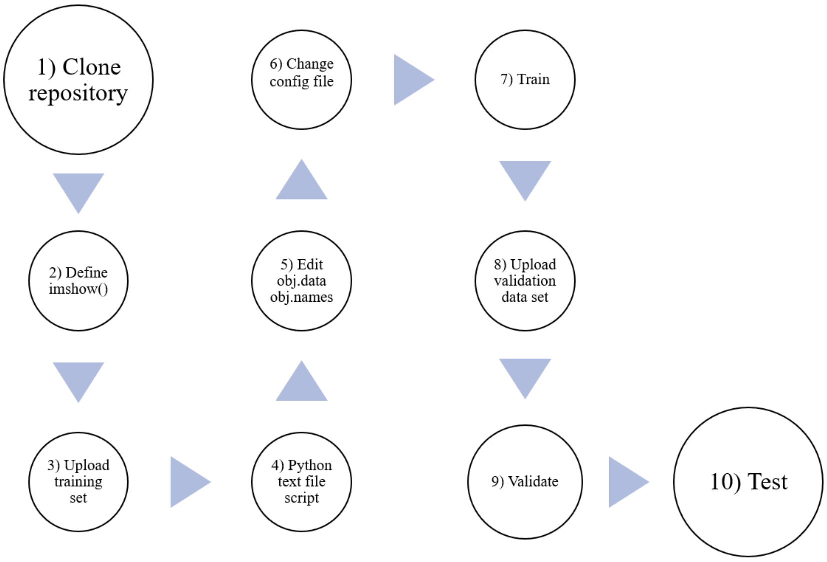


Figure 6-17: YOLOv3 implementation process

Table 2: YOLOv3 hyperparameters

Parameters	**Explanation**

# Training 	
batch=64
subdivisions=16
width=416
height=416
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1
	The training set will be divided into random batches of 64 images of height 416x416x3 pixels. Having big batch sizes will make the CNN more generic as the weights will not be changed significantly for every single picture.
	The momentum measures how large changes in the weights during training are penalized, and the decay prevents overfitting by penalizing when the weight values are too large, this the CNN is overfitting.
	The last three parameters, saturation, exposure, and hue just transform the colors of the image so that object detection is less influenced by lighting, contrast, and color.

learning_rate=0.001	The learning rate defines how fast the CNN learns new features. A large value will cause steps to be larger and therefore be less precise, while smaller values will make the network more precise, but will take longer to train.
burn_in=1000	The burn in (or warm up) hyperparameter refers to the iterations it takes to the CNN to reach the specified learning rate.
max_batches = 2000	Maximum number of iterations the CNN will be trained at. It should be the number of classes times 2000.

policy=steps
steps=1600,1800
scales=.1,.1
	These lines will make YOLO change the learning rate during training. When it reaches 1600 it will be multiplied by a scale factor of 0.1, and when it reaches 1800 it will then again be multiplied times 0.1.

To better illustrate the hyperparameters: ‘learning_rate’, ‘burn_in’ and ‘steps’, the log during the training of YOLOv3 was outputted for the first 1700 iterations. The most meaningful lines have been selected.

```
1: 1882.593140, 1882.593140 avg loss, 0.000000 rate, 5.678190 seconds, 64 images, 1.000000 h
ours left
999: 1.170234, 1.755827 avg loss, 0.000996 rate, 2.862937 seconds,
63936 images, 0.984242 hours left
1000: 1.867372, 1.766982 avg loss, 0.001000 rate, 2.925025 seconds, 64000 images, 0.982360 h
ours left
1300: 2.272686, 1.589324 avg loss, 0.001000 rate, 4.064846 seconds, 83200 images, 0.810453 h
ours left
1600: 2.351540, 1.780402 avg loss, 0.000100 rate, 2.885864 seconds, 102400 images, 0.473643
hours left
```

It can be observed how the learning rate "rate" starts at 0 and increases to 0.001 in the first 1000 epochs. Then it is kept at 0.001 until iteration 1600, where it is multiplied by 0.1 and results in a learning rate of 0.0001.

6.2.2.9.3. Results

Training was done with 2000 images. Darknet outputs a chart (*Figure 6-18*) that illustrates how the loss function drops at each iteration. During the first 200 iterations the value is over 1700, thus it is not shown, however it can be observed how the loss drops significantly after 180 iterations. Then it stabilizes and continues decreasing until it reaches 1.4526. Furthermore, the precision measured was 33.67mAP for a detection time of 90 milliseconds.

Table 3 shows how the test was performed in two images using a confidence threshold of 0.1 for the left and 0.3 for the right column. The confidence scores for each balloon detection are shown under each image.

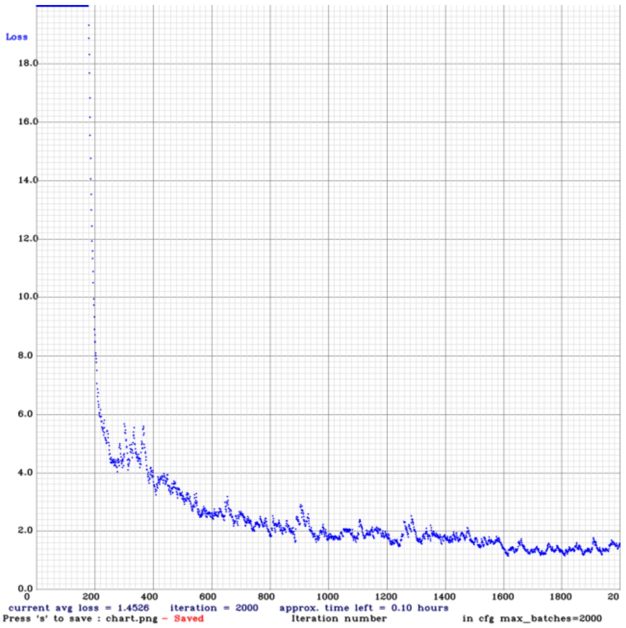
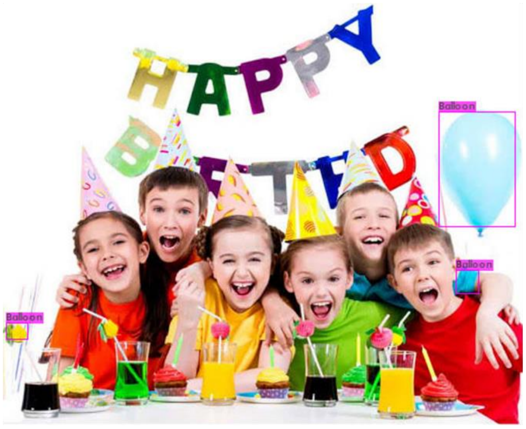
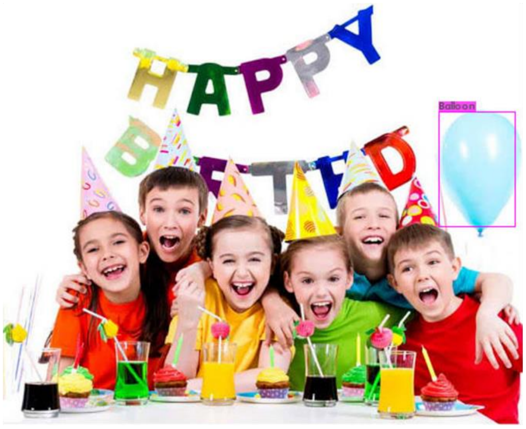
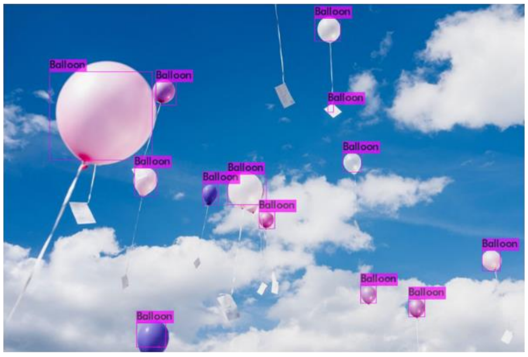
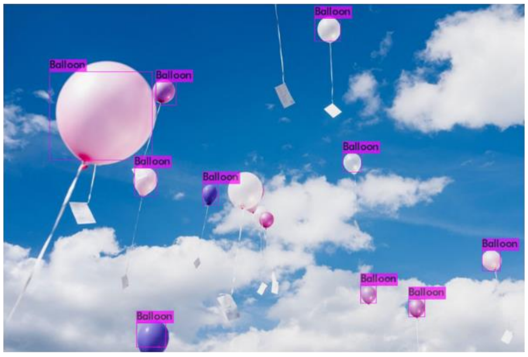


Figure 6-18: YOLOv3 epochs-avg loss chart

Table 3: YOLOv3 results

	
Balloon: 28%, Balloon: 74%, Balloon: 13%	Balloon: 74%
	
Balloon: 36%, Balloon: 58%, Balloon: 33%, Balloon: 52%, Balloon: 44%, Balloon: 14%, Balloon: 29%, Balloon: 58%, Balloon: 25%, Balloon: 50%, Balloon: 40%, Balloon: 54%, Balloon: 71%,	Balloon: 36%, Balloon: 58%, Balloon: 33%, Balloon: 52%, Balloon: 44%, Balloon: 29%, Balloon: 58%, Balloon: 25%, Balloon: 50%, Balloon: 40%, Balloon: 54%, Balloon: 71%,

6.2.2.9.4. Conclusion

As a result of studying YOLOv3, the main goal of understating the working principle of a CNN was achieved by analyzing its architecture and loss function. Moreover, implementing the network lead to a better understating of how to use it for custom object detection, which could valuable for cone detection in the Formula Student autonomous driving competition. The network was successfully configured, trained, and tested giving 33.67mAP for 90ms, that can be improved by using more training images, a more powerful graphics unit, or even experimenting with other YOLO variations and the FSOCO database.

6.2.2.9.5. Note in combining YOLO and LIDAR information

A question that might arise is how the bounding boxes containing the object can be placed in a 3-Dimensional map to plan the trajectory of the autonomous vehicle. Milan Aryal, graduate student from Grand Valley State University gives an easy-to-understand approach in his master's thesis 'Object Detection, Classification, and Tracking for Autonomous Vehicle' [69].The LIDAR point cloud is projected into the camera image, and only the points that are inside the YOLO bounding boxes are considered. The distance is measured and taken to a 3D reference space. The positions of the objects are then fed to the Extended Kalman Filter for processing and path planning. *Figure 6-19* illustrates this way of combining CNN bounding box and laser scan information.

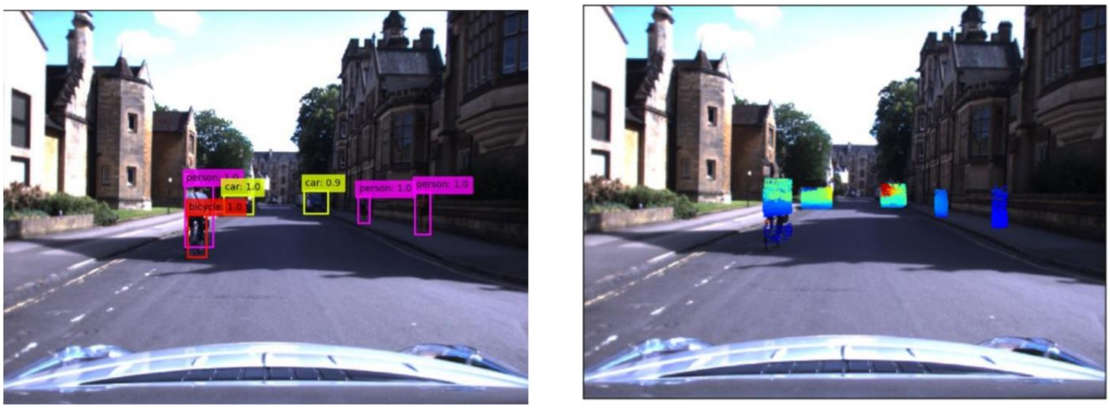


Figure 6-19: Combining LIDAR data and YOLOv3 results [69]

6.3. SLAM (Simultaneous Localization and Mapping)

Simultaneous Localization and Mapping is the computational problem of generating a map of the environment, while keeping track of the position of the vehicle withing that map. This issue is commonly presented in robotics, and it comes up when the vehicle needs to drive through an unknown environment. Moreover, it covers a very wide field of research and hence only a brief introduction will be provided.

Terms & Definitions:

Bayes' Theorem: The Bayes' Theorem is a statistical rule that gives the conditional probability of a random event A given B. The mathematical formula behind it will be:

$$P(A_i|B) = \frac{P(B|A_i)P(A_i)}{P(B)} \tag{30}$$

Where $P(B|A_i)$ is the probability of B given A and $P(A)$ and $P(B)$ are the likelihood of A and B.

The steps followed for SLAM are shown in (*Figure 6-20*). First, the sensors measure the state of the vehicle and detect the surroundings, then the different points in the map at each timestep need to be matched, after finishing the mapping of the area, the robot returns to the origin and closes the loop.

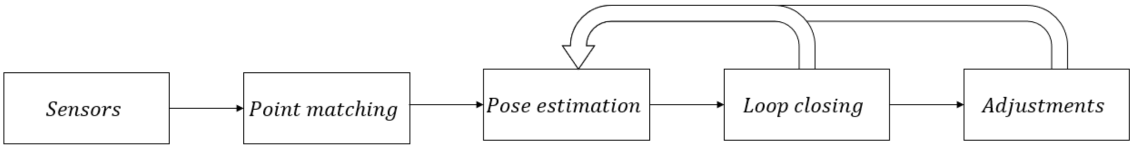


Figure 6-20: Simultaneous Localization and Mapping (SLAM) steps

The paper published in the 'International Journal of Engineering & Technology' by researchers from the University of Technology MARA' in Malaysia [70], summarizes the three main SLAM solving methods for autonomous vehicles classified into two categories, filter based, and optimization based. The article also has useful bibliography to dive deeper in the details of each type of SLAM.

Filter based methods include EKF SLAM, and Fast SLAM. They are based on the Bayes' Theorem, that applied to SLAM, has the goal of predicting the car position in the map, by previously knowing the sensor data and the input controls.

The formulation of the problem is represented in eq. (31), where x_k is the car pose estimation, M is the map of the environment, z_k is the sensor readings, and u_k are the vehicle's inputs.:

$$P(x_k, M) = P(x_k, M|z_{0:k}, u_{0:k}) \tag{31}$$

The optimization method described in the paper is Graph SLAM, that instead of predicting the states from the new inputs, the vehicle's current pose in the map is identified from the sensors.

7. Control

7.1. Car Model

Terms & Definitions:

State: Collection of variables that summarize the past of a system in order to predict its future.

State Vector: The collection of the state variables written as a vector $x \in \mathcal{R}^n$.

State Space Model: A model that describes a system using the differential equations.

$$\frac{dx}{dt} = f(t, x, u) \tag{32}$$

Cornering stiffness: Cornering stiffness is the ability of a tire to resist deformation in its shape when the vehicle corners. It is a function of the lateral force of the tire and its slip angle.

The first step in the control of an unmanned vehicle is defining the car model. It will be the one used by the controller to reduce state error. It might also serve the purpose of using it in simulation environments to test the driving algorithms. This model should be able to imitate the car's behavior.

Subsequently, the closest the model is to the real vehicle the better the results will be. However, high level of accuracy will require large computational power, so based on the requirements and resources available, different car models can be selected. The car-like model would suit the more precise approach, while the bicycle model usually gives mathematical simplicity still being feasible in most cases [71].

The models can also be developed in three stages, the geometric model, the kinematic model, and the dynamic model, depending on the car's speed and the required precision. This division is introduced in 'Modelling and Control Strategies in Path Tracking Control for Autonomous vehicles' by Noor Hafizah et al. [72]. The kinematic and dynamic model are summarized based on the book 'Vehicle Dynamics and Control' by Rajesh Rajamani [73].

7.1.1. Car-like model

Car-like mathematical models will represent the vehicle precisely, however, they become very complex when introducing the lateral dynamics of all four wheels. According to the book 'Robot Motion Planning and Control', by De Luca et al. [74], they are non-holonomic systems. This means that the number of controllable DoF is smaller than the number of the system's DoF. In this, case, the car has 4 degrees of freedom (m=4), motion in x, motion in y, orientation (phi) and heading or wheel steering angle (delta), and only two (n=2) are controllable, longitudinal direction and lateral direction.

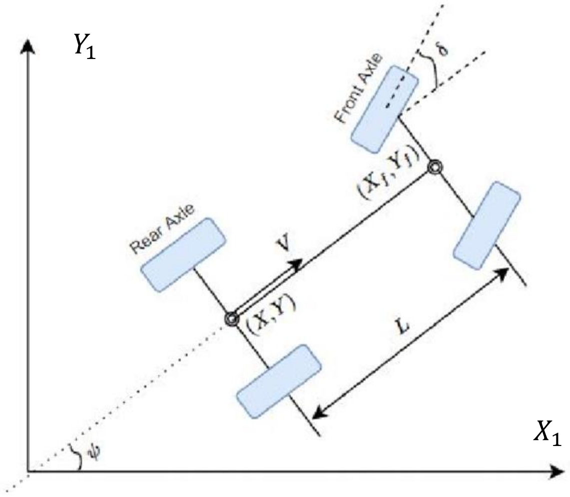


Figure 7-1: Car-like vehicle model [71]

- (X ,Y): Local reference frame. In car-like models the origin is in the rear axle.
- (X_1, Y_1): Global reference frame.
- (X, Y): Position in global reference frame.
- φ: Angular difference between global and local reference frame.
- δ: Front wheel steering angle.
- L: Length between front and rear axles.
- V: Velocity vector of the vehicle.

7.1.2. Bicycle model

The bicycle model is a simplification of the car-like model by assuming that the steering angles are going to be small. It can be implemented in real self-driving vehicles and its complexity is reduced under the assumptions stated at a paper published from the Toyota Technical Institute, 'A Tutorial On Autonomous Vehicle Steering Controller Design, Simulation and Implementation' [71]:

1. The left and right wheels in both axles are lumped in a single wheel.
2. Roll and pitch motions are ignored (no load transfer due to roll and pitch motions).
3. Vehicle runs on a smooth road with a constant speed.

The bicycle model is represented in *Figure 7-2*, showing the tire variables that will be necessary for the dynamic model.

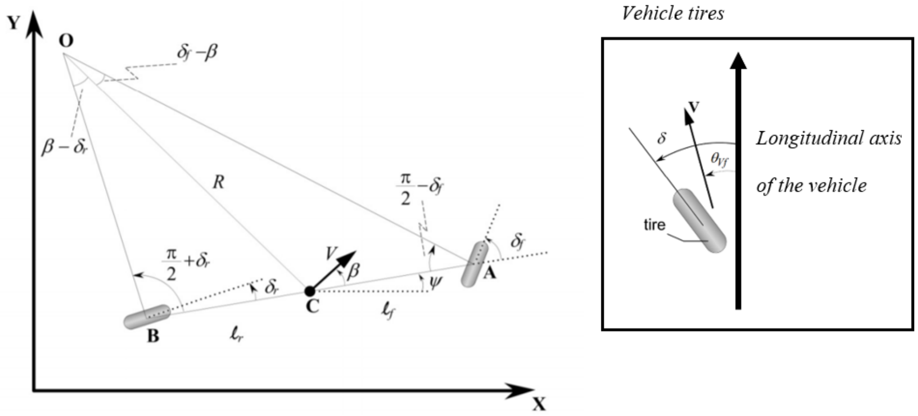


Figure 7-2: Bycicle model [73]

- O: Instantaneous rolling center of the vehicle. Intersection between the perpendicular lines to the wheels in A and B.
- C: Center of gravity of the vehicle.
- A, B: Front and rear axles.
- (X, Y): Global reference frame.
- ψ: Angular difference between global and local reference frame.
- δ_f: Front wheel steering angle.
- δ_r: Rear wheel steering angle. Zero if there is only front steering.
- l_f: Length between front axle and center of mass.
- l_r: Length between rear axle and center of mass.
- L: Length between front and rear axles ($L = l_f + l_r$).
- V: Velocity vector of the vehicle in the center of mass.
- θ_{Vf}: Angle that the velocity vector makes with the longitudinal axis of the vehicle and δ_f. The front slip angle will be $\alpha_f = \delta_f - \theta_{Vf}$.
- θ_{Vr}: Angle that the velocity vector makes with the longitudinal axis of the vehicle and δ_r. The rear slip angle will be $\alpha_r = \delta_r - \theta_{Vf}$.
- β: Direction of the velocity at the center of gravity with respect to the car's longitudinal axis.

7.1.3. Geometric vehicle model

This one is the simplest type of car model; it only considers the position and the geometric constraints of the vehicle. As a result, the precision of the geometric model is low. as It does not consider the speed and acceleration. Implementation of this vehicle model is purely geometric based on the Ackerman Steering model, which computes the steering angle according to the road's curvature radius. Here, the geometry of the road will determine the next approaching point. The Pure Pursuit controller, developed in the 80s at Carnegie Mellon University [75], is an example of application of the geometric model.

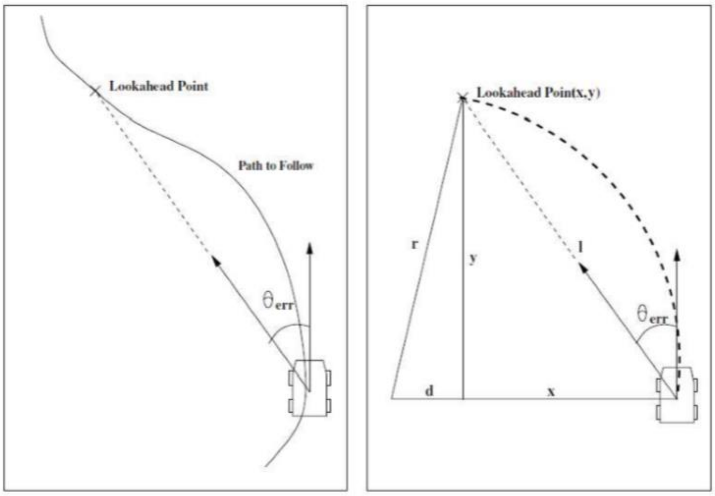


Figure 7-3: Geometric-based model path planning [32]

7.1.4. Kinematic model

The kinematic model assumes that there is no wheel slippage angle whatsoever, hence only being valid for low speed applications. It considers position, velocities, and acceleration, but it does not consider the body's internal forces, inertia, and energy. How the kinematic equations of motion are obtained can be studied in Rajesh Rajamani's book [73]. The final kinematic equations for a front-steering vehicle are:

$$\begin{aligned} \dot{X} &= V\cos(\psi + \beta) \\ \dot{Y} &= V\sin(\psi + \beta) \\ \dot{\psi} &= V\cos(\beta)\frac{\tan(\delta_f)}{L} \end{aligned} \tag{33}$$

7.1.5. Dynamic model

The kinematic model would be enough for an early-stage development of an autonomous vehicle. However, if the speed increases, tires start to slip, and the cornering stiffness coefficient comes into place. This is highly influenced by the tire properties and is one of the main mathematical difficulties when driving on the limits of handling.

[32] *'Path tracking controller for automated driving', Ádám Bárdosa, Sándor Vass, Ádám Nyerges, Viktor Tihanyi and Zsolt Szalay, Department of Automotive Technologies, Faculty of Transportation Engineering and Vehicle Engineering, Budapest University of Technology and Economics*

The dynamic model is based on Newton's equilibrium equations. This leads to an approach that can be used together with the kinematic model, and includes the position, velocity, acceleration, as well as the internal forces, energy, and momentum. How the dynamic equations of motion are obtained can be studied in Rajesh Rajamani's book [73]. The final dynamic equations for a front-steering vehicle can be represented in the form $\dot{x} = Ax + Bu$. If only the steering angle is considered as an input, $\mathrm{u} = \delta_\mathrm{f}$, the result would be:

$$\frac{d}{dt}\begin{Bmatrix} y \\ \dot{y} \\ \psi \\ \dot{\psi} \end{Bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{2C_{\alpha f} + 2C_{\alpha r}}{mV_x} & 0 & -V_x - \frac{2C_{\alpha f} l_f + 2C_{\alpha r} l_r}{mV_x} \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{2l_f C_{\alpha f} - 2l_r C_{\alpha r}}{I_z V_x} & 0 & -\frac{2l_f^2 C_{\alpha f} + 2l_r^2 C_{\alpha r}}{I_z V_x} \end{bmatrix} + \begin{Bmatrix} 0 \\ \frac{2C_{\alpha f}}{m} \\ 0 \\ \frac{2l_f C_{\alpha f}}{I_z} \end{Bmatrix} \delta_\mathrm{f} \tag{34}$$

It can be observed how the cornering stiffness coefficients C_α play an important role in the equations, as they are part of the lateral forces in the tires. These forces are also influenced by the tire slip angles α_f and α_r. Rajesh Rajamani points out that the factor 2 accounts for the fact that there are two wheels in each axis.

7.1.6. Frequency domain

In the case of a simple vehicle model, it is sometimes useful to know the vehicle's transfer function in the frequency domain. The paper by Noor Hafizah et al. goes from the kinematics and dynamics of the model to a transfer function [72].

His vehicle kinematic model is simpler than the one presented above, not considering the slip angle β. It is presented as:

$$\begin{aligned} \dot{X} &= V \cdot \cos(\psi) \\ \dot{Y} &= V \cdot \sin(\psi) \\ \dot{\psi} &= V \cdot \frac{\tan(\delta_\mathrm{f})}{\mathrm{L}} \end{aligned} \tag{35}$$

Simple actuator dynamics are considered so that:

$$\dot{\delta} = -\frac{\delta}{\tau} + \frac{K_a}{\tau}\delta_\mathrm{f} \tag{36}$$

Where $K_a$is the gain assumed to be 1, tau is the actuator constant, and δ_f is the input reference steering angle.

After this, the full model can be transformed to the Laplace domain resulting in a transfer function:

$$G(s) = \frac{Y(s)}{\delta_\mathrm{f}(s)} = \frac{v^2/L}{s^2 \cdot (\tau \cdot s + 1)} \tag{37}$$

7.2. Path Planning

Once the vehicle model is defined, it is necessary to define a path planning protocol so that it can navigate autonomously.

Terms & Definitions:

Graph: A graph is a combination of nodes linked by edges and represented as $\mathcal{G}(V, E)$, where V are the nodes of the graph, and E are the edges, or links, between the nodes.

Complete algorithm: An algorithm is considered complete if it reports whether there is a solution for any input in a finite amount of time.

Optimal algorithm: An algorithm is considered optimal if the solution found is optimal in a finite amount of time.

Probabilistically complete algorithm: An algorithm is considered probabilistically complete if given enough points, the probability of finding an existing solution converges to one.

Dijkstra's Algorithm: Dijkstra's Algorithm is a classical pathfinding algorithm first programmed in 1956. It has typically been used as a method to find the shortest path in a graph, considering the weight in the edges, and finding the route with lowest cost. If there are no weights, the algorithm explores all the nodes in every direction until the goal is achieved.

A:* A* can be considered an improved version of Dijkstra's Algorithm developed in 1968. Instead of analyzing all the possible paths to achieve the target, it measures the cost to the goal and follows the path with lower cost. This is done using heuristics, that estimate the cost functions of each route. If there is no cost function, the A* simply turns into the Dijkstra's Algorithm. A* is commonly used in combination with other techniques when determining the optimal solution from a feasible set.

Configuration Space: Configuration space contains all possible configurations of the mobile robot. It is formed by the free space and the obstacle space.

Convex set: A set $\chi \in \mathcal{R}^2$ is convex if:

$$\theta x_1 + (1-\theta)x_2 \epsilon \chi \tag{38}$$

for all $x_1, x_2 \in \chi$, $\theta \in [0,1]$. An example of a convex set is a convex polytope, which is the region where the optimal solution will be searched, denoted as:

$$P = \{\chi \to \mathcal{R}^n | A_{mxn}x = b\} \tag{39}$$

Convex function: A function $f: \chi \to \mathcal{R}^2$ is convex if the set χ is convex and

$$f(\theta x_1 + (1-\theta)x_2) \leq \theta f(x_1) + (1-\theta)f(x_2) \tag{40}$$

for all $x_1, x_2 \in \chi, \theta \in [0,1]$

Strictly convex function: A function $f: \chi \to \mathcal{R}^2$ is strictly convex if the set χ is convex and

$$f(\theta x_1 + (1-\theta)x_2) < \theta f(x_1) + (1-\theta)f(x_2) \tag{41}$$

for all $x_1, x_2 \in \chi, \theta \in [0,1]$.

Positive definite matrix: A matrix $M \in \mathcal{R}^{nxn}$ is said to be positive definite if:

$$x^T M x > 0 \qquad \forall\, x \in \mathcal{R}^{nx1} \tag{42}$$

Note that $x^T M x$ will be a real number.

Positive semidefinite matrix: A matrix $M \in \mathcal{R}^{nxn}$ is said to be positive definite if:

$$x^T M x \geq 0 \qquad \forall\, x \in \mathcal{R}^{nx1} \tag{43}$$

Note that $x^T M x$ will be a real number.

Path planning for autonomous vehicles is usually hierarchically divided into three sections, mission planning, behavior planning and motion planning.

The mission planning stage is associated with generating a high-level route for the vehicle. The route planner is provided with a previously designed file, known best as Road Network Definition Files (RNDF). This is used as a graph map with nodes and edges to plan the shortest path between two points with classical algorithms such as Dijkstra's or A*. This planner level can be typically seen in GPS devices such as Google Maps. The path generated between your starting point and the target is the mission planner generated route. No motion commands are generated at this level.

The behavior planning section is more specific to the vehicle. Behavior planning is about decision making of the car. This can be approached using Finite State Machines (FSM) that will choose the state depending on the triggering conditions. The actions executed here can be used to decide whether to overtake another vehicle, to stay in the left lane under certain circumstances, or to avoid road blockage. An example of a FSM is presented in *Figure 7-4* from Stanford's Entry in the Urban Challenge DARPA [76].

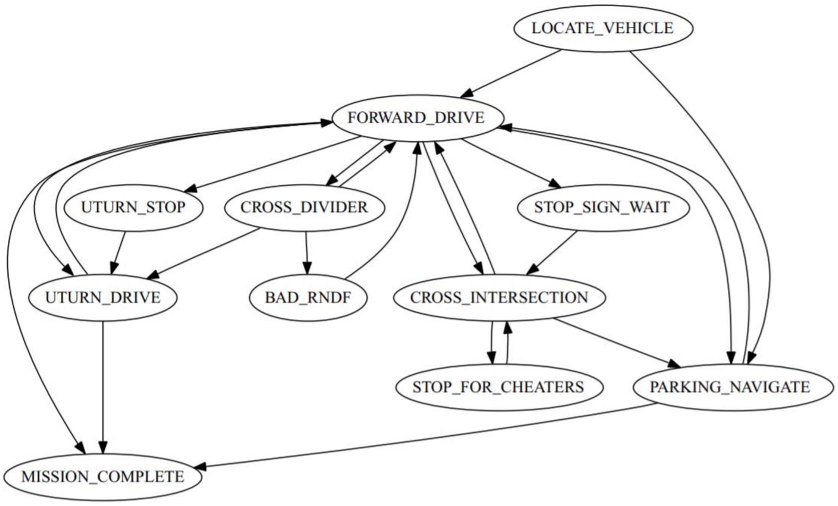


Figure 7-4: Unmanned car driving behavior state machine [76]

7.2.1. Motion planning

Motion planning is the third step in the path planning hierarchy. It is based on obtaining a safe and dynamically feasible trajectory for the self-driving vehicle to go from the initial position to the target. The goal might be different depending on the application. Motion generation tasks sequentially classified in the Kristoffer Bergman's bachelors' thesis 'On Motion Planning Using Numerical Optimal Control' [77] are divided into path generation, path optimization, and trajectory planning, or motion profile planning.

7.2.1.1. Path generation

The purpose of path generation is the creation of a collision free path that connects the initial state with the target. This is done via a path parameter s that represents the progression along the path. The constraints of the vehicle are usually ignored at this step and considered in further tasks.

A path is represented as: $\left(x(s), u(s)\right)$ with $, s \in [0, s_{goal}]$. The variable $x(s)$ is the state as a function of s, and $u(s)$ is the input vector. The parameter s goes from 0 to the goal (s_{goal}) and there is no time dependence.

The path planning problem can be divided into two separate problems. The feasibility problem and the optimization problem. They are described in Steven M. La Valle's 'Planning Algorithms' book [78].

Let χ be a bounded connected subset of $\mathcal{R}^2$. And let the obstacle region and goal region be called χ_{obs} and χ_{goal} and be subsets of χ. Let us denote the free space as χ_{free} and the initial position as $x_{init} \in \chi_{free}$.

Feasibility problem*:* The feasibility problem pretends to find collision free path that starts at x_{init} and ends at the goal region χ_{goal}. It is defined as finding a path $\sigma: [0, s] \to \ \chi_{free}$ such that $\sigma(0) = x_{init}$ and $\sigma(s) \in \chi_{goal}$. Otherwise, it should report failure.

Optimality problem*:* The optimality problem tries to find collision free optimal path that starts at x_{init} and ends at the goal region χ_{goal}. Having the cost function c such that $c: \sum \chi_{free} \to \mathcal{R} > 0$ it is defined as finding a feasible path $\sigma^*: [0, s] \to \ c(\chi_{free})$ such that:

$$\begin{aligned} &\text{i)} \quad \sigma^*(0) = x_{init} \text{ and } \sigma^*(s) \in \chi_{goal} \\ &\text{ii)} \quad c(\sigma^*) = \min_{\sigma \epsilon \sum \chi_{free}} c(\sigma) \end{aligned} \tag{44}$$

And if no path exists report failure.

7.2.1.2. Path optimization

In the path optimization step, the trajectory is smoothed and transformed according to the kinematic and dynamic model of the vehicle. No dynamic obstacles will be considered at this step. The problem can be formulated as an optimal control problem (OCP) to optimize an objective function J minimizing the loss function $L\big(x(t), u(t)\big)$:

$$
\begin{aligned}
&\underset{x(\cdot),u(\cdot),t_f}{\text{minimize}} \quad J = \int_{t_i}^{t_f} L\big(x(t), u(t)\big)dt \\
&subject\ to \\
&\quad x(t_i) = x_{init}, \qquad x(t_f) = x_{final} \\
&\dot{x}(t) = f\big(x(t), u(t)\big), \qquad t \in [t_i, t_f] \\
&x(t) \in \chi_{free}(t) \cap \chi_{valid}, \qquad t \in [t_i, t_f] \\
&u(t) \in \mathcal{U}, \qquad\qquad t \in [t_i, t_f]
\end{aligned} \tag{45}
$$

- x_{init} and x_{final} are the initial and final states.
- t_f the time when the goal state is reached.
- $\chi_{free}(t)$ is the obstacle free region.
- X_{valid} and $\mathcal{U}$ describe the physical constraints on the states and controls, typically described by convex sets.
- $u(t)$ is the control signal, which in the steering car model would be the steering angle and the velocity.

It is assumed that $x(t) \in \chi_{free}(t) \cap \chi_{valid}$ and $x_{term} \in \chi_{free}(t) \cap \chi_{valid}$

Linearization techniques are used in this step to make the optimization problem simpler. Two challenges of this OCP are when the problem is non-convex, and when the system is non-linear, as in the case of the self-driving car.

7.2.1.3. Motion profile generation

Time law is introduced in this task. It consists of computing the reference inputs for the controller of the autonomous car. The trajectory generator will consider the path created previously, as well as the kinematic and dynamic constraints of the robot, to compute a feasible motion profile. The steering angle can sometimes be obtained from the previous steps, thus being $u(t)$ the only unknown variable. Along each path segment, the velocity profile will be:

$$\dot{s}(t) = |v(t)| \tag{46}$$

A trajectory is represented as: $\big(x(s(t)), u(s(t))\big), t \in [t_i, t_f]$, with t varying from the start t_i to the finish t_f.

The task of speed profile generation can be fulfilled using the curvature profile of the generated path. This can be easier in the case of an autonomous race car, which will do an exploration lap first, at a low constant speed, and obtaining a feasible route. The following laps will be completed with the computed speed profile. This idea is introduced by Nitin R. Kapania in his doctor thesis from Stanford [79]. A first solution for the speed profile can be obtained with:

$$v(s) = \sqrt{\frac{\mu g}{|\kappa(s)|}} \tag{47}$$

Where μ is the friction coefficient, and $g = 9.81\ m/s^2$ is the gravitational constant and $\kappa(s)$ is the curvature of the path in s.

In the case of a more generic autonomous vehicle, two approaches could be used. On the one hand, similarly to the Formula Student racecar, it seems reasonable to use the curvature of the path from the local planner to create the motion profile. On the other hand, Anton Anastassov et al. explain in their paper 'Driving Speed Profiles for Autonomous Vehicles' how a planned route can be created from a GPS global motion and updated every predefined time interval [80].

7.2.2. Motion planning algorithms

There is a wide variety of algorithms in mobile robotics with regards to motion planning. The publication 'A Review of Motion Planning for Highway Autonomous Driving' published in the IEEE Transactions on Intelligent Transportation Systems [81] makes a good classification of them based on their approach. It can serve as a starting point to decide on the kind of algorithm desired for the autonomous vehicle.

They are globally classified into six main sets. The ones that can be of greater use for the Formula Student racecar implementation will be explained more extensively, while the rest will be briefly summarized.

Furthermore, it is essential to bear in mind that motion planning algorithms can be combined in order to obtain different results.

7.2.2.1. Sampling Based algorithms

Sampling based algorithms discretize the configuration space. The main goal is to generate a path from the initial state by capturing the free space using a graph search. The three most popular algorithms with this purpose are Probabilistic Road Map (PRM), Rapidly Exploring Random Tree (RRT) and its variant RRT*, as well as Lattice State Planning.

7.2.2.1.1. Probabilistic Road Map

Even though this method might not be the best for an autonomous racecar, it is convenient to understand how it works to learn other sampling-based methods, and it is explained in LaValle's book Chapter 5 [78].

Probabilistic Road Map is a global space discretization algorithm that is usually combined with other path finding algorithms. It is based on discretizing the configuration space evenly on a grid, which is good for small spaces, and can be convenient for a global approach of a racecar in a track. This algorithm cannot be defined as probabilistically complete since it will not report failure if no solution can be found. However, it offers two very useful advantages, as it can be used with systems of a lot of Degrees of Freedom, and the road map can be reused.

The pseudocode of the algorithm is presented in *Algorithm 1: Probabilistic Road Map* . It was obtained from combining LaValle's explanation and the PRM class notes from Penn State [82].

Algorithm 1: Probabilistic Road Map pseudocode

```
Inputs:   $N$ (Number of Samples), $Map$
Outputs:  $PRM\ Graph\ \mathcal{G}$
          $V \leftarrow \emptyset$
          $E \leftarrow \emptyset$
          while $|V| < N$ do
              $x_{rand} \leftarrow random\ point\ in\ \mathcal{C} - Space$
              if $CollisionCheck(x_{rand})$ do
                  $V \leftarrow V \cup \{x_{rand}\}$
              end if
          end while
          for all $x \in V$ do
              $N_q \leftarrow according\ to\ Dist()\ the\ closest\ neighbors\ of\ x$
              for all $x_n \in N_q$ do
                  $LocalPlanner(x, x_n)$
              end for
          end for
```

1. $Dist(x_1, x_2)$: The goal of this function is to get the distance between two points in the configuration space. The distance will necessarily be a real number and it can be obtained in various ways. Two of the most popular ones are shown in *Figure 7-5*:
   a. Euclidean distance:

$$d_{euclidean}(i,j) = \sqrt{\left(x_{i1} - x_{j1}\right)^2 + \left(x_{i2} - x_{j2}\right)^2} \tag{48}$$

   b. Manhattan distance: Based on the building distribution from New York's district:

$$d_{manhattan}(i,j) = \left|x_{i1} - x_{j1}\right| + \left|x_{i2} - x_{j2}\right| \tag{49}$$

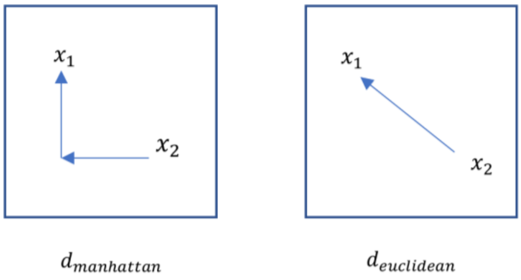


Figure 7-5: Manhattan distance (left) and Euclidean distance (right)

2. $CollisionCheck(x)$: Checks if the statement $x \in \chi_{free}$ is fulfilled.
3. $LocalPlanner(x_1, x_2)$: The goal of this function is to define whether there is a path between two points, or if it is being blocked by an obstacle. A common approach is by constructing a straight line of evenly spaced samples and use the $CollisionCheck()$ function to test if each of those points are free. If they are, then: $E_{new} \leftarrow E \cup \{E(x, x_n)\}$

In lines 1 and 2 an empty graph is generated. After this, a while loop is entered that will occur while the number of nodes in the graph is less than the maximum number of samples. Line 4 represents the generation of a new random point in the configuration space. This is checked with $CollisionCheck(x)$ to see if it is located in empty space, and if this is fulfilled, the point is added to the node list in the graph. This is how random nodes are sampled all over the space.

In order to link the nodes and the edges, another loop is created. For each of the nodes, the nearest ones are identified using the function $Dist(x_1, x_2)$, and with $LocalPlanner(x_1, x_2)$ a test is performed to see if edges in free space can be created between them.

Once the algorithm is finished, the discretized space as a graph is completed with the required number of samples N.

This map can be used by a graph path finding algorithm such as Dijkstra's or A*. The first option, given the fact that all edges have the same weight, it will go from the starting point to the goal by testing all the nodes around the first one. Meanwhile, A* will consider the closeness to the goal, and approach it in that direction.

The whole process is presented in *Figure 7-6*.

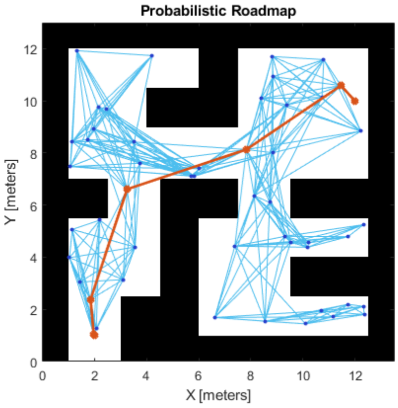


Figure 7-6: Probabilistic Road Map path planning [33]

[33] *https://www.mathworks.com/help/robotics/ug/probabilistic-roadmaps-prm.html*

7.2.2.1.2. Rapidly Exploring Random Tree (RRT)

RRT is a more advanced sampling-based algorithm used in many autonomous vehicle applications. It is called Rapidly Exploring Dense Trees because the main idea is to incrementally build a tree that densely covers the space. It is also introduced in LaValle's book [78]. The feasibility problem will be resolved by this algorithm, but not the optimality problem. Therefore, the first path found to go from the starting point to the goal will be the one being selected.

RRT is a very simple way of finding a path, and it offers the possibility of adding dynamic constraints to the algorithm so that the path is kinematically feasible. In addition, it has been modified into different variants. The most useful one is RRT* introduced by Sertac Karaman and Emilio Frazzoli [83] because it finds a feasible, and optimal path, taking RRT to the next level.

The simplicity and adaptability that rapidly exploring random trees and their variations, makes it an appropriate solution for the Formula Student Racecar models.

The pseudocode of the algorithm is presented in *Algorithm 2: Rapidly Exploring Random Tree pseudocode* and explained. Like with the PRM code, it was obtained from combining LaValle's explanation and the PRM class notes from Penn State [82]. Moreover, the functions defined in this algorithm are the same as the ones used in the Probabilistic Road Maps.

Algorithm 2: Rapidly Exploring Random Tree pseudocode

```
Inputs:     $x_{init}(initial\ position), N(Number\ of\ Samples), \delta(\max distance)$
Outputs:    $RRT\ Graph\ \mathcal{G}$
$V \leftarrow x_{init}$
$E \leftarrow \emptyset$
while $|V| < N$ do
    $x_{rand} \leftarrow random\ point\ in\ \mathcal{C} - Space$
    if $CollisionCheck(x_{rand})$ do
        $x_{near} \leftarrow Nearest\_vertex(x_{rand}, G)$
        if $Dist(x_{rand}, x_{near}) < \delta$ do
            $x_{new} \leftarrow x_{rand}$
        else if $Dist(x_{rand}, x_{near}) > \delta$ do
            $z \leftarrow Point\ between\ (x_{rand}, x_{near})\ so\ that\ Dist(x_{rand}, x_{near}) < \delta$
            $x_{new} \leftarrow z$
        end if
        $LocalPlanner(x_{near}, x_{new})$ do
    end if
end while
```

Firstly, an empty graph is generated with the starting point of the vehicle. Then, for N iterations, the algorithm generates a random sample position and checks if it belongs to free space using $CollisionCheck(x)$. Then, the nearest vertex is found in the graph. The distance between the vertex and the random position is checked, and if it is smaller than a threshold set by the user, the random node is assigned to a new node of the graph. Otherwise, an intermediate point in the same path between the random sample and the nearest node is generated and set as the new node of the graph.

In order to link the nodes and the edges $LocalPlanner(x_1, x_2)$ is used, so that a test is performed to see if edges in free space can be created between them. If the space is obstacle free, the edge is generated.

The whole process is presented in *Figure 7-7*, where all the tree branches can be observed, and the final path is indicated with a red line.

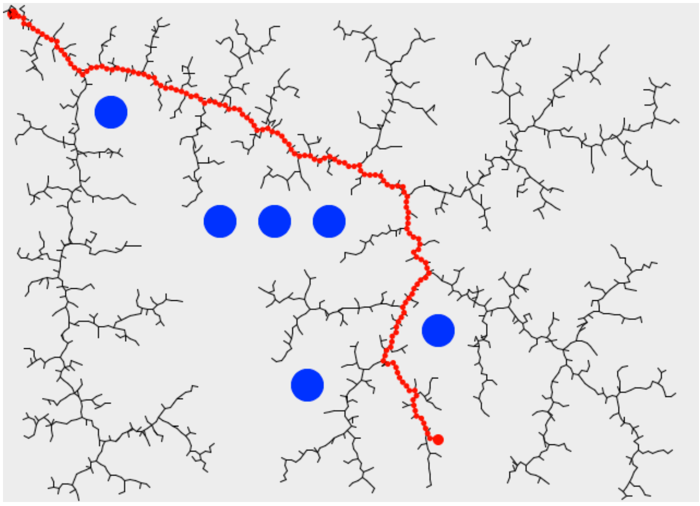


Figure 7-7: RRT path planning [34]

One of the concerns that might arise when applying this algorithm is to realize how is it possible to transform a segment-based path into a feasible trajectory that can be followed by the car.

A simple way of doing this could be using the segment values as inputs, and then smoothing the trajectory using the vehicle's controller. A second approach is to follow a paper by Fang Zhang et al. 'Drift control for cornering maneuver of autonomous vehicles' [84], where after finding the nearest neighbor and generating the edge, the RRT algorithm samples a random input and then propagates the dynamics of the car (with the bicycle model) forward one-time step, starting at x_{near} and resulting in x_{new}. The sampled input accounts for input rate constraints by constraining the sampling domain to be around the previously applied input, which limits the input magnitude and ensures a smooth path. After propagating the node forward, the algorithm ensures the vehicle remains within the track boundary. Thirdly, a curve definition algorithm could be used to obtain a smooth continuous path between the initial state and the goal.

[34] *https://mappingignorance.org/2015/09/04/shortcuts-for-efficiently-moving-a-quadrotor-through-the-special-euclidean-group-se3-1/*

7.2.2.1.3. Connected Cells algorithms

Connected cell-based algorithms is well explained in the article ' Safe Path Planning Using Cell Decomposition Approximation' by Ahmad Abbadi and Václav Přenosil [85].

The main goal of this method is to find free regions in the space, discretized it into a group of cells, and build an adjacency graph from it. Finally, extract the path in combination with another algorithm like A* or Dijkstra's.

There are two ways of doing this, using exact cell decomposition, or approximations. As for the first technique, cells are constructed by avoiding the obstacles and using their vertices (*Figure 7-8*). The composition of these cells will represent the autonomous vehicle's free space. However, the computational power is high, especially when the obstacles are complex. Regarding approximated methods (*Figure 7-9*), computational power is reduced by generating a grid map in the whole space, and then those cells that contain obstacles are excluded. This method depends on the resolution of the grid.

Cell-based decomposition is a popular method that can be useful for route planning and global path planning, however the application is not popular in autonomous cars.

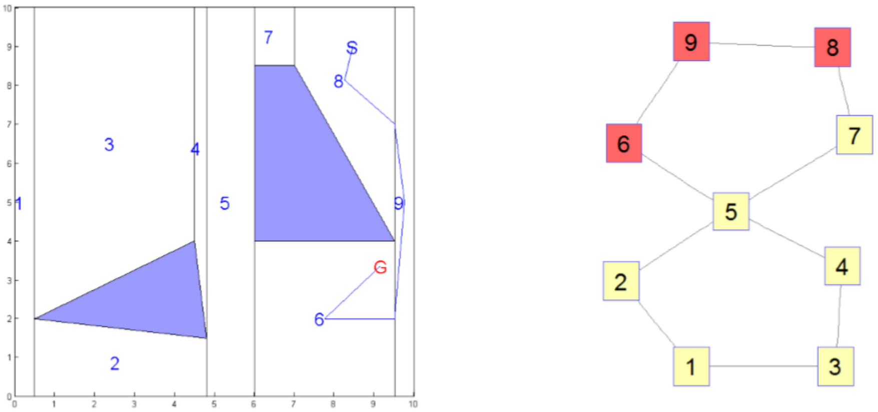


Figure 7-8: Exact cell decomposition map [85]

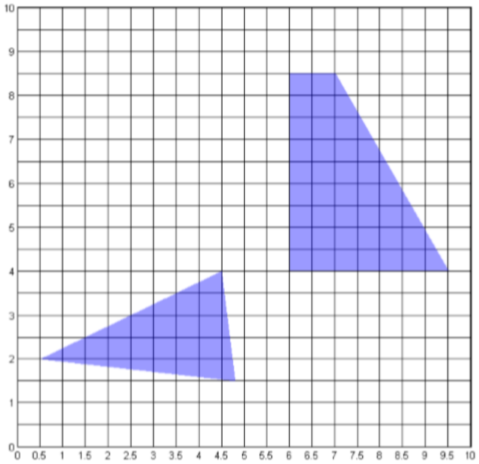


Figure 7-9: Approximate cell decomposition map [85]

7.2.2.2. Potential Field algorithms

Potential Field Algorithms suggest an interesting approach to motion planning by creating a potential energy map that will guide the car to its goal. The main idea is to construct a smooth function over the configuration space that will have high values close to the obstacles and low values further away. The maximum value will be assigned to the initial position of the vehicle and the minimum value will be at the goal. A paper that demonstrates how potential field algorithms can be used in the control loop of Autonomous Vehicles is 'Path planning with fractional potential fields for autonomous vehicles' by Julien Moreau et al. [86].

This method can be difficult to understand at first. An easy way of thinking about it can be: Some piles of books are placed in the floor, that work as the obstacles, some distance away, a large wooden pole, that will simulate the starting point, is set up vertically, and finally, a bed sheet is dropped over the pole and the piles of books, leaving some gap between the books, and having the highest point at the wooden pole. If a marble is dropped at the starting point, it will move through the lowest places in the sheet, reaching the goal, which has the minimum distance to the floor.

The main advantage of the algorithm is that it is a simple, direct approach for avoiding obstacles and following a path based on local sensor data such as position and laser data.

Ideally, the method will work, and global minima will be found. However, the path planning technique might converge to local minima and get stuck there. This is the main reason why artificial potential field algorithms should be used as a local planner and not as a global planner.

Artificial Potential Fields algorithm is explained by the Penn State professor Bob Trenwith in his robotics course [82]. It is divided into four steps. A common approach is presented for each of them.

Attractive potential function: An attractive potential function is constructed for attraction to the goal. The proposed method for 2D is by defining a function such that it is zero at the goal and increases as the car is further away. The function is defined as:

$$f_a(x, y) = \xi(\left|\left|(x, y) - (x_g.y_g)\right|\right|)^2 \tag{50}$$

Being the position of the goal (x_g, y_g), and the function scaling constant ξ.

Repulsive potential function: A repulsive potential function is constructed to represent the repulsion of the obstacles to the car. The input will be the distance in the 2D configuration space to the obstacle $\rho(x, y)$, and the output, the distance to the closest configuration space obstacle. The function is defined as:

$$f_r(x, y) = \begin{cases} \eta\left(\dfrac{1}{\rho(x, y)} - \dfrac{1}{d_0}\right)^2, & \rho(x, y) \leq d_0 \\ 0, & \rho(x, y) > d_0 \end{cases} \tag{51}$$

Being the function scaling constant η and d_0 a threshold that controls the influence of the repulsive potential. It can be observed that the smaller the distance, the greater $f_r(x, y)$ will be.

Addition: The final potential function is constructed simply by adding the two functions:

$$f(x, y) = f_a(x, y) + f_r(x, y) \tag{52}$$

Use on the robot: The final use of it on the robot will be to use the potential field function gradient to decide the direction of motion of the robot.

$$v \propto -\nabla f(x,y) = -\begin{pmatrix} \dfrac{df(x,y)}{dx} \\ \dfrac{df(x,y)}{dy} \end{pmatrix} \tag{53}$$

The robot speed $||v||$ is chosen independently.

This final step indicates a benefit of artificial potential methods because by evaluating only ∇f the robot can plan a path only knowing its position and sensor data.

Figure 7-10 shows the potential field map indicating the start and target position, as well as the obstacles.

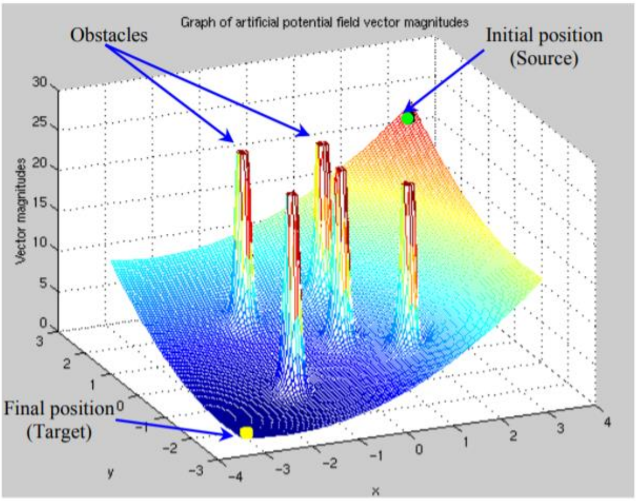


Figure 7-10: Potential fields path planning [35]

[35] *'Mobile manipulator path planning based on artificial potential field: Application on RobuTER', Hargas, Y., Mokrane, A., Hentout, A., Hachour, O., & Bouzouia, B., 'International Conference on Electrical Engineering (ICEE)' 2015*

7.2.2.3. State lattice planning

State lattice planning is a method based on generating the movements offline, and in combination with another algorithm such as RRT, or A*, develop an optimal, feasible path that fulfills the vehicle's constraints. It has mainly been used for off-road driving challenges such as space exploration [87].

This method might be useful for AVs because it saves computing time by defining the offline movements. Instead of inputting a segment-based trajectory to the controller, a model-based trajectory is used for easier control.

The space χ is defined in a regular form, the set χ^{lat} is called state lattices, and $x_t \in \chi^{lat} \subset \chi$ is obtained sampled in a regular form,. The nodes are linked using all possible movements of the car. Depending on the other selected algorithm (RRT, A*…), if there is an obstacle, it is detected, and a feasible and optimal path is selected.

All the paths between nodes are computed offline, allowing for a more efficient implementation of path planning in a robot. Another advantage of lattice states is the possibility of working in multi-dimensional spaces, and the fast and accurate evaluation of motion costs. *Figure 7-11* describes the grid of states linked by all possible movements.

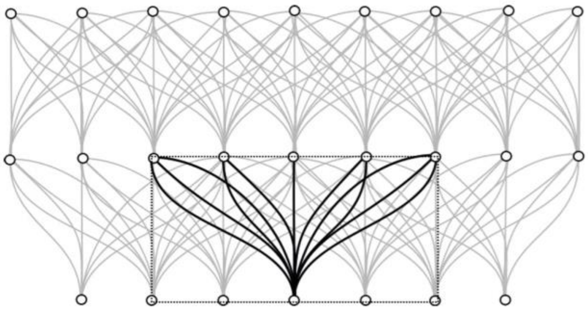


Figure 7-11: State lattice path planning [36]

[36] *'A State Lattice Approach for Motion Planning under Control and Sensor Uncertainty', Adrián González-Sieira, Manuel Mucientes, and Alberto Bugarín, Centro de Investigación en Tecnologias da Información (CITIUS), University of Santiago de Compostela, Spain, 'ROBOT2013: First Iberian Robotics Conference', 2013*

7.2.2.4. Curve definition algorithms

The main goal of curve definition algorithms is to define a smooth and continuous trajectory that can be followed by the vehicle. They are mostly used in combination with the other presented algorithms to optimize and improve the generated path. For example, a good idea could be to use the optimal pathfinder RRT* that returns a segment-based trajectory, with the curve generator that generates a spline in the path points. This outputs an optimal, smooth, and continuous route.

7.2.2.4.1. Dubin's Curves

The simplest approach to curve definition is Dubin's curves for forward movement. The path created will be formed by circular arcs and straight lines, that will compose the optimal trajectory. According to this technique, the optimal path will always have one of the following representations: RSR, RSL, LSR, LSL, RLR, LRL. Where R means right turn, L means left turn, and S means Straight Driving. Two examples are shown in *Figure 7-12.*

This method will fulfil the maximum steering angle constraints of the autonomous car. In combination with Reeds-Sheep curves, it is a good way of approaching car parking, as presented in 'Path-Planning for Autonomous Parking with Dubins Curves' by Christoph Siedentop et al. [88]. However, it might not be successful when smoothing the intersection between a line and an arch. Moreover, the limited movements that can be carried out with this method is a negative feature for driverless-car application.

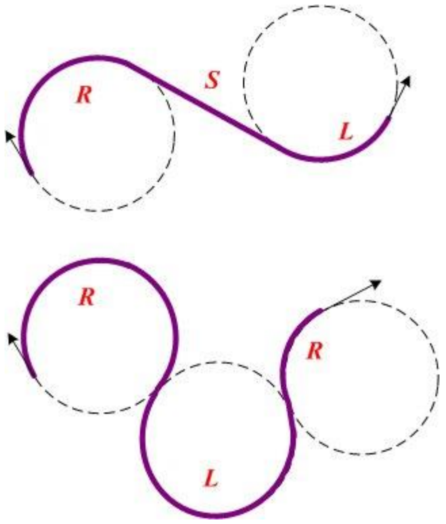


Figure 7-12: Dubin's curves examples. Right-Straight-Left (top) and Right-Left-Right (bottom) [37]

7.2.2.4.2. Polynomial splines

The polynomial splines method is based on composing a polynomial by considering the points detected by the path generation algorithms. There are a wide variety of ways to use polynomials to generate trajectories: using polynomials function of the curvature, or with cubic or quantic polynomials. Two simple approaches are presented that might be useful for path planning by calculating the unknown coefficients from a polynomial $(a_0, a_1, \dots, a_n)$.

[37] *'Smooth 3D Dubins Curves Based Mobile Data Gathering in Sparse Underwater Sensor Networks', Wenyu Cai and Meiyan Zhang, 'Sensors 2018', 2018*

On the one hand, if the car starts from a static state and finishes in a static state, it is possible to obtain the coefficients using the initial and final states. The initial velocity $\dot{x}(t = 0) = 0$. The final velocity $\dot{x}(t = t_f) = 0$. The following example shows this procedure for a third order polynomial, function of the distance to the origin (*Figure 7-13*).

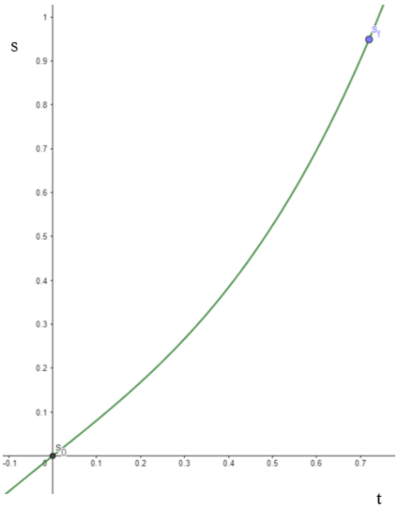


Figure 7-13: Third order polynomial, origin at s_0 and final state at s_f

$$\begin{gathered} s(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \\ \dot{s}(t) = a_1 + 2a_2 t + 3a_3 t^2 \\ s_o = a_0 \\ s_f = a_0 + a_1 t_f + a_2 t_f^2 + a_3 t_f^3 \end{gathered} \tag{54}$$

From velocity equation we obtain:

$$\begin{gathered} 0 = a_1 \\ 0 = a_1 + 2a_2 t_f + 3a_3 t_f^3 \end{gathered} \tag{55}$$

The coefficients will be:

$$\begin{gathered} a_0 = s_o \qquad a_2 = \frac{3}{t_f^2}(s_f - s_0) \\ a_1 = 0 \; a_3 = \frac{-2}{t_f^2}(s_f - s_0) \end{gathered} \tag{56}$$

On the other hand, it is also possible to develop a third order polynomial P_3 based on four points, A(x_o, y_0),B(x_1, y_1),C(x_2, y_2),D. (x_3, y_3), (*Figure 7-14*). The following system can be solved using Gauss method.

$$\begin{gathered} P_3(x_0) = a_0 + a_1 x_0 + a_2 x_0^2 + a_3 x_0^3 = y_0 \\ P_3(x_1) = a_0 + a_1 x_1 + a_2 x_1^2 + a_3 x_1^3 = y_1 \\ P_3(x_2) = a_0 + a_1 x_2 + a_2 x_2^2 + a_3 x_2^3 = y_2 \\ P_3(x_3) = a_0 + a_1 x_3 + a_2 x_3^2 + a_3 x_3^3 = y_3 \end{gathered} \tag{57}$$

If the polynomial is required to be function of the distance to the origin s, then it would be parametrized assiging s to the distance to the origin.

$$s(t) = \sqrt{(x(t) - x_0)^2 + (y(t) - y_0)^2} \tag{58}$$

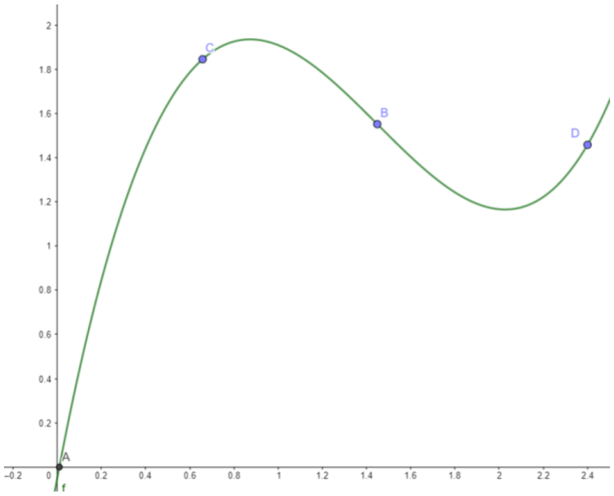


Figure 7-14: Third order polynomial with three points A,B,C,D

In 'Trajectory Generation for Autonomous Vehicles' by Vu Trieu Minh from the University of Tallinn [89] a second order polynomial trajectory generation approach is presented and tested for faster computation. It is done specifically for the simplified kinematic bicycle model (*see Bicycle model*).

It is essential to bear in mind that the higher the order of the polynomial, the smoother the trajectory will be, as well as the velocity and acceleration profile. Quintic polynomials have been used for a safe and comfortable acceleration profile for the driver [90]. On the flip side, the number of operations and the processing time needed will also be higher.

7.2.2.4.3. Bezier curves

Another well-known approach to curve generation is using Bezier curves [91]. Using them offers great advantages, as they are easy to manipulate. They are defined by more than two control points that will be modified by the obstacle avoidance algorithm. A curve of degree n has $(n+1)$ control points, and is defined by the following expression:

$$s(t) = \sum_{i=0}^{n} P_i \cdot B_{i,n}(t)\,;\ i \geq 3 \tag{59}$$

Where P_i are the control points, and $B_{i,n}(t)$ the Bernstein polynomials such that:

$$B_{i,n}(t) = \binom{n}{i} \cdot (\frac{t_1 - t}{t_1 - t_0})^{n-i} \cdot (\frac{t - t_0}{t_1 - t_0})^{i};\ i \in \{0, \dots, n\} \tag{60}$$

The most interesting features of Bezier curves in terms of motion planning are:

- They always pass through the first and last control points, P_0 and P_f
- They are always tangent to the line between P_0 and P_1, as well as to the line between P_{n-1} and P_n.
- They are always convex to the control polygon, the polygon formed by the control points.

7.2.2.5. Variational techniques: trajectory optimization, optimal control

Optimization is a well-known technique in many aspects of the world, ranging from economics to engineering. Path Planning is one of the many applications of this field, and therefore vast research has been done to find the most efficient techniques. The main advantage it offers is the high speed and efficiency when converging to an optimal solution. According to the paper 'A review of Motion Planning for Highway Autonomous Driving' [81], optimization based trajectory generation tries to minimize a cost function in in a sequence of states variables under a set of constraints. These constraints can be linear, or non-linear, which will vary the difficulty of the problem. The solver methods are similar to the ones used for path optimization. Based on this, three main approaches can be found, Linear Programming (LP), Nonlinear Programming and Quadratic Programming (QP).

It is convenient to note that convex functions have two interesting properties. These will help knowing whether a global minimum can be reached just by understanding the cost function:

1) The intersection of an arbitrary number of convex sets is a convex set.
2) If $f_1 \dots f_n$ are convex functions, then $\sum \alpha_i f_i$ is a convex function for all $\alpha_i \geq 0; i \in \{1, \dots, N\}$.

An optimization problem is convex, if the objective function is convex, and the feasible set is convex. And in these types of problems, local minima are global optimizers. This means that every time the problem is solved, the solution will be a global minimum. Therefore, it is always beneficial to determine a cost function in a feasible region that is convex.

7.2.2.5.1. Linear Programming

The easiest kind of optimization is linear programming (LP) [92]. The optimization problem in this case is convex, what makes it easy to solve. The objective function and the constraints are linear. Any Linear Program can be represented in the following way:

$$\begin{aligned} &\text{minimize} \quad J = c_1x_1 + c_2x_2 + \ldots + c_nx_n \\ &subject\ to \\ &\quad a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ &\quad a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \dots \\ &\quad a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n = b_m \\ &\quad a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n = b_m \end{aligned} \tag{61}$$

and $x_1 \geq 0, x_2 \geq 0 \dots x_n \geq 0$

The simplified form of this would be:

$$\begin{aligned} &\text{minimize} \quad J = c^T x \\ &subject\ to \\ &\quad Ax = b \\ &\quad \text{and } x \geq 0 \end{aligned} \tag{62}$$

where $c^T \in \mathcal{R}^{1xn}$, $A \in \mathcal{R}^{mxn}$, $x \in \mathcal{R}^{nx1}$ and $b \in \mathcal{R}^{mx1}$

$linprog()$ can be used in MATLAB to solve linear programming problems. However, autonomous cars will usually have nonlinear model and constraints. Thus, there are two options, linearize the model dynamics, or use Nonlinear Programming (NLP), which will be presented next.

7.2.2.5.2. Nonlinear Programming

Nonlinear Programming (NLP) is an alternative to linear programming when it comes to cost functions and constraints that are nonlinear. It is a good way of finding optimal solutions in more complex models [93]. The downside of NLP is that the problem is non-convex, and the solution will depend on the initial conditions. If the initial conditions are set so that they are close to a local minimum, this local minimum will be found instead of a global minimum. On the other hand, if the initial state is close to the global minima, global minima will be found first.

An example in MATLAB has been implemented to show NLP procedure using $fmincon()$ function. Upper and lower boundaries are needed as inputs for the function, and just and equality constraint was used for simplicity.

The problem is defined as:

$$\begin{gathered} \text{minimize } f(x) = sin(x_1) + 0.1x_2^2 + 0.05x_1^2 \\ subject\ to \\ (x_1+8)^3 - x_2 = 0 \\ x_1 \in [-10,10], x_2 \in [-10,10] \end{gathered} \tag{63}$$

The results obtained where graphically represented. *Figure 7-15, left,* shows the values of the objective function in a 3-dimensional grid, where the global and local minima can be observed. The code is found in *APPENDIX IV: Nonlinear programming example.*

The 2D iso-line graph is presented to see the solution to the optimization plotted. The '$*$' denote the initial states, and the 'o' denotes the solutions.

If the optimizer is started at $[-8,0]$, it converges to a local minima at $[-7.07,0]$, however, if the initial conditions are $[-4,0]$, then, the optimizer converges at global minima at $[1.427,0]$

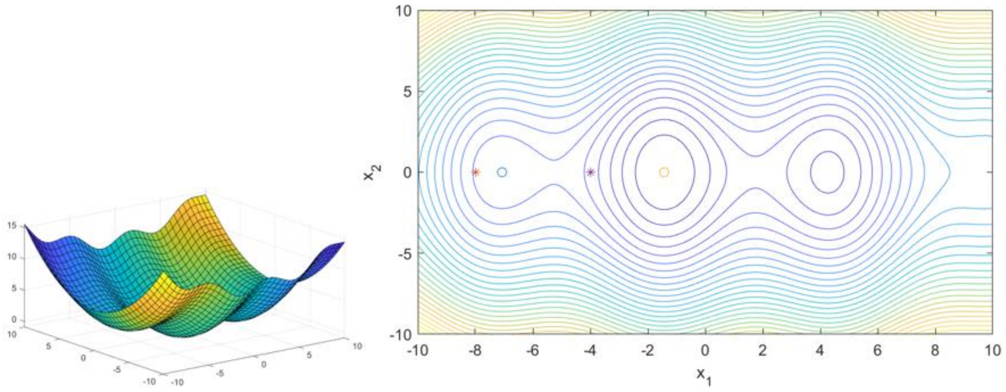


Figure 7-15: Nonlinear programming example in MATLAB

7.2.2.5.3. Quadratic Programming

Quadratic programming (QP) is an interesting optimization technique as it solves a convex problem converging into global minima. Therefore, the initial conditions will only influence the computation time of the solution. The objective function in a Quadratic Programming problem will be formed by square and/or linear and/or bilinear terms, and the constraints must be linear. Just like in the other two optimization methods, MATLAB has a built-in function to test and work with QP, called $quadprog()$. More advanced popular QP solvers are Gurobi [94] and CPLEX by IBM [95].

The standard form of quadratic programming is:

$$\begin{aligned} &\text{minimize}\, J = \frac{1}{2}x^T Q x + c^T x \\ &subject\ to \\ &\qquad\qquad Ax \geq b \\ &\qquad\qquad A_{eq}x = b_{eq} \end{aligned} \tag{64}$$

Where $Q \in \mathcal{R}^{nxn}$ is the quadratic term positive semi-definite matrix, $c \in \mathcal{R}^{nx1}$ is the linear coefficient vector, $A \in \mathcal{R}^{mxn}$ is the inequality coefficient matrix, $b \in \mathcal{R}^{nx1}$ is the inequality right hand side vector and $x \in \mathcal{R}^{nx1}$ is the variable vector.

An example in MATLAB is presented for better understanding. The problem is defined as:

$$\begin{aligned} &\text{minimize}\ 0.4x_1^2 - 5x_1 + x_2^2 - 6x_2 + 50 \\ &subject\ to \\ &\qquad\qquad x_2 - x_1 \geq 2 \\ &\qquad\qquad x_2 + 0.3x_1 \geq 8 \end{aligned} \tag{65}$$

and $input\ constraints\ x_1 \in [0,10], x_2 \in [0,10]$

The first step is to transform it to the solver representation of the problem. Each solver might have a different way of presenting it; MATLAB uses the standard form in eq. (64).

$$A = \begin{bmatrix} 1 & -1 \\ -0.3 & -1 \end{bmatrix};\ b = \begin{bmatrix} -2 \\ -8 \end{bmatrix}; Q = \begin{bmatrix} 0.4 & 0 \\ 0 & 1 \end{bmatrix} \cdot 2; c = \begin{bmatrix} -5 \\ -6 \end{bmatrix} \tag{66}$$

Code can be found in *APPENDIX V: Quadratic programming example*. The solver output is shown in graph, where the two inequality constraints are plotted as well, and the feasible region is located on top of them. The '$*$' denotes the minimum states, and the 'o' denotes the solution.

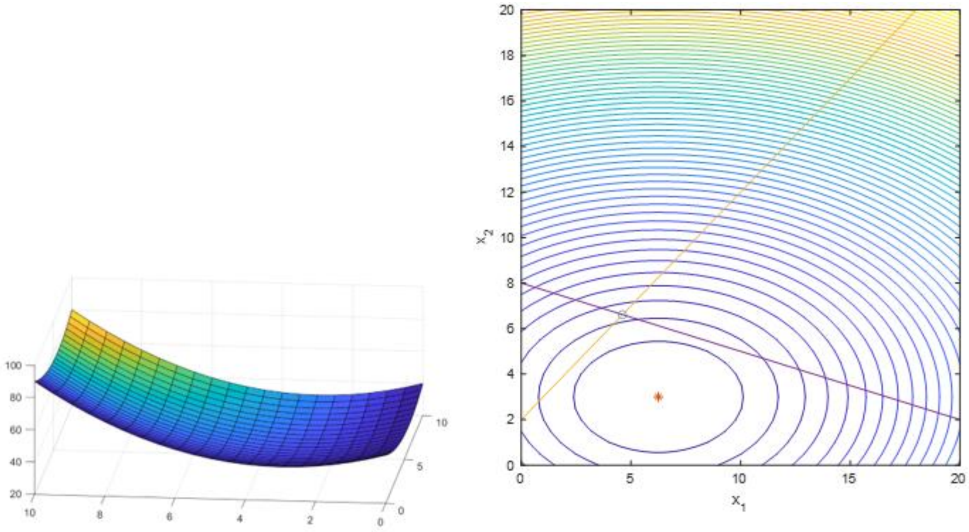


Figure 7-16: Quadratic programming example in MATLAB

It can be observed that the optimal solution is closest to the minimum of the objective function, and at the same time it is fulfilling the inequality constraints.

To put it in the case of a self-driving car, the constraints would represent the obstacles, while the minimum is the target point to follow the path.

7.3. Motion Control

After generating a feasible trajectory with the motion profile, the vehicle must follow it as precisely as possible. This is done by measuring the lateral displacement error of the car with respect to the input trajectory. The system will be divided into three main blocks, the controller, the car model, that will transform the output of the controller into motion, and the sensors, which will detect the real position of the vehicle.

Terms & Definitions:

x = f(x(t), u(t)): Shortened representation of the vehicle's state where $x(t) = [x_1(t), x_2(t) \dots, x_n(t)]$ and $u(t) = [u_1(t), u_2(t) \dots, u_m(t)]$. The full representation would be $x(x(t), u(t)) = ([x_1(t), x_2(t) \dots, x_n(t)], [u_1(t), u_2(t) \dots, u_m(t)])$.

Euler approximation: Euler's Method is an integration method to numerically solve differential equations. It divides the time between the start and finish in n steps of equal length h. The function approximation will be in the form:

$$x(h+1) = x(h) + h \cdot f(t_{h-1}, x_{h-1}) \tag{67}$$

Algebraic Ricatti Equation: The Algebraic Ricatti Equation is a type of nonlinear equation that must be solved in infinite horizon optimization problems such as Linear Quadratic Regulators. It can be used in the continuous time space as well as in the discrete time space, and it is solved starting from the final state.

Continuous space:

$$A^T S + SA - SBR^{-1}B^T S + Q = 0 \tag{68}$$

Discrete space:

$$S = A^T SA - (A^T SB) \cdot (R + B^T SB)^{-1}(B^T SA + Q) + Q \tag{69}$$

S is the unknown matrix while A, B, R and Q are real known matrices.

The position control loop can generally be represented as *Figure 7-17*, where the input to the system is $x_{ref}(x(t), u(t))$, the output of the controller is the steering angle δ, the velocity profile v, and the output of the vehicle model, which is taken by the sensor, is $x(x(t), u(t))$. The chances are high that there is a disturbance factor that will affect the vehicle model.

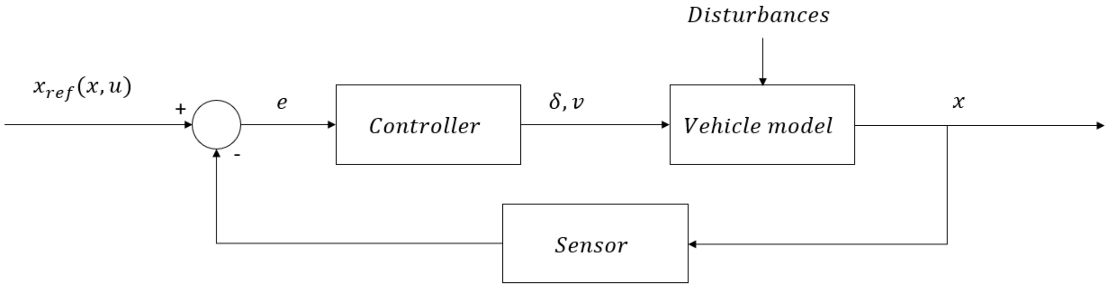


Figure 7-17: Autonomous system control loop

Just like with path generation, motion controllers can be combined in a single vehicle, using one technique for lateral displacement and a different one to take over the brake and throttle system.

There is a wide range of control methods. The most used ones for unmanned driving can be optimization based, like Linear and Nonlinear Model Predictive Control (MPC) and Linear Quadratic Regulators (LQR). Classical methods are still used like PID Control for its feasibility and robustness.

Regarding the requirements that must be satisfied for a self-driving racecar, they are mainly a fast response and small overshoot. Meanwhile, in a commercial vehicle the response must be smoother and should have low steady state error to follow a road lane precisely.

7.3.1. Optimization based control

Optimization control methods have the same principle as optimization-based path planning (*see Variational techniques: trajectory optimization, optimal control*). The difference will be that instead of finding the optimal and feasible path, the solver will output the optimal control command for the vehicle so that it can approach the desired state. The information in this section will be closely related to the book by Borrelli, F. referenced in Pedro F. Lima's Masters' Thesis from KTH [96]. Notes from the University of Michigan have also been used for the mathematical formulations.

Control techniques for solving optimization problems include Model Predictive Control and Linear Quadratic Regulator. The way they solve the problem might be different, but their cost function has a similar form.

In the continuous time space, the objective function $J(x,u)$ is formulated as.

$$J(x(t),u(t)) = \int_0^T (x^T(t) \cdot Q(t) \cdot x\,(t) + u^T(t) \cdot R(t))dt \qquad (70)$$

- $x(t) \in \mathcal{R}^{nx1}$: State vector.
- $u(t) \in \mathcal{R}^{mx1}$: Input vector.
- $Q(t) \in \mathcal{R}^{nxn}$: State weight matrix. Symmetric, positive semidefinite and bigger or equal to zero.
- $R(t) \in \mathcal{R}^{mxm}$: Input weights matrix. Symmetric, positive definite and bigger strictly bigger than zero.

For a better interpretation of the cost function, a two state, two input, objective function will be formulated as an example.

State vector: $x(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix}$. Input vector: $u(t) = \begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix}$. State weights: $Q(t) = \begin{bmatrix} q_{11}(t) & 0 \\ 0 & q_{22}(t) \end{bmatrix}$. Vector weights: $R(t) = \begin{bmatrix} r_{11}(t) & 0 \\ 0 & r_{22}(t) \end{bmatrix}$

$$J(x(t),u(t)) = \int_0^T (q_{11} \cdot x_1(t)^2 + q_{22} \cdot x_2(t)^2 + r_{11} \cdot u_1(t)^2 + r_{22} \cdot u(t)^2\,)dt \qquad (71)$$

Note that if the values of $Q(t)$ are considerably bigger than the values of $R(t)$, the states will drive the cost function, so the state term must go to zero as fast as possible. This will make the controller fast and aggressive.

Meanwhile, if the values of $R(t)$ are considerably bigger than the values of $Q(t)$, the control inputs will drive the cost function. This will give less importance to the states, and it will be the inputs that really matter. The controller will be slow and conservative.

Moreover, it is important to mention that in literature, the cost function may appear either in the matrix form or in the extended form from the example.

It is the labor of the engineer to select an appropriate cost and state matrix in each case.

7.3.1.1. Model Predictive Control (MPC)

Model Predictive Control is the term that defines a group of different control strategies, that obtain the optimal control signal for a process based on the model of a system. It has been used in many different fields, including chemical processes and oil refineries.

The main idea under MPC is shown in *Figure 7-18*. The engineer provides the controller with the mathematical model of a system, the constraints, and a cost function. The controller will minimize the objective function and generate a control signal for the next N steps (called prediction horizon). From these N output signals, the first one will be selected, and the process will be repeated at each timestep t.

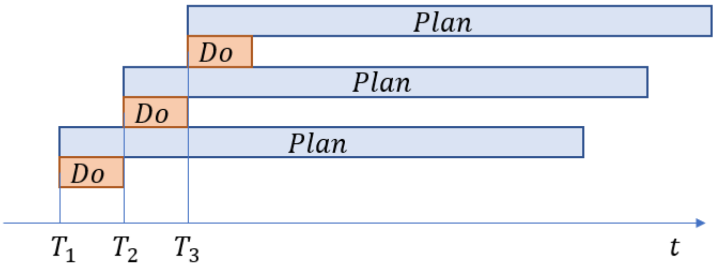


Figure 7-18: MPC graphic explanation

The paper 'The Application of Model Predictive Control (MPC) to Fast Systems such as Autonomous Ground Vehicles (AGV)' by Solomon Sunday Oyelere [97] decouples the MPC controller in three different components *Figure 7-19*, the Autonomous Guided Vehicle (AGV) model, the cost function and constraints, and the optimizer. The first two parts will be the ones that categorize the problem into nonlinear, or linear.

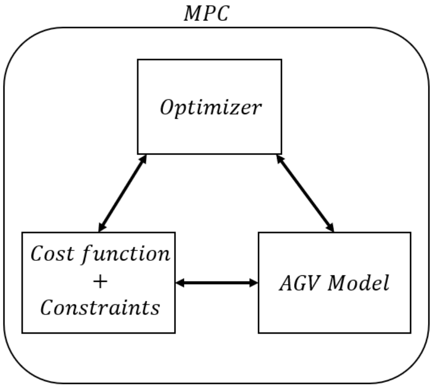


Figure 7-19: Components of an MPC controller

The procedure to apply it to a car-like model is presented. The simplified model will be (*see Car Model)*:

$$\begin{aligned} \dot{x}(t) &= v(t) \cdot \cos(\psi(t)) \\ \dot{y}(t) &= v(t) \cdot \sin(\psi(t)) \\ \dot{\psi(t)} &= v(t) \cdot \frac{\tan(\delta(t))}{L} \end{aligned} \tag{72}$$

The model is discretized using Euler Approximation in the following way:

$$\begin{aligned} x(k+1) &= x(k) + \Delta t \cdot v(k) \cdot \cos(\psi(k)) \\ y(k+1) &= y(k) + \Delta t \cdot v(k) \cdot \sin(\psi(k)) \\ \psi(k+1) &= \psi(k) + \Delta t \cdot v(k) \cdot \frac{\tan\big(\delta(k)\big)}{L} \end{aligned} \tag{73}$$

The optimal output values u^*, y^* with the discretized model will be:

$$\begin{aligned} u^*(k) &= \tilde{u}^*(k) - u_{ref}(k) \\ x^*(k) &= \tilde{x}^*(k) - x_{ref}(k) \end{aligned} \tag{74}$$

Ideally, $x^*(k) = 0$ and $u^*(k) = 0$.

Figure 7-17 represented the general implementation of trajectory following controllers. Nevertheless, there is also the possibility of generating the output using the input vector $u_{ref}(t) = f\,(v_{ref}(t), \delta_{ref}(t))$ and the double loop from *Figure 7-20*.

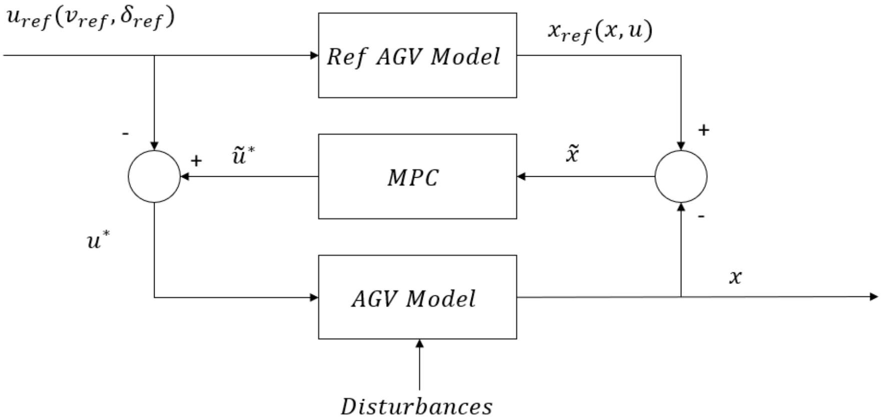


Figure 7-20: Decomposed control loop [97]

The discrete objective function for MPC can be represented as:

$$J(x,u) = \sum_{j=1}^{N} (x^T(k+j|k) \cdot Q(k) \cdot x\,(k+j|k) + u^T(k+j|k) \cdot R(j) \cdot u(k+j|k)) \tag{75}$$

- $x(k+j|k)$: Planned vehicle's state $k+j$ computed at timestep k.
- $u(k+j|k))$: Planned control signal $k+j$ computed at timestep k.

The solution to the optimization problem will be:

$$u^* = [u^*(k|k), u^*(k+1|k), \ldots, u^*(k+n|k)\,] \tag{76}$$

Only the first control action $u^*(k|k)$ will be considered and inputted into the vehicle plant. The state vector $x^*(k)$ can be obtained by introducing u^*into the vehicle model.

Depending on the model, constraints and application, Model Predictive Control can be divided into Nonlinear MPC and Linear MPC.

7.3.1.1.1. Nonlinear MPC

In the case of nonlinear MPC, nonlinear system models with nonlinear constraints are considered. In addition, the cost function will be non-quadratic, therefore the problem is complex, and it will require high computational power, but the solution will be accurate.

The nonlinear MPC optimization problem is formulated as:

$$\begin{gathered} \underset{x(k),u(k)}{\text{minimize}} \quad J(x,u) \\ subject\ to \\ x(t_{init}) = x_{init} \\ \dot{x}(x,u,k) = 0 \end{gathered} \tag{77}$$

7.3.1.1.2. Linear MPC

The alternative for Nonlinear Model Predictive control in linear systems with linear constraints is called Linear MPC. The optimization problem will be a quadratic program, thereby a convex problem.

However, since car dynamics are nonlinear, an intermediate step of linearization needs to be performed before the vehicle model reaches de controller.

Linearization can be performed about a point, or about a trajectory. It is the second type that will be of greater relevance for a self-driving car. A way of Jacobian linearization is presented, based on notes from the University of Michigan 'Self-Driving Cars: Perception and Control' .

The problem presented is to linearize a system $\dot{x}(t) = f\big(x(t), u(t)\big)$ about a trajectory $(x_e(t), u_e(t))$.

Consider $\delta x(t) = x(t) - x_e(t)$ and $\delta u(t) = u(t) - u(t)$.

$$\dot{x}(t) = f\big(x(t), u(t)\big) = f(\delta x(t) + x_e(t), \delta u(t) + u(t)) \tag{78}$$

If the Taylor expansion is applied ignoring the higher order terms:

$$\dot{x}(t) \approx f\big(x_e(t), u_e(t)\big) + \delta x(t) \cdot \frac{\partial f}{\partial x} |_{\, x_e(t),u_e(t)} + \delta u(t) \cdot \frac{\partial f}{\partial u} |_{\, x_e(t),u_e(t)} \tag{79}$$

- $\delta x(t) \cdot \frac{\partial f}{\partial x} |_{\, x_e(t),u_e(t)} = A(t) \in \mathcal{R}^{nxn}$: The Jacobian matrix of $\dot{x}(t)$ at $(x_e(t), u_e(t))$ with respect to the state variables $x(t)$.
- $\delta u(t) \cdot \frac{\partial f}{\partial u} |_{\, x_e(t),u_e(t)} = B(t) \in \mathcal{R}^{nxm}$: The Jacobian matrix of $\dot{x}(t)$ at $(x_e(t), u_e(t))$ with respect to the input variables $u(t)$.

The input to the MPC controller will be the error:

$$\dot{x}(t) - \dot{x}_e(t) = \dot{\delta} x(t) = A(t) \cdot \delta x(t) + B(t) \cdot \ \delta u(t) \tag{80}$$

The discrete model, linearized about the trajectory $(x_e(t), u_e(t))$ will be represented as:

$$x(k+1) = A \cdot x(k) + B \cdot u(k) \tag{81}$$

The linear MPC optimization problem in the discrete space is formulated as:

$$\begin{gathered}\underset{x(k),u(k)}{\text{minimize}} \quad J(x,u) \\ subject\ to \\ x(k_0) = x_{init} \\ x(k+1) = A \cdot x(k) + B \cdot u(k)\end{gathered} \tag{82}$$

The main variation between NLMPC and LMPC is the model used in optimization. In the first case, the non-linearized model is used, while in the second one the model is linearized about the input trajectory.

7.3.1.2. Linear Quadratic Regulator (LQR)

LQR control is a similar approach to MPC. It is simpler, so it takes less time to compute, and it does not need an initial condition. On the negative side, it generally cannot deal with constraints. However, research has been done in this area and solutions such as 'Constrained LQR' [98] have been proposed.

Notes on LQR are based on a lecture of Christopher Lum, PhD graduate from the University of Washington, of Linear Quadratic Regulators [99], and University of Michigan class notes.

The objective function used in LQR depends on the final state and inputs at N step. It can be represented as follows:

$$\begin{gathered}J(x,u) = \sum_{j=1}^{N-1} (x^T(k) \cdot Q(k) \cdot x\,(k) + u^T(k) \cdot R(t) \cdot u(k)) \\ +(x^T(N) \cdot Q(N) \cdot x\,(N) + u^T(N) \cdot R(N) \cdot u(N))\end{gathered} \tag{83}$$

The optimization problem will be formulated as:

$$\begin{gathered}\underset{u}{\text{minimize}} \quad J(x,u) \\ subject\ to \\ x(k+1) = A(k) \cdot x(k) + B(k) \cdot u(k)\end{gathered} \tag{84}$$

The solution to the problem will be:

$$u^*(k) = -K \cdot x(k) \tag{85}$$

with $K = R^{-1} \cdot B^T \cdot S$.

Where S is the solution to the Ricatti Algebraic Equation, that must be solved backwards (starting at step N) eq. (68).

The steps can be summarized into the following way:

1) Linearize the vehicle model to obtain $A(k) \in \mathcal{R}^{nxn}$ and $B(k) \in \mathcal{R}^{mxn}$ matrices.
2) Choose appropriate $Q(t)$ and $R(t)$ matrices.
3) Solve the Ricatti Algebraic Equation for S.
4) Compute optimal gain K.
5) There will be multiple solutions in steps 3) and 4) therefore the K solution that yields a stable system will be selected.

The controlled system can be observed in *Figure 7-21*. Ideally, the gain matrix K should only be solved once as it does not depend on the initial state. The 'Constrained Iterative LQR for On-Road Autonomous Driving Motion Planning' paper [98] shows a way of doing this iteratively so that it can be adjusted on varying conditions.

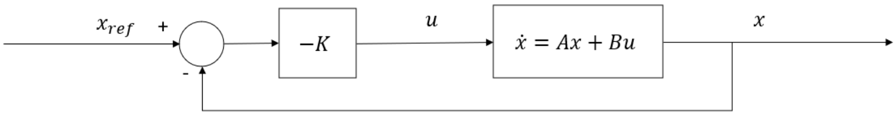


Figure 7-21: LQR control loop

7.3.2. Proportional Integral Derivative Controller (PID)

PID control is a classical control method commonly used in variations such as PI or PD. This control strategy control has numerous advantages. The most important one is the simplicity of implementation. Moreover, it will work for almost any system, as it is not model based, and the error is the only variable that matters. The vehicle model is considered a black box, so the dynamics can be either linear or nonlinear.

On the negative side, the tuned gains might not be ideal for every single condition. Nevertheless, studies have been carried out for tuning to occur iteratively depending on the vehicle's conditions [100]. In addition, PID control does not work well for highly non-linear systems, and it might not be a good option for conditions with important external disturbances.

These benefits and disadvantages make PID controllers a good option to be implemented in combination with other controllers such as MPC.

In the time domain, a PID controller is represented in the time domain the following way:

$$K_P \cdot e(t) + K_I \cdot \int_0^t e(t)dt + K_d \cdot \frac{de(t)}{dt} \tag{86}$$

Taking it to the Laplace domain, the controller looks like:

$$K_P \cdot s + K_I \cdot \frac{1}{s} + K_d \cdot s \tag{87}$$

Proportional (K_P): The proportional component minimizes the error of the system multiplying the error by the parameter K_P. It will increase the response speed and reduce the steady state error. However, it will increase instability. Moreover, in high amplitude oscillations, it will amplify the oscillations as well.

Integral (K_I): Acts with the integral of the error. The main function is to decrease steady state error by penalizing the sum of error. The response will converge to the reference trajectory.

Derivative (K_d): The Derivative term reduces the overshoot by recognizing the speed at which it approaches the reference (the slope of the response). It will not affect the steady state error, but it will decrease the speed of the system. It is important to consider that the Derivative term will affect high frequency noise amplifying it, solved by using a lowpass cutoff filter. The PID controller, in the frequency domain, with this filter will look like:

$$K_P + K_I \cdot \frac{1}{s} + K_d \cdot \frac{N}{1 + N \cdot \frac{1}{s}} \tag{88}$$

These terms individually might give certain advantages but will be of little use for an automated driving system. Controllers are commonly developed by combining them. P, PD, PI and PID are presented. D controller is also explained for a better understanding.

The car model transfer function (*see Frequency domain*) will be considered. The open loop root locus diagram is represented as *Figure 7-22*.

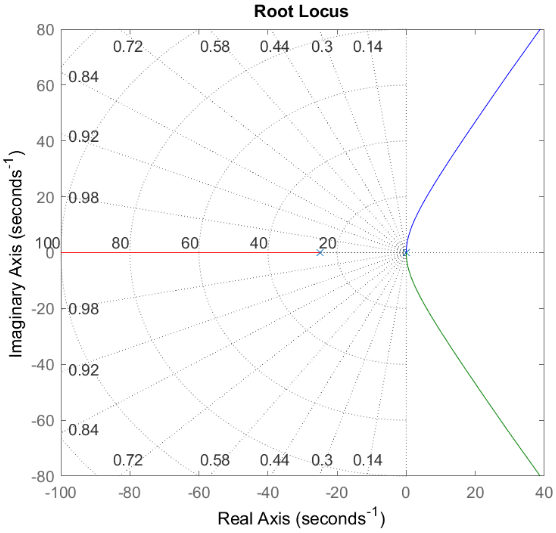


Figure 7-22: Open loop car transfer function root locus

The root locus with damping rate ξ different from 1 is in the positive part from the real axis component, where the system is unstable. The main goal of the controller will be to partially have these branches with negative real component.

7.3.2.1. P Controller

Using a P Controller will not be a feasible solution, as it will only vary the position of the dominant poles along the branches. The system will be unstable for whichever value that is introduced for K_P.

7.3.2.2. D Controller

A D controller will introduce a zero to the system in the origin. This will make the system controllable and the branches will now be in the negative side of the real axis (*Figure 7-23*). The parameter K_D should be tuned according to the requirements. It is important to consider the filter mentioned previously, as it will get rid of high frequency noise and disturbances.

However, D controllers are rarely used separately. They measure the change in error, but not the error itself, therefore a derivative control will not bring the system to a setpoint. This fact can be emphasized in the case when the error is constant. Then, the output of the derivative controller will be zero.

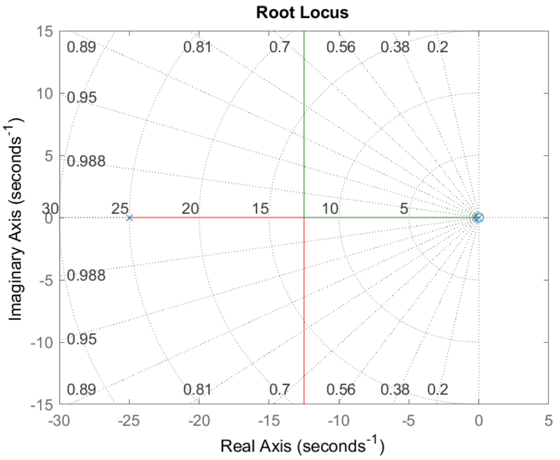


Figure 7-23: D controlled open loop car transfer function root locus PD Controller

The PD Controller solves the limitation from the D controller, by introducing a Proportional component for overshoot correction. A Proportional Derivative controller is optimal for fast and unstable systems, as it will improve the transient state. However, it will not accurately correct the steady state error.

$$C(s) = K_P + K_d \cdot \frac{N}{1 + N \cdot \frac{1}{s}} \tag{89}$$

The time response of a PD controller is presented in *Figure 7-24*. In the case of a self-driving racecar, the parameters must be tuned properly so that the overshoot is as small as possible while keeping a fast response. After this, the PD is a good option to be used in the vehicle.

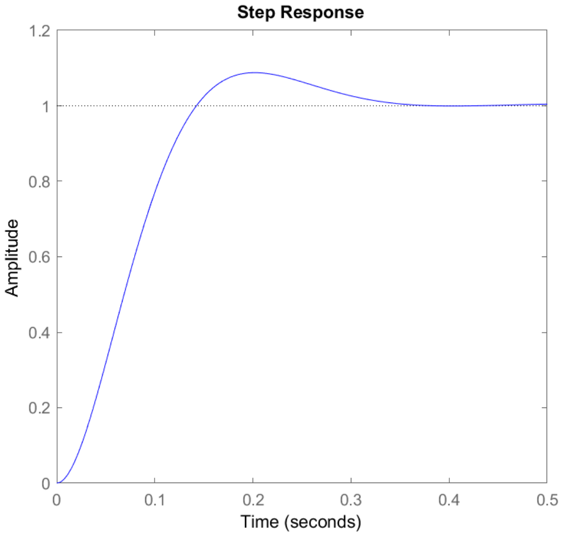


Figure 7-24: PD controlled closed loop car transfer function step response

7.3.2.3. PI Controller

Proportional Integral Controllers improve the steady state response while keeping minimum variations in the transitory state. With the transfer function of the vehicle used in this example, PI control would not be a feasible solution. It would introduce another zero at the origin making the system even more unstable (Figure 7-25).

$$C(s) = K_P + K_I \cdot \frac{1}{s} \tag{90}$$

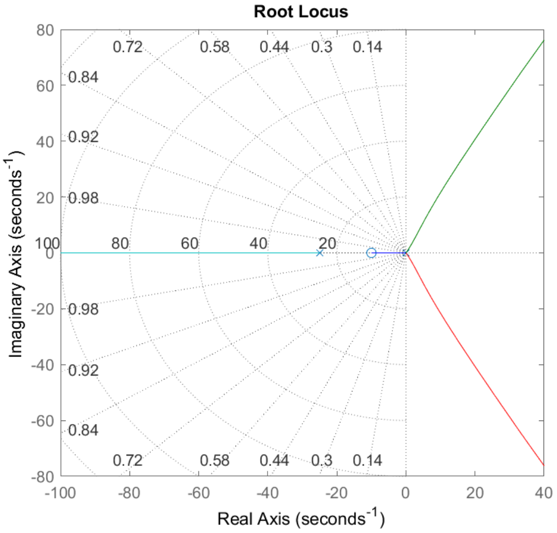


Figure 7-25: PI controlled open loop car transfer function root locus

7.3.2.4. PID Controller

Proportional Integral Derivative Controllers combine the steady state error reduction from a PI controller but makes possible the control with the D term. The result is very similar to *Figure 7-25*. Even though the control will be good, it makes the system more complex, so the response-complexity tradeoff should be studied. This is the main reason why it is not as used as the other controllers.

$$C(s) = K_P + K_I \cdot \frac{1}{s} + K_d \cdot \frac{N}{1 + N \cdot \frac{1}{s}} \tag{91}$$

8. Formula Student Vehicles Overview

The aim of this section is to provide some real examples of autonomous vehicles that illustrate how the different sensing, perception and control techniques studied in this project fit together.

Terms & Definitions:

RANSAC: The Random Sample and Consensus algorithm is used to discard outliers to prevent them from having an impact on a generic model. It works by estimating the data set parameters randomly and counting the number of points within a distance range, the inliers. The estimation that has the most data samples will be the one that is considered. Model estimation process is shown in *Figure 8-1*.

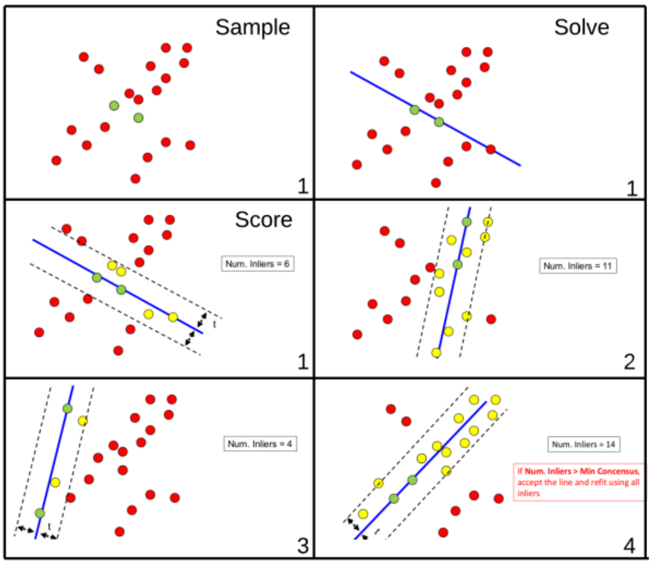


Figure 8-1: RANSAC algorithm steps (from top left to bottom right) [38]

Support Vector Machine (SVM): A Support Vector Machine is a supervised learning model that divides a set of samples. It creates a plane between the two classes that has maximum distance between them. In *Figure 8-2* the two groups, and the two dimensional planes that were guessed. Only the red one would be valid.

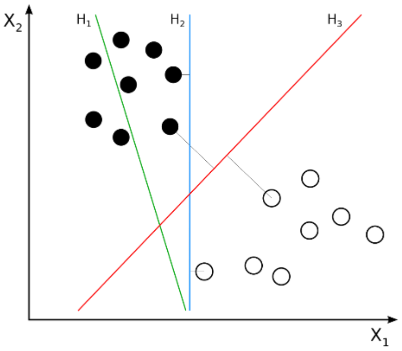


Figure 8-2: SVM simple example. First try is H_1, second try is H_2, last and best try H_2 [39]

[38] *https://bitesofcode.wordpress.com/tag/opencv/*

[39] *https://en.wikipedia.org/wiki/Support_vector_machine*

8.1. AMZ

The AMZ Team, from ETH Zurich, has been an important figure in terms of autonomous vehicles in the Formula Student Germany competition. They have published their autonomous system approach in the paper 'AMZ Driverless: The Full Autonomous Racing System' [101]. Their autonomous system structure can be observed in *Figure 8-3*.

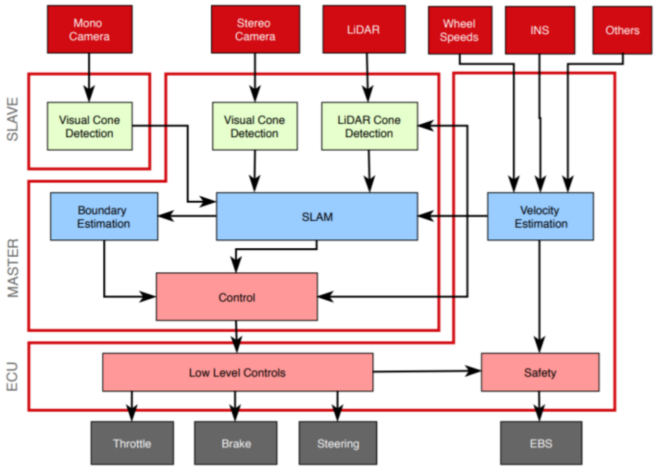


Figure 8-3: AMZ autonomous system diagram [101]

8.1.1. Sensing

The vehicle has a LIDAR placed on the front wing for cone detection. Its restricting factor is the vertical resolution as it is necessary to capture the whole region of interest with cones in it. The point cloud is cleaned for noise and ground using Himmelsbach's algorithm [6].

The image-based detection consists of three CMOS cameras. Two of them are used for close cone detection using stereo imaging, while the third monocular camera detects cones that are further away. This camera also has its own slave computer.

The other sensors incorporated in the car are resolvers for each the motors and a non-contact optical ground speed sensor. Resolvers are odometry sensors similar to electronic transformers that measure angular speed. It also has an encoder to measure the changes in the steering wheel. The sensors are integrated using an Extended Kalman Filter.

8.1.2. Perception

Perception of the AMZ team racecar has a central SLAM algorithm that collects all the data. Two pipelines are designed for object detection.

The first one is LIDAR-based, where cone estimations are done using clustering and regression in the point clouds. Then, a custom CNN detects the cone color based on the laser points intensity. Finally, the cone data is taken to the SLAM algorithm.

The second path is image-based. Here, the convolutional neural network YOLOv2 is used for its reliability. Cone detection is performed in one of the stereo images and bounding boxes are propagated to the second image. Features are matched for cone detection using SIFT and similar descriptors.

8.1.3. Control

The AMZ team car will only perform motion planning during its first lap when it computes the entire map of the track. After this, the loop is closed and the vehicle switches from SLAM to Localization mode.

During this first lap a variation of RRT* algorithm is used. First of all, the space between the cones is discretized in triangles because it is the only available space where the car can move. Then, multiple paths are generated, and the least feasible ones are discarded. Lastly, a cost function detects the optimal branch for the car. Once the first lap is completed, the track will be known to the car and a nonlinear MPC controller takes the control.

8.2. Beijing Institute of Technology

Beijing Institute of Technology Formula SAE team also published their AV approach, with less detail than AMZ studied above [102]. The autonomous system diagram is represented in Figure 8-4.

8.2.1. Sensing

Regarding sensing of the unmanned car, it has the two stereo cameras, the LIDAR, and the GPS-INS, as well as some odometry sensors in the wheels and brakes.

The ground noise is also being removed using a RANSAC based method, a region of interest is being extracted from the point cloud, and the LIDAR and GPS data is being fused.

8.2.2. Perception

Cone detection is done based on two principles. The first one is using the point clouds and using the Euclidean distance in clusters to detect the cones. The second one, instead of implementing a CNN, uses the stereo cameras and a support vector machine classifier with HOG descriptors (similar to SIFT descriptors) to detect the cones in the image. The colors are distinguished by avoiding the use of the V components in the HSV color space for less effect of the illumination.

8.2.3. Control

Little is known about the control part of this vehicle. However, it does mention a similar approach to the AMZ vehicle of computing the path in the first lap and following it with the controller the next laps.

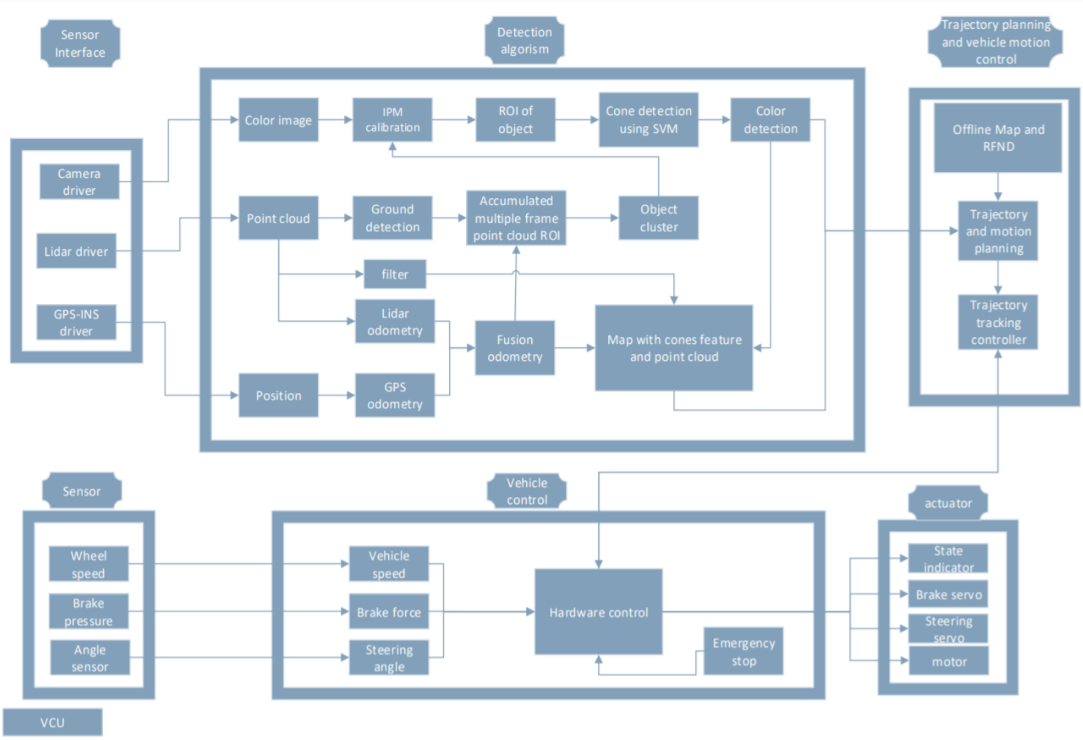


Figure 8-4: Beijing Institute of Technology autonomous system diagram [102]

8.3. TUW Racing

The Technical University of Vienna published in 2017 their approach for the Formula Student Autonomous Vehicle [103].

It is important to note that all the approaches so far have been built over a ROS (Robot Operating System) framework. This allows the data to be managed in a simple way by using a node publisher/subscriber policy. The TUW ROS diagram is shown in *Figure 8-5*.

8.3.1. Sensing

Sensors in the car include a LIDAR, a stereo camera, GPS, IMU and Odometry, used in combination with an Extended Kalman Filter for a SLAM algorithm.

8.3.2. Perception

Their perception method is based on an EKF-SLAM algorithm, and they have two techniques for cone detection. The first one is by extracting object centers using LIDAR, projecting them into the stereo images, and creating image patches around. Then, color thresholding is used to detect whether the object inside is a cone. The second way of identifying cones is by color thresholding both stereo images and triangulating the distance to the cones.

8.3.3. Control

Path planning is done simply by computing the center between the two cones and projecting a normal vector. Instead of using this just in the first lap, they use this for each of them, and combine this path planning with an MPC controller.

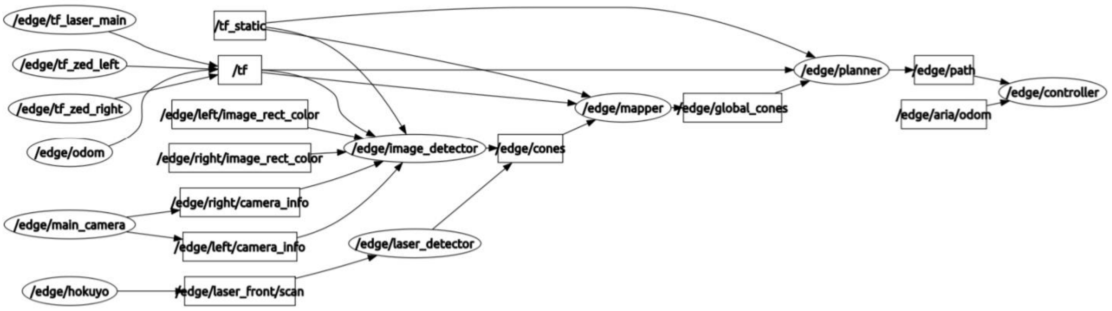


Figure 8-5: TUW autonomous system ROS diagram [103]

8.4. University of Western Australia Formula SAE

University of Western Australia presented in 2014 a different approach presented in their paper [104].

8.4.1. Sensing, Perception and Control

The sensing system was based on GPS and IMU fused together, and then a Kalman Filter integrates the LIDAR to detect the track cones.

The laser sensor will detect the cones, and consider them as posts, and they will be placed in the GPS map. Waypoints are set in the map and a cubic spline is fitted through these waypoints that will be the trajectory followed by the vehicle. The map is updated every time a position is measured, and the motion is controlled using a PID controller.

9. Conclusion

All the different components of vehicle autonomy have been covered in the project. Grouped in three main blocks, sensing, perception, and control, the main technologies have been presented to give a global understanding of autonomous cars. Practical examples in some of the principal sections have been implemented using tools such as MATLAB, Google Collaboratory and Python that will hopefully serve as a reference for future research.

A suggested approach to start with the implementation of an autonomous system to the Formula Student racecar is to study and integrate solutions similar to those already available, and then analyze different variations that will change the performance of the car.

10. APPENDIXES

10.1. APPENDIX I: ORB feature extraction

```
cone = cv2.imread('cone.png');
orb = cv2.ORB_create(nfeatures=100)
kp1, des1 = orb.detectAndCompute(cone, None);
img=cv2.drawKeypoints(cone,kp1,cone,flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
cv2.imshow("ORB result", img);
cv2.waitKey(0);
cv2.destroyAllWindows()
```

10.2. APPENDIX II: BruteForce matching

```
cone = cv2.imread('cone.png');
multiple_cones = cv2.imread('multiple_cones.png');
orb = cv2.ORB_create(nfeatures=500)
kp1, des1 = orb.detectAndCompute(cone, None);
kp2, des2 = orb.detectAndCompute(multiple_cones, None);
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
matches = bf.match(des1,des2)
result = cv2.drawMatches(cone,kp1,multiple_cones,kp2,matches, None)
cv2.imshow("Matching result", result);
cv2.waitKey(0);
cv2.destroyAllWindows()
```

10.3. APPENDIX III: YOLOv3 implementation

First, it is necessary to clone the GitHub repository, enable the prerrequisites GPU, OpenCV and cuDNN, the Nvidia CUDA Deep Neural Network library, and build the darknet file.

```
!git clone https://github.com/AlexeyAB/darknet
%cd darknet
!sed -i 's/OPENCV=0/OPENCV=1/' Makefile
!sed -i 's/GPU=0/GPU=1/' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/' Makefile
!make
```

An imshow() helper function is done as well for plotting the results.

```
def imShow(path):
  import cv2
  import matplotlib.pyplot as plt
  %matplotlib inline
  # This will plot the results directly under the cell
  image = cv2.imread(path)
  # Opens the image from specified path
  height, width = image.shape[:2]
  # Extracts the first two sizes of the image: height & width
  resized_image = cv2.resize(image,(3*width, 3*height), interpolation = cv2.INTER_CUBIC)
  # Increases the size of the image for visualization
  fig = plt.gcf()
  fig.set_size_inches(18, 10)
  plt.axis("off")
  plt.imshow(cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB))
  # Shows and converts the colorspace
  plt.show()
```

Afterwards, the dataset "obj" file must be copied to the Colab "/content/" workspace to unzip it. For this, it is kept in a Google Drive folder. The commands are as follows:

```
!cp /content/drive/My\ Drive/CNN2/obj.zip ../
!unzip ../obj.zip -d /content/darknet/data/
```

The code for it has been changed in our to suit my needs. It is provided below, and comments have been added for a better understanding of the code.

"generate_train.py"

```
import os
# First, an empty image list is created
image_files = []
# Next, the working directory is going to be changed, in this case, being in /content/, it wi
ll go to '/darknet/data/obj' to find the files
os.chdir(os.path.join("darknet","data", "obj"))
# Each of the image names in the folder will be appended to the imag_files[] array
for filename in os.listdir(os.getcwd()):
   if filename.endswith(".jpg"):
      image_files.append("darknet/data/obj/" + filename)
# Goes back to '/data/'
os.chdir("..")
# Writes the path for each image in a file called 'train.txt'
with open("train.txt", "w") as outfile:
   for image in image_files:
      outfile.write(image)
      outfile.write("\n")
   outfile.close()
os.chdir("..")
```

"generate_train.py" must be run while being in the /content/ directory if the above code is run without modifications.

```
!python generate_train.py
```

There are three files to be edited before training is started.

The first two must be created, using the names "obj.names" and "obj.data". These, after being edited on the computer, can be dragged and dropped at the collaboratory '/content/darknet/data/' directory.

"obj.data" will contain the number of classes, 1 for balloons only, and the path of the files that are going to be used, the validation set path may be written, however as this hasn't been done in the project it will be ignored.

“obj.data”

```
classes = 1
train = /content/darknet/data/train.txt
valid = /content/darknet/data/test.txt
names = /content/darknet/data/obj.names
```

“obj.names” will only contain the name of the class that the CNN is going to be trained for, Ballon in this case.

“obj.names”

```
Balloon
```

The third file will be the one that modifies the parameters of YOLO, “yolov3_custom2.cfg”.

```
[net]
# Testing
#batch=1
#subdivisions=1
# Training
batch=64
subdivisions=16
width=416
height=416
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.001
burn_in=1000
max_batches = 2000 #number of classes times 2000
policy=steps
steps=1600,1800 #80% of max batches, and 90% of max batches
scales=.1,.1
```

There are some other parameters that need to be changes as well, more specific to de CNN itself.

[yolo]
mask = 6,7,8
anchors = …
classes=1
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1
	In each of the [yolo] layers, the number of classes must be changed. As only balloons are going to be detected, it is set to 1.
	It can be observed how the thresholds are set for the network. The object will be detected if the IoU value is greater than 0.7.

[convolutional]
size=1
stride=1
pad=1
filters=18
activation=linear
	In each of the [convolutional] layers that are located before a [yolo] layer, it can be seen that the activation function is linear, referring to a ReLU function. As mentioned in the Darknet GitHub repository [66], it is necessary to change the number of filters to:
	$3 * (number\ of\ classes\ + 5)$
	If there is just one class, the number of filters will be 18.

This third configuration file must be dragged to "content/darknet/cfg".

Once the three files have been loaded into the workspace, and in order to take shorter for the network to train, it is convenient to use transfer learning by starting from some already pretrained weights. The final command to start training and output a log file is:

```
!/content/darknet/darknet detector train /content/darknet/data/obj.data /content/darknet/cfg/yolov3_custom2.cfg /content/darknet/darknet53.conv.74 > /content/darknet/train.log -dont_show
```

10.4. APPENDIX IV: Nonlinear programming example

```
%% Problem formulation
% min sin(x1) + 0.1*x2^2 + 0.05*x1^2
% subject to
% -5<=x1<=1
% -3<=x2<=3
fun = @(x) sin(x(1)) + 0.1*x(2)^2 + 0.05*x(1)^2;
X0 = [-8, 0];
lb = [-10, -10];
ub = [10, 10];
A = [];
b = [];
Aeq = [];
beq =[];
nc = @nonlcon;
 x_min = fmincon(fun,X0,A,b,Aeq,beq,lb,ub)
 x1_lim = 10;
x2_lim = 10;
x1_range = [-x1_lim:0.05:x1_lim];
x2_range = [-x2_lim:0.05:x2_lim];
 % Plot 3D
[x1,x2] = meshgrid(x1_range, x2_range);
x3 = sin(x1) + 0.1.*x2.^2 + 0.05.*x1.^2;
C = x1.*x2;
figure(1)
surf(x1,x2,x3,C)
% Plot 2D
figure(2)
contour(x1,x2,x3,30)
% Plot point & constraints
hold on
plot(x_min(1),x_min(2), 'o')
plot(X0(1),X0(2), '*')
xlim([-x1_lim,x1_lim])
ylim([-x2_lim,x2_lim])
xlabel('x_1')
ylabel('x_2')
function [C, Ceq] = nonlcon(x)
    Ceq = (x(1)+8)^3 - x(2);
    C = [];
end
```

10.5. APPENDIX V: Quadratic programming example

```
%% Problem formulation
% min 0.4*x1^2 - 5*x1 + x2^2 - 6*x2 + 50
% subject to
% x2 - x1 >= 2
% 0.3*x1 + x1 >= 8
% x1,x2 [0,10]
 fun = @(x) 0.4*x(1)^2 - 5*x(1) + x(2)^2 - 6*x(2) + 50;
lb = [0, 0];
ub = [10, 10];
H = [0.4, 0; 0, 1]*2;
A = [1, -1; -0.3, -1];
b = [-2, -8];
f = [-5, -6];
Aeq = [];
beq = [];
x_min = quadprog(H,f,A,b,Aeq,beq,lb,ub)
x_min_real = quadprog(H,f)
x1_lim = 20;
x2_lim = 20;
x1_range = [0:0.5:x1_lim];
x2_range = [0:0.5:x2_lim];
 % Plot 3D
[x1,x2] = meshgrid(x1_range, x2_range);
x3 = 0.4.*x1.^2 - 5.*x1 + x2.^2 - 6.*x2 + 50;
C = x1.*x2;
figure(1)
surf(x1,x2,x3,C)
% Plot 2D
figure(2)
contour(x1,x2,x3,60)
% Plot point & constraints
hold on
plot(x_min(1),x_min(2), 'o')
plot(x_min_real(1),x_min_real(2), '*')
plot(x1_range,x1_range+2);
plot(x1_range,8-0.3*x1_range);
xlim([0,x1_lim])
ylim([0,x2_lim])
xlabel('x_1')
ylabel('x_2')
```

11. References

[1] A. Lafrance, "Your Grandmother's Driverless Car," The Atlantic, June 2016. [Online]. Available: https://www.theatlantic.com/technology/archive/2016/06/beep-beep/489029/. [Accessed 11 2020 June].

[2] "An Oral History of the DARPA Grand Challenge, the Grueling Robot Race That Launched the Self-Driving Car," Wired, 8 March 2017. [Online]. Available: https://www.wired.com/story/darpa-grand-challenge-2004-oral-history/. [Accessed 11 June 2020].

[3] R. B. R. a. S. Cousins, "3D is here: Point Cloud Library (PCL)," IEEE International Conference on Robotics and Automation (ICRA), May 2011. [Online]. Available: https://pointclouds.org/. [Accessed 11 June 2020].

[4] T. Trafina, "Construction of 3D Point Clouds Using LiDAR Technology - Bachelors' Thesis," Czech Technical University, Prague, 2016.

[5] A. Pomares, J. L. Martínez, A. Mandow, M. A. Martínez, M. Morán and J. Morales, "Ground Extraction from 3D Lidar Point Clouds with the Classification Learner App," Mediterranean Conference on Control and Automation , Croatia, 2018.

[6] M. Himmelsbach , F. Hundelshausen and H.-J. Wuensche, "Fast Segmentation of 3D Point Clouds for Ground Vehicles," IEEE Intelligent Vehicles Symposium , San Diego, California, 2010.

[7] O. Sorkine-Hornung and M. Rabinovich, "Least-Squares Rigid Motion Using SVD," ETH, Zurich, 2017.

[8] D. Chetverikov, D. Svirko and D. Stepanov, "The Trimmed Iterative Closest Point Algorithm," IEEE, Prague, 2002.

[9] W. Burgard, C. Stachniss and M. Bennewitz, *Introduction to mobile robotics, Iterative Closest Point,* George Mason University, Department of computer science.

[10] P. Jensfelt, D. Kragic, J. Folkesson and M. Björkman, "A Framework for Vision Based Bearing Only 3D SLAM," Centre for Autonomous System Royal Institute of Technology , 2014.

[11] "Stereo Vision Lecture," University of Stanford, [Online]. Available: http://vision.stanford.edu/teaching/cs131_fall1314_nope/lectures/lecture9_10_stereo_cs131.pdf. [Accessed 11 June 2020].

[12] "Stereo Vision Lecture," University of Michigan, [Online]. Available: https://web.eecs.umich.edu/~jjcorso/t/598F14/files/lecture_1027_stereo.pdf. [Accessed 11 June 2020].

[13] J. González, "Desarrollo de algoritmos de un coche autónomo para la competición de SEAT autonomous driving challenge - Bachelors' Thesis," Tecnun, Universidad de Navarra, San Sebastian, 2019.

[14] H. Kong, H. C. Akakin and S. E. Sarma, "A Generalized Laplacian of Gaussian Filter for Blob Detection and Its Applications," IEEE Transactions on Cybernetics , 2013.

[15] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," International Journal of Computer Vision, Vancouver, B.C, Canada, 2004.

[16] E. Karami, S. Prasad and M. Shehata, "Image Matching Using SIFT, SURF, BRIEF and ORB: Performance Comparison for Distorted Images," arXiv:1710.02726 [cs.CV], Ebrahim Karami, Siva Prasad, and Mohamed Shehata, 2017.

[17] M. W. SCHWARZ, W. B. COWAN and J. C. BEATTY, "An Experimental Comparison of RGB, YIQ, LAB, HSV, and Opponent Color Models," ACM Transactions on Graphics, University of Waterloo, Canada, 1987.

[18] Ansys, "Autonomous Vehicle Radar: Improving Radar Performance with Simulation," 2018. [Online]. Available: https://www.ansys.com/-/media/ansys/corporate/resourcelibrary/article/autonomous-vehicle-radar-aa-v12-i1.pdf. [Accessed 11 June 2020].

[19] X. Meng, H. Wang and B. Liu, "A Robust Vehicle Localization Approach Based on GNSS/IMU/DMI/LiDAR Sensor Fusion for Autonomous Vehicles," Sensors 2017, Northeastern University, Shenyang, China , 2017.

[20] "Centimeter-accurate GPS for self-driving vehicles," Society of Autonomotive Engineers (SAE), [Online]. Available: https://www.sae.org/news/2016/10/centimeter-accurate-gps-for-self-driving-vehicles. [Accessed 11 June 2020].

[21] D. Yuan, X.-m. Cui, S.-y. Wang, Y.-h. Qiu, G. Wang and J.-j. Jin, "The Coordinate Transformation Method and Accuracy Analysis in GPS Measurement," International Conference on Environmental Science and Engineering , Beijing, China, 2011.

[22] "Understanding Sensor Fusion and Tracking," Matlab, 21 October 2019. [Online]. Available: https://www.youtube.com/watch?v=6qV3YjFppuc. [Accessed 11 June 2020].

[23] F. Castanedo, "A Review of Data Fusion Techniques," Hindawi Publishing Corporation, The Scientific World Journal , Bilbao, Spain, 2013.

[24] S. Kamçı, D. Aksu and M. A. Aydin, "Lane Detection For Prototype Autonomous Vehicles," arXiv:1912.05220 [cs.CV], Istanbul University-Cerrahpasa, Istanbul, Turke, 2019.

[25] P. Dwivedi, "CarND-Advanced Lane Finder-P4," GitHub, 9 March 2017. [Online]. Available: https://github.com/priya-dwivedi/CarND/tree/master/CarND-Advanced%20Lane%20Finder-P4. [Accessed 11 June 2020].

[26] "MATLAB Color Thresholder," MATLAB, [Online]. Available: https://www.mathworks.com/help/images/ref/colorthresholder-app.html. [Accessed 11 June 2020].

[27] "OpenCV Library," OpenCV, [Online]. Available: https://opencv.org/. [Accessed 11 June 2020].

[28] A. Jakubovic and J. Velagic, "Image Feature Matching and Object Detection using Brute-Force Matchers," Sarajevo, Bosnia and Herzegovina, 2018.

[29] A. E. Stefan Zickler, "Detection of Multiple Deformable Objects using PCA-SIFT," Carnegie Mellon University, Pittsburgh, U.S.A, 2007.

[30] B.MEHLIG, in *Artifical Neural Networks*, University of Gothenburg, Gothenburg, Sweden, pp. 9-14.

[31] K. Fukushima, "Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position," Biol. Cybernetics 36, 193 202 , Kinuta, Setagaya, Tokyo, Japan, 1980.

[32] Y. LeCun, "Backpropagation Applied to Handwritten Zip Code Recognition," Massachusetts Institute of Technology, Massachusetts, U.S.A, 1989.

[33] Goodfellow, Bengio and Courville, "Deep Learning Book p200," 2016. [Online]. Available: https://www.deeplearningbook.org/contents/mlp.html#pf33. [Accessed 11 June 2020].

[34] P. Sadowski, "Notes on Backpropagation," University of California Irvine Irvine, CA, U.S.A.

[35] L. Torrey and J. Shavlik, "Transfer Learning," Handbook of Research on Machine Learning Applications, IGI Global, University of Wisconsin, Madison WI, USA, 2009.

[36] L. v. G. C. W. J. W. A. Z. Mark Everingham, "PASCAL VOC Dataset," [Online]. Available: http://host.robots.ox.ac.uk/pascal/VOC/. [Accessed 11 June 2020].

[37] "ImageNet Dataset," Stanford University, Princeton University, [Online]. Available: http://www.image-net.org/. [Accessed 11 June 2020].

[38] T.-Y. Lin, G. Patterson, M. R. Ronchi, Y. Cui, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, L. Zitnick and P. Dollár. [Online]. Available: http://cocodataset.org/#home]. [Accessed 11 June 2020].

[39] D. Dodel, "FSOCO Dataset," [Online]. Available: https://ddavid.github.io/fsoco/. [Accessed 11 June 2020].

[40] J. Brownlee, "What is the Difference Between Test and Validation Datasets?," Machine Learning Mastery, 27 July 2017. [Online]. Available: https://machinelearningmastery.com/difference-test-validation-datasets/. [Accessed 11 June 2020].

[41] N. Srivastava, "Dropout," *Journal of Machine Learning,* 2014.

[42] J. Hui, "mAP (mean Average Precision) for Object Detection," Medium, [Online]. Available: https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173. [Accessed 11 June 2020].

[43] Y. LeCun, K. Kavukcuoglu and C. Farabet, "Convolutional Networks and Applications in Vision," IEEE, Courant Institute of Mathematical Sciences, New York University, U.S.A, 2010.

[44] "Supervised Learning: Pooling," Stanford University, [Online]. Available: http://ufldl.stanford.edu/tutorial/supervised/Pooling/. [Accessed 11 June 2020].

[45] "CS231n: Convolutional Neural Networks for Visual Recognition," Stanford, [Online]. Available: http://cs231n.stanford.edu/. [Accessed 11 June 2020].

[46] S. Grigorescu, T. Cocias, B. Trasnea, T. Cocias and G. Macesanu, "A Survey of Deep Learning Techniques for Autonomous Driving," arXiv:1910.07738v2, Transilvania University of Brasov. Brasov, Romania , 2020.

[47] K. He, :. X. Zhang, S. Ren and J. Sun, "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition," arXiv:1406.4729v4 [cs.CV] , Microsoft Research, Beijing, China, 2015.

[48] R. Girshick, J. Donahue, T. Darrell and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," arXiv:1311.2524v5 [cs.CV] , UC Berkeley, CA, U.S.A., 2014.

[49] J. Redmon and A. Farhadi, "YOLO: Real-Time Object Detection," [Online]. Available: https://pjreddie.com/darknet/yolo/. [Accessed 11 June 2020].

[50] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu and A. C. Berg, "SSD:SingleShotMultiBoxDetector," arXiv:1512.02325v5 [cs.CV], University of Michigan, Ann-Arbor, MI, U.S.A., 2016.

[51] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus and Y. LeCun, "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks," arXiv:1312.6229 [cs.CV], New York University , New York, U.S.A., 2014.

[52] H. Law and J. Deng, "CornerNet: Detecting Objects as Paired Keypoints," arXiv:1808.01244v2 [cs.CV] , Princeton University, Princeton, NJ, U.S.A., 2019.

[53] F. N. Iandola, SongHan, M. W. Moskewicz, K. Ashraf, W. J. Dally and K. Keutzer, "Squeezenet: Alexnet-level Accuracy with 50x Fewer Parameters and <0.5MB Model Size," arXiv:1602.07360v4 [cs.CV] , UC Berkeley & Stanford University, U.S.A., 2016.

[54] C. R. Qi, H. Su, K. Mo and L. J. Guibas, "PointNet: DeepLearningonPointSetsfor3DClassificationandSegmentation," arXiv:1612.00593v2 [cs.CV] , Stanford University, Pittsburgh, U.S.A., 2017.

[55] K. Shin, Y. P. Kwon and M. Tomizuka, "RoarNet: A Robust 3D Object Detection based on RegiOn Approximation Refinement," arXiv:1811.03818v1 [cs.CV] , UC Berkeley, CA 94720, U.S.A., 2018.

[56] "Medium," Medium Corporation, [Online]. Available: https://medium.com/. [Accessed 11 June 2020].

[57] "Amazon Web Services," Amazon, [Online]. Available: https://aws.amazon.com/es/. [Accessed 11 June 2020].

[58] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified,Real-Time Object Detection," arXiv:1506.02640v5 , University of Washington, Washingont, U.S.A, 2016.

[59] L. Weng, "Object Detection Part 4: Fast Detection Models," [Online]. Available: https://lilianweng.github.io/lil-log/2018/12/27/object-detection-part-4.html#yolo-you-only-look-once. [Accessed 11 June 2020].

[60] R. Huang, J. Pedoeem and C. Chen, "YOLO-LITE: A Real-Time Object Detection Algorithm Optimized for Non-GPU Computers," arXiv:1811.05588v1 , Georgia Institute of Technology Atlanta, U.S.A, 2018.

[61] M. J. Shafiee, B. Chyw, F. Li and A. Wong, "Fast YOLO: A Fast You Only Look Once System for Real-time Embedded Object Detection in Video," arXiv:1709.05943 [cs.CV], University of Waterloo, ON, Canada, 2017.

[62] J. Redmon and A. Farhadi, "YOLO9000: Better,Faster,Stronger," arXiv:1612.08242v1 [cs.CV], University of Washington, Washington, U.S.A., 2016.

[63] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," arXiv:1804.02767v1 [cs.CV], University of Washington, Washington, U.S.A., 2018.

[64] A. Kathuria, "What's new in YOLO v3?," Medium, 23 April 2018. [Online]. Available: https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b. [Accessed 11 June 2020].

[65] M. Bhobé, "MNIST — Digits Classification with Keras," Medium, 27 September 2018. [Online]. Available: https://medium.com/@mjbhobe/mnist-digits-classification-with-keras-ed6c2374bd0e. [Accessed 11 June 2020].

[66] A. Bochkovskiy, "YOLOv3 - Neural Networks for Object Detection (Windows and Linux version of Darknet)," GitHub, [Online]. Available: https://github.com/AlexeyAB/darknet. [Accessed 11 June 2020].

[67] "OIDv4_ToolKit," GitHub, [Online]. Available: https://github.com/theAIGuysCode/OIDv4_ToolKit. [Accessed 11 June 2020].

[68] "YoloGenerateTrainingFile," GitHub ,The AI Guy, [Online]. Available: https://github.com/theAIGuysCode/YoloGenerateTrainingFile. [Accessed 11 June 2020].

[69] M. Aryal, "Object Detection, Classification, and Tracking for Autonomous Vehicle - Master Thesis," Grand Valley State University, 2018.

[70] T. T. O. Takleh, N. A. Bakar, S. A. Rahman, R. Hamzah and Z. A. Aziz, "A Brief Survey on SLAM Methods in Autonomous Vehicle," International Journal of Engineering & Technology, Universiti Teknologi MARA (UiTM), Shah Alam, Malaysia, 2018.

[71] A. Boyali, S. Mita and V. John, "A Tutorial On Autonomous Vehicle Steering Controller Design, Simulation and Implementation," arXiv:1803.03758v1 [cs.RO] , Nagoya, Tenpaku Ward, Hisakata, Japan, 2018.

[72] N. H. Amer, H. Zamzuri, K. Hudha and Z. A. Kadir, "Modelling and Control Strategies in Path Tracking Controlfor Autonomous Ground Vehicles: A Review of State of the Art and Challenges," Universiti Pertahanan Nasional Malaysia, Kuala Lumpur, Malaysia, 2016.

[73] R. Rajamani, "Lateral Vehicle Dynamics," in *Vehicle Dynamics and Control* , University of Minnesota, U.S.A., 2006, pp. 15-31.

[74] A. De Luca and G. Oriolo, "Feedback Control of a Nonholonomic Car-like Robot," Università di Roma “La Sapienza”, Rome, Italy, 1997.

[75] R. C. Conlter, "Implementation of the Pure Pursuit Path 'hcking Algorithm," Camegie Mellon University Pittsburgh, Pennsylvania, U.S.A., 1992.

[76] M. M. e. a. "Junior: The Stanford Entry in the Urban Challenge," 2009.

[77] K. Bergman, "On Motion Planning Using Numerical Optimal Control," Linköping University, Linköping, Sweden , 2019.

[78] S. M. LaValle, "Chapter 1: Introduction, Chapter 5: Sampling-Based Motion Planning," in *Planning Algorithms*, Cambridge University Press, 2006, pp. 18, 237, 228.

[79] N. R. Kapania, "Trajectory Planning and Control for an Autonomous Race Vehicle - Doctor of Philosophy Thesis," Stanford University, Pittsburgh, U.S.A., 2016, p. 55.

[80] A. Anastassov, D. Jang and G. Giurgiu, "Driving Speed Profiles for Autonomous Vehicles," IEEE Intelligent Vehicles Symposium (IV) , Redondo Beach, CA, U.S.A., 2017.

[81] L. Claussmann, M. Revilloud, D. Gruyer and S. Glaser, "A Review of Motion Planning for Highway Autonomous Driving," IEEE Transactions on Intelligent Transportation Systems , IFSTTAR-LIVIC laboratory, Versailles, France, 2019.

[82] B. Trenwith, "Robotics [Penn State] 2.3.1.2 - Probabilistic Road Maps, 2.3.2 - Introduction to Rapidly Exploring Random Trees, 2.4.1 - Constructing Artificial Potential Fields," Penn State Univeristy, June 2018. [Online]. Available: https://drive.google.com/drive/folders/1S9FfOKmYFbj7EgHSOvu9FeS8Wn3nubOY. [Accessed 11 June 2020].

[83] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," The International Journal of Robotics Research, Massachusetts Institute of Technology, Cambridge, MA, USA, 2011.

[84] F. Zhang, J. Gonzales, S. E. L. F. Borrelli and K. Li, "Drift control for cornering maneuver of autonomous vehicles," Elsevier Ltd. , 2019.

[85] A. Abbadi and V. Přenosil, "Safe Path Planning Using Cell Decomposition Approximation," Masaryk University, Botanicka, Czech Republic , 2015.

[86] J. Moreau, P. Melchior, S. Victor, F. Aioun and F. Guillemard, "Path planning with fractional potential fields for autonomous vehicles," IFAC (International Federation of Automatic Control) Hosting by Elsevier Ltd. , 2017.

[87] T. M. Howard and A. Kelly, "Optimal Rough Terrain Trajectory Generation for Wheeled Mobile Robots," The International Journal of Robotics Research, Carnegie Mellon University Pittsburgh, PA, USA , 2007.

[88] C. Siedentop, R. Heinze, D. Kasper, G. Breuel and C. Stachniss, "Path-Planning for Autonomous Parking with Dubins Curves," University of Bonn, Germany, 2015.

[89] V. T. Minh, "Trajectory Generation for Autonomous Vehicles," Springer International Publishing Switzerland , Tallinn University of Technology, Tallinn, Estonia , 2014.

[90] A. Piazzi and C. G. L. Bianco, "Quintic G2-splines for trajectory planning of autonomous vehicles," Conference: Intelligent Vehicles Symposium, 2000. IV 2000. Proceedings of the IEEE, Universidad de Parma, Parma Italy, 2000.

[91] J.-w. Choi, R. Curry and G. Ekaim, "Path Planning Based on Bézier Curve for Autonomous Ground Vehicles," Advances in Electrical and Electronics Engineering - IAENG Special Edition of the World Congress on Engineering and Computer Science , University of California Santa Cruz Santa Cruz, U.S.A., 2008.

[92] "Linear Programming Basics," Massachusetts Institute of Technology, [Online]. Available: http://web.mit.edu/lpsolve/lpsolve-default/doc/LPBasics.htm. [Accessed 11 June 2020].

[93] "Nonlinear Programming," [Online]. Available: https://web.mit.edu/15.053/www/AMP-Chapter-13.pdf. [Accessed 11 June 2020].

[94] "Gurobi Optimization," Gurobi, [Online]. Available: https://www.gurobi.com/. [Accessed 11 June 2020].

[95] "CPLEX Optimizer," IBM, [Online]. Available: https://www.ibm.com/analytics/cplex-optimizer. [Accessed 11 June 2020].

[96] P. F. Lima, "Optimization-Based Motion Planning and Model Predictive Control for Autonomous Driving - Masters' Thesis," Stockholm, Sweden , 2018.

[97] S. S. Oyelere, "The Application of Model Predictive Control (MPC) to Fast Systems such as Autonomous Ground Vehicles (AGV)," IOSR Journal of Computer Engineering (IOSR-JCE) , University of Technology Yola, Nigeria, 2014.

[98] J. Chen, W. Zhan and M. Tomizuka, "Constrained Iterative LQR for On-Road Autonomous Driving Motion Planning," 2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC), University of California; Berkeley, U.S.A., 2017.

[99] C. Lum, "Introduction to Linear Quadratic Regulator (LQR) Control," YouTube, 3 December 2018. [Online]. Available: https://www.youtube.com/watch?v=wEevt2a4SKI. [Accessed 11 June 2020].

[100] L. Alonso, J. Perez-Oria, B. M. Al-Hadithi and A. Jimenez, "Self-Tuning PID Controller for Autonomous Car Tracking in Urban Traffic," 17th International Conference on System Theory, Control and Computing (ICSTCC), Sinaia, Romania, 2013.

[101] J. Kabzan, M. d. l. I. Valls, V. Reijgwart, H. F. C. Hendrikx, C. Ehmke, M. Prajapat, A. Bühler, N. Gosala, M. Gupta, R. Sivanesan, A. Dhall, E. Chisari, N. Karnchanachari and S. Brits, "AMZ Driverless: The Full Autonomous Racing System," arXiv:1905.05150 [cs.RO], ETH Zurich, Zurich, Switzerland, 2019.

[102] H. TIAN, J. NI and J. HU, "Autonomous Driving System Design for Formula Student Driverless," IEEE Intelligent Vehicles Symposium (IV), Beijing Institute of Technology, Beijing, China, 2002.

[103] M. Zeilinger, R. Hauk, M. Bader and A. Hofmann, "Design of an Autonomous Race Car for the Formula Student Driverless (FSD)," OAGM&ARW Joint Workshop , University of Applied Sciences, Vienna, Austria , 2017 .

[104] T. H. Drage, J. Kalinowski and T. Bräunl, "Development of an Autonomous Formula SAE Car with Laser Scanner and GPS," Proceedings of the 19th World Congress The International Federation of Automatic Control , The University of Western Australia, Perth, Australia, 2014.

[105] M. M. e. a. "Junior: The Stanford Entry in the Urban Challenge".

[106] N. H. Amer, H. Zamzuri, K. Hudha and Z. A. Kadir, "Modelling and Control Strategies in Path Tracking Control for Autonomous Ground Vehicles: A Review of State of the Art and Challenges," Journal of Intelligent and Robotic Systems, 2016.