



Faculty of Engineering - Cairo University Credit Hour System Programs

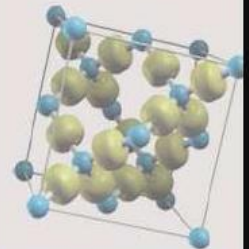
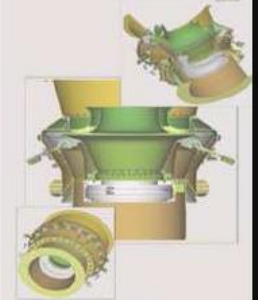
**Electronics and
Communication Engineering**

**Graduation Project Report
2020**

**High Performance Racing Vehicle
Driven by Formula1-Class AI Driver**

**Prepared by:
Abdelrahman Salah**

**Supervised by:
Prof. Dr. Hassan Mostafa**





Graduation Project-2

“High Performance Racing Vehicle Driven by Formula1 Class AI Driver”

Final Report

Submitted by:

Abdelrahman Salah

Supervised by:

Prof. Dr Hassan Mostafa




Cairo University

Credit Hour Systems Programs
Directory of B.Sc. Graduation Projects



Faculty of
Engineering

Electronics and Communication Engineering (CCE-E)

| | | |
|----------------------|---|--|
| Project Code | CCEN481-201* | |
| Project Title | High Performance Racing Vehicle Driven by Formula1-Class AI Driver | |
| Keywords | Autonomous Racing Vehicles, Computer vision, Artificial Intelligence, Localization, Mapping, Robotics, Deep Learning, Motion Control, Model Predictive control | |
| Students | Name: Abdelrahman Salah Amer | |
| |  <p>E-mail: abdelrahmansalahm10@gmail.com Phone: +20 1143447955 Address: 199 street, Degla, Maadi</p> | |
| Supervisor | Name: Dr. Hassan Mostafa | E-mail: hassanmostafahassan@gmail.com |
| | Signature: | Phone: +20 1200866660 |
| | | |

Project Summary

This project aims to create a formula1-class AI driver, developed to drive a high performance autonomous racing vehicle. The approach that was used combines **state-of-the-art techniques** from different fields of **robotics, computer vision and control**. Specifically, perception, estimation, and control are incorporated into one high performance autonomous car. Starting with the perception, a robust & reliable **software pipeline for a single monocular camera** was developed with the purpose of getting the most accurate results in estimating the 3D positions of track landmarks, and then fusing these results with a **stereo camera pipeline** to further improve perception accuracy and achieve **redundant-perception**. For localization and mapping, a simultaneous localization and mapping system that utilizes data from various sensing technologies in the vehicle, ensuring that the vehicle is accurately localized and the map is always updated in a real time Manner. Subsequently, a **path planning algorithm** was applied such that it chooses the mid-point path along the track length. Lastly, a **model predictive controller** was implemented to generate a trajectory that maximizes the vehicle's performance, minimizes lap time, and uses low-level hardware commands to control the vehicle.



Cairo University
Faculty of Engineering
Department of Computer Engineering

High Performance Racing Vehicle Driven by Formula1-Class AI Driver



A Graduation Project Report Submitted to
Faculty of Engineering, Cairo University
in Partial Fulfillment of the requirements of the degree
of
Bachelor of Science in Electronics and Communication Engineering.

Presented by
Abdelrahman Salah

Supervised by
Professor Hassan Mostafa

30/07/2020

All rights reserved. This report may not be reproduced in whole or in part, by photocopying or other means, without the permission of the authors/department.

Abstract

This project aims to create a formula1-class AI driver, developed to drive a high performance autonomous racing vehicle. This AI driver is created to replace human drivers, in order to eliminate human error, and drive the vehicle to its fullest allowing racing car manufacturers to test the full performance potential of the vehicle. The project gives a contribution to the racing motorsports industry which in turn serves for a development in the sense-plan-act cycle of the urban autonomous vehicles that is trying to reach full autonomy.

In order to fulfill the Society of Automation Engineers (SAE) level 4 autonomy, no driver attention must be required, even in emergency situations. Although a major part of autonomous driving on the public roads will happen in standard situations, a crucial aspect to reach full autonomy is the ability to operate a vehicle close to its limits of handling. Much like traditional motorsports, autonomous racing provides a platform to develop and validate new technologies under challenging conditions. Self-driving race cars provide a unique opportunity to test software required in autonomous transport, such as redundant perception, failure detection, and control in challenging conditions.

The approach that was used combines **state-of-the-art techniques** from different fields of **robotics, computer vision and control**. Specifically, perception, estimation, and control are incorporated into one high performance autonomous car. Starting with the perception, a robust & reliable **software pipeline for a single monocular camera** was developed with the purpose of getting the most accurate results in estimating the 3D positions of track landmarks, and then fusing these results with a **stereo camera pipeline** to further improve perception accuracy and achieve **redundant-perception**. For localization and mapping, a simultaneous localization and mapping system that utilizes data from various sensing technologies in the vehicle, ensuring that the vehicle is accurately localized and the map is always updated in a real time manner. Subsequently, a **path planning algorithm** was applied such that it chooses the mid-point path along the track length. Lastly, a **model predictive controller** was implemented to generate a trajectory that maximises the vehicle's performance, minimizes lap time, and uses low-level hardware commands to control the vehicle.

The project outputs a complete simulated intelligent model that is capable of exploring an unknown track, detecting landmarks, while simultaneously mapping the track and localizing itself, to finally generate a trajectory that enforces maximum performance along the track, and hopefully getting deployed in the previously developed electric racing vehicle by the team (Cairo University Racing Team) developing this project.

الملخص

هدف هذا المشروع هو إنشاء سائق ذو ذكاء اصطناعي من فئة الفورمولا 1، تم تطويره لقيادة سيارة سباق ذاتية التحكم عالية الأداء. بحيث يتم إنشاء سائق ذو ذكاء اصطناعي ليحل محل البشر، من أجل القضاء على الأخطاء البشرية، وقيادة السيارة إلى أقصى حد، حيث يسمح لمصنعي سيارات السباق باختبار إمكانات الأداء الكامل للسيارة. ويعد المشروع مساهمة في مجال رياضة السباق، والتي تساعد بدورها في تطوير دورة الاستشعار والتخطيط والفعل التي تتبعها السيارات ذاتية القيادة، والتي تسعى لتحقيق الاستقلال الكامل عن العنصر البشري.

من أجل تحقيق التحكم الذاتي من المستوى الرابع لجمعية مهندسي التشغيل الآلي (SAE)، لا يجب الانتباه أو الاعتماد على السائق، حتى في الحالات الحرجة والطارئة. وبالرغم من أن جزءًا كبيرًا من القيادة الذاتية على الطرق العامة سيحدث في الحالات الإعتيادية، فإن أحد الجوانب الحاسمة والمهمة للغاية للوصول إلى التحكم الذاتي الكامل هو القدرة على تشغيل السيارة قريبة من أقصى حدود أدائها.

وكما يشبه إلى حد كبير رياضة السيارات، فقد وفر سباق السيارات ذاتية القيادة منصة لتطوير والتحقق من التقنيات الجديدة في ظل ظروف قيادة صعبة. وتوفر سيارات السباق ذاتية القيادة فرصة فريدة من نوعها لاختبار البرامج المطلوبة في وسائل النقل ذاتية القيادة، مثل الإدراك أو المعرفة الزائدة، واكتشاف الأعطال، والتحكم في الظروف الصعبة.

ويجمع هذا النهج الذي تم استخدامه بين أحدث التقنيات من مجالات مختلفة من علم الروبوتات وأنظمة إبحار الحاسب الآلي والتحكم. على وجه التحديد، يتم دمج الإدراك والتقدير والتحكم في سيارة ذاتية القيادة عالية الأداء. بدءًا من الإدراك، تم تطوير خطوات تنفيذ البرمجيات بشكل قوي وموثوق به كاميرا أحادية العدسة بهدف الحصول على النتائج الأكثر دقة في تقدير المواضع ثلاثية الأبعاد للمسار، ثم دمج هذه النتائج مع خطوات تنفيذ البرمجيات كاميرا ثنائية العدسات لتحسين دقة الإدراك.

من أجل تحديد موقع السيارة وتخطيط المسار، يستخدم نظام تحديد موقع السيارة وتخطيط المسار بشكل متزامن البيانات من تقنيات الاستشعار المختلفة في السيارة، مما يضمن تحديد موقع السيارة بدقة وتحديث الخريطة دائمًا في الوقت الفعلي.

بعد ذلك، تم تطبيق خوارزمية تخطيط المسار بحيث يختار مسار نقطة المنتصف على طول المسار. أخيرًا، تم تنفيذ وحدة تحكم تنبؤية نموذجية لإنشاء مسار يزيد من أداء السيارة إلى أقصى حد ويقلل وقت الدورة ويستخدم الأوامر سهلة التعامل مع السيارة بغرض التحكم في السيارة.

ينتج المشروع نموذجًا ذكيًا كاملًا ومحاكيًا قادرًا على استكشاف مسار غير معروف، واكتشاف المعالم، وفي نفس الوقت رسم خريطة المسار وتحديد موقع السيارة، لإنشاء مسار يفرض الحد الأقصى من الأداء على طول المسار، ويتم نشره في سيارة السباق الكهربائية التي تم تطويرها سابقًا من قبل الفريق (فريق جامعة القاهرة لسباقات السيارات).

ACKNOWLEDGMENT

This work encloses our several trials in contribution towards science and research in order to learn and apply, studied and received knowledge during the past years studying at Bachelor level in Cairo University, Faculty of Engineering.

The success and final outcome of this project required a lot of guidance and assistance from many people and we were extremely privileged to have got this all along the completion of our project. All that we have done is only due to such supervision and assistance and we would not forget to thank them.

Thanks to Prof Dr. Hassan Mostafa for giving us all support and guidance which made us complete the project duly. We are extremely thankful to him for providing us with such nice support and guidance.

We also would like to thank Cairo University Racing Team (CURT) for providing us with information about vehicle models and providing us with parameters and calculations that make a clear image about how to build our model.

Finally we would like to thank Formula Students Artificial Intelligence (FSAI) Competition for providing us with a lot of priceless data and recorded sensors' data that helps us in our testing phase.

Table of Contents

| | |
|--|-----------|
| Abstract | 2 |
| الملخص | 3 |
| ACKNOWLEDGMENT | 4 |
| Table of Contents | 5 |
| List of Figures | 10 |
| List of Tables | 14 |
| List of Abbreviations | 15 |
| List of Symbols | 16 |
| Chapter 1: Introduction | 1 |
| Motivation and Justification | 2 |
| The Essential Question | 3 |
| Project Objectives and Problem Definition | 4 |
| Project Outcomes | 6 |
| Document Organization | 7 |
| Chapter 2: Market Feasibility Study | 8 |
| 2.1 Targeted Customers | 8 |
| 2.1.1 Automotive Companies (Valeo) | 8 |
| 2.1.2 Computer Vision & Robotics Companies | 9 |
| 2.2. Market Survey | 9 |
| 2.2.1. Roborace | 9 |
| 2.3 Business Case and Financial Analysis | 10 |
| 2.3.1 Business Case | 10 |
| 2.3.2 Financial Analysis | 10 |
| Chapter 3: Literature Survey | 11 |
| 3.1 Literature Review on Perception techniques | 12 |
| 3.1.1 Background information | 12 |
| 3.1.2 Comprehensive Study | 17 |
| 3.1.3 Implemented Approach | 18 |
| 3.2 Literature review on Object Detection and Recognition | |
| 3.2.1 Background on Object Detection and Recognition | 19 |
| 3.2.2 Comparative Study on Object Detection and Recognition | 29 |
| 3.2.3 Implemented Approach of Object Detection and Recognition | 30 |
| 3.3 Literature Review on 3D Object Localization and Pose Estimation From Single Images | 32 |

| | |
|---|-----------|
| 3.3.1 Background on 3D Object Localization and Pose Estimation From Single Images | 32 |
| 3.3.2 Comparative Study on 3D Object Localization and Pose Estimation From Single Images | 35 |
| 3.3.3 Implemented Approach on 3D Object Localization and Pose Estimation From Single Images | 37 |
| 3.4 Literature review on Mapping and Localization | 38 |
| 3.4.1 Background on Mapping and Localization | 38 |
| 3.4.2 Implemented approach for Mapping and Localization | 56 |
| 3.5 Literature Survey on Motion Control | 57 |
| 3.5.1 Background on Motion control | 57 |
| 3.5.2 Comparative Study on Motion Control | 58 |
| 3.5.3 Implemented Approach on motion control | 68 |
| 3.6 Literature Survey on the Deployment of the modules and communication between them | 70 |
| 3.6.1 Background Information | 71 |
| 3.6.2 Implemented approach | 74 |
| Chapter 4: System Design and Architecture | 75 |
| 4.1. Overview and Assumptions | 75 |
| 4.2. System Architecture | 77 |
| 4.2.1. Block Diagram | 80 |
| 4.3 2D Space Localization / Cone Position and Color Detection | 81 |
| 4.3.1. Functional Description | 81 |
| 4.3.2. Modular Decomposition | 82 |
| 4.3.3. Design Constraints | 86 |
| 4.4. Key Points extraction Residual Network | 88 |
| 4.4.1. Functional Description | 88 |
| 4.4.2. Modular Decomposition | 89 |
| 4.4.3. Design Constraints | 95 |
| 4.5 Stereo camera pipeline | 96 |
| 4.5.1 Functional description | 96 |
| 4.5.2 Modular decomposition | 97 |
| 4.5.3 Design constraints | 100 |
| 4.6 EKF Robot Localization and Mapping | 101 |
| 4.6.1 Overview and Assumptions | 101 |
| 4.6.2 EKF Robot localization Design Parameters | 103 |
| 4.6.3 Visual Odometry Modular Decomposition | 104 |
| 4.6.4 K-means-based Global Mapping Modular Decomposition | 106 |
| 4.6.5 K-means-based Global Mapping Design Constraints | 108 |
| 4.7 Model Predictive Control | 109 |
| 4.7.1 Overview and Assumptions | 109 |

| | |
|--|------------|
| 4.7.2 MPC Design parameters | 110 |
| 4.7.3 Block Diagram | 111 |
| 4.7.4 Functional Description | 112 |
| 4.8 Communication and Simulation module | 114 |
| 4.8.1 Functional Description | 114 |
| 4.8.3 Modular Decomposition | 115 |
| Chapter 5: System Testing and Verification | 118 |
| 5.1. Testing Setup | 119 |
| 5.2. Testing Plan and Strategy | 121 |
| 5.2.1 Testing the cone 2D localization module | 122 |
| 5.2.1 Testing the Cone Color Detection module | 128 |
| 5.2.3 7 Key Points Extraction Network | 131 |
| 5.2.4 Testing 3D space localization of cone: | 132 |
| 5.2.5 Testing EKF Robot Localization and Mapping Modules | 134 |
| 5.2.5 Testing Model Predictive Controller | 140 |
| 5.3. Testing Schedule | 143 |
| 5.4. Comparative Results to Previous Work | 144 |
| Chapter 6: Conclusions and Future Work | 145 |
| 6.1. Faced Challenges | 145 |
| 6.2. Gained Experience | 146 |
| 6.3. Conclusions | 147 |
| 6.4. Future Work | 148 |
| References | 149 |
| Appendix A: Development Platforms and Tools | 150 |
| A.1. Hardware Platforms | 150 |
| A.2. Software Tools | 151 |
| Appendix B: Use Cases | 153 |
| Appendix C: User Guide | 154 |
| Appendix E: Feasibility Study | 157 |

List of Figures

Figure 1.1: Autonomous Formula Race Car 2

Figure 1.2: Project Main Outcomes6

Figure 2.1: RoboRace cars9

Figure 3.1: Literature survey steps overview graph.....11

Figure 3.2: The LiDAR pipeline used to detect cones and estimate their color.....13

Figure 3.3: View of LiDAR.....14

Figure 3.4: An illustration of the cone reconstruction process..... 14

Figure 3.5: The intensity gradients for the pre-defined yellow and blue cones along with their point cloud returns.....15

Figure: 3.6 The stereo Setup.....16

Figure3.7: Camera vs LIDAR comparative result.....18

Figure 3.8: Monocular and Stereo camera ranges.....19

Figure 3.9 Different object detection approaches.....20

Figure 3.10 R-CNN object detection.....22

Figure 3.11 Spatial Pyramid Pooling Networks.....23

Figure 3.12 Fast RCNN.....24

Figure 3.13 Faster RCNN.....24

Figure 3.14 YOLO object detection.....25

Figure 3.15 SSD object detection.....26

Figure 3.16 SSD object detection layers.....27

Figure 3.17 RetinaNet object detection.....27

Figure 3.18 YOLOV3 object detection.....28

Figure 3.19 Comparative Study on Object Detection and Recognition.....29

Figure 3.20 Yellow, blue, and orange cones.....31

Figure 3.21 The voting process in Viewpoint Aware Object Detection Approach...33

Figure 3.22 Viewpoints and Key Points Approach.....34

Figure 3.23: Pascal VOC 2007 cars.....35

Figure 3.24: The developed DNN to extract cone key points.....37

Figure 3.25: Illustration of Kalman filters.....48

Figure 3.26: Simple example on applying Bayes Filter by using Kalman Filter Algorithm.....49

Figure3.27: The linearization process of the Extended Kalman Filter algorithm.....51

Figure 3.28: Localization example, finding the robot’s pose from landmarks location and sensor measurements.....55

Figure 3.29: Mapping example, finding landmarks location from robot’s pose and sensor measurements.....55

Figure 3.30: SLAM example, finding landmarks locations and robot’s pose at the same time using sensor measurements.....56

Figure 3.31: PID Controller.....59

Figure 3.32: Feedback Control.....60

Figure 3.33: Feedforward Control.....60

Figure 3.34: Feedforward and Feedback Control.....61

Figure 3.35:MPC Main Components.....63

Figure 3.36:Internal MPC configuration.....64

Figure 3.37: Tire Model.....67

Figure 3.38: Model Predictive Controller.....70

Figure4.1: The restricted environment (Track and cones).....75

Figure 4.2: Monocular and Stereo camera ranges.....76

Figure 4.3 : The track layout of a trackdrive discipline (FSG, 2018). Blue and yellow cones mark the track boundaries.....77

Figure4.4: Overall system block diagram.....80

Figure 4.5: Exemplary images from varying lighting and weather conditions81

Figure 4.6 Samples from the object detection dataset of the track cones.....83

Figure 4.7 CV-C Data Loader Pre-processing Stages.....84

Figure 4.8: The compounding benefits for the changes in YOLOv3.....85

Figure 4.9: Sample of landmarks which are used to create and update the track map.....87

Figure 4.10 Cone size vs focal length.....88

Figure 4.11: Cone’s 3D Model with the 7 key points.....89

Figure 4.12: Cone batches (left) and Cone batches with key points detected.....90

Figure4.13 :3D model of the cone and a representative sub-image patch.....91

Figure 4.14: Architecture of the “keypoint network”.....92

Figure4.15: An exemplary 80×80 cone patch with extracted “key points”.....93

Figure 4.16: ResNet architecture (left). Vectors used in geometric loss function (right).....93

Figure 4.17 Schematic illustrating matching of 2D-3D correspondence and estimation of transformation between the camera frame and the world frame.....95

Figure 4.18 Stereo camera pipeline’s block diagram.....97

Figure 4.19: Image rectification example.....98

Figure 4.20: 2D Cone bounding box detection(left) boundary box points (right).....99

Figure 4.21: Stereo camera depth map 2D image.....100

Figure 4.22: EKF Robot localization and mapping block diagram.....102

Figure 4.23: Vehicle Frame to Global Frame Transformation.....103

Figure 4.24: Stereo Rectification.....105

Figure 4.25: Stereo Feature Matching.....106

Figure 4.26: Stereo Camera Visual Odometry Pose Estimation.....107

Figure 4.27: The process of adding and updating cone estimates in the global map.....108

Figure 4.28: parameters of MPC.....111

Figure 4.29: Architecture of implemented MPC.....112

Figure 4.30: Vehicle’s path in blue car vs reference path in red color.....114

Figure 4.31: best predicted path close to reference taken over p time steps.....115

Figure 4.32 Robot operating system environment.....115

Figure 4.33: ROS nodes and published topics of the object detection module.....116

Figure 4.34: ROS nodes and published topics of the Key points extraction.....116

Figure 4.35: Rqt Graph of the ROS nodes and topics in mapping and localization algorithm connected to the node of ROS visualization rViz.....117

Figure 4.36 shows a sample from the rViz tool window118

Figure 5.1: The Main Goal of the project120

Figure 5.2: Testing Setup and Procedure overview.....123

Figure 5.3: Intersection-over-union - examined detection accuracy across three landmark sizes.....124

Figure 5.4: Testing the detection module in sunny climate condition.....125

Figure 5.5: Testing the detection module in high light climate condition.....126

Figure 5.6: Testing the detection module in cloudy climate condition.....127

Figure 5.7: Testing the detection module in low light climate condition.....128

Figure 5.8: Testing the detection module in rainy climate condition.....129

Figure 5.9: Color detection module testing metrics on the saturated epochs of training.....130

Figure 5.10: Testing the color detection with object detection.....131

Figure 5.11: Key points extraction network dataset.....132

Figure 5.12: Key points extraction network’s best loss result.....132

Figure 5.13: Key points extraction network’s results and points’ heatmap.....133

Figure 5.14: 3D Cone points, camera information, visualization of the rosbag in rViz.....134

Figure 5.15: Testing the 3D localization PnP algorithm with the detection and key points extraction modules.....135

Figure 5.16: Difference between GPS positioning (green) and EKF pose estimation position (red).....136

Figure 5.17: Visualizing the difference between Monocular Mapping pipeline and velodyne 3D scans |138

Figure 5.18: Visualizing the difference between our pipeline and the 3D scanned point-cloud. ||.....139

Figure 5.19: Visualizing the difference between our pipeline and the 3D scanned point-cloud. |||140

Figure 5.20: Simplest path at the center of the estimated track map from the mapping algorithm.....141

Figure 5.21: The actual track which produced by the MPC algorithm with the estimated map track.....142

Figure 5.22: State variables during the testing simulation of the MPC module....143

Figure 5.23: Control variables during testing simulation of the MPC module.....143

Figure 5.24 Distance errors over various distances of Monocular and Stereo Camera.....145

Figure A.1 Overview of the hardware sensors and actuators of the autonomous kit.....151

Figure A.2 Use case of the project.....154

List of Tables

| | |
|---|-----|
| Table 3.1: Wider variety of object detection benchmarks..... | 30 |
| Table 3.2: Pascal VOC 2007 cars. Average precision achieved by our detectors compared to a 2D baseline..... | 35 |
| Table 3.3: Viewpoint Estimation with Ground Truth box Performance..... | 36 |
| Table 3.4: Algorithm for Bayes Filter Update Rule..... | 43 |
| Table 3.5: Algorithm for Extended Kalman Filter..... | 52 |
| Table 3.6: Differentiating between Mapping, Localization, and SLAM..... | 55 |
| | |
| Table 5.1: Rosbags topics and input sensors' data..... | 121 |
| Table 5.2: The results of detection accuracy | 124 |
| Table 5.3: X, Y locations received from both EKF Localization module and GPS positions..... | 137 |
| Table 5.4: Gant chart for the testing schedule..... | 144 |
| Table E.1 Initial costs of the project..... | 157 |
| Table E.2 Cost estimation for future improvements..... | 158 |
| Tables E.3, E.4, and E.5 show the criterias we used to evaluate a potential risk... | 159 |
| Table E.6 shows our risk assessment and solutions. | 160 |
| Table E.7 shows the Gantt chart of the prototyping (implementation phase)..... | 178 |

List of Abbreviations

| | |
|--------|---|
| AI | Artificial Intelligence |
| CNN | Convolutional Neural Network |
| DNN | Deep Neural Network |
| KF | Kalman Filter |
| EKF | Extended Kalman Filter |
| FSAI | Formula Student Artificial Intelligence |
| FPS | Frame Per Second |
| FOV | Field Of View |
| GUI | Graphical User Interface |
| LIDAR | Light Detection And Ranging |
| ML | Machine Learning |
| MPC | Model Predictive Controller |
| MPCC | Model Predictive Contouring Control |
| MIMO | multi-inputs multi-outputs |
| PID | Proportional, Integral, Derivative |
| PnP | Perspective n points |
| ROS | Robot Operating System |
| ResNet | Residual Neural Network |
| SLAM | Simultaneous Localization and Mapping |
| SAE | Society of Automation Engineers |
| YOLO | You Only Look Once |
| GPU | Graphical Processing Unit |

List of Symbols

| | |
|--------------|-------------------------------------|
| $x(t)$ | System state vector |
| $u(t)$ | Control input vector |
| $y(t)$ | Output of the system. |
| (X, Y) | Position of vehicle |
| φ | Heading angle |
| (v_x, v_y) | Longitudinal and lateral velocities |
| r | Yaw rate. |
| δ | Steering angle |
| D | Driving command |

Contacts

Team Members

| Name | Email | Phone Number |
|------------------------|--|----------------|
| Abdelrahman Salah Amer | abdelrahmansalahm10@gmail.com | +20 1143447955 |

Supervisor

| Name | Email | Number |
|------------------------|--|----------------|
| Prof Dr Hassan Mostafa | hassanmostafahassan@gmail.com | +20 1200866660 |

This page is left intentionally empty

Chapter 1: Introduction

Autonomous racing has grown in popularity in the past years as a method of pushing the state-of-the-art for various autonomous robots. Self-driving vehicles improved safety, universal access, convenience, efficiency, and reduced costs compared to conventional vehicles. There is no driver attention must be required to fulfil even in emergency situations and under challenging weather conditions, a crucial aspect to reach full autonomy is the ability to operate a vehicle close to its limits of handling.

We will describe in this report our project and give an entire autonomous racing platform, covering all required software modules reaching from environment perception to vehicle dynamics control. Our project is divided into three main parts which is perception, simultaneous localization and mapping, and motion control. Starting with the perception pipeline, the developed system works using stereo camera and monocular camera. For localization and mapping, we utilize a graph based SLAM system, facilitating the detection and association of landmarks. Subsequently, we propose a custom approach to plan paths which takes the center of the track boundaries. Lastly, we present a control framework that directly minimizes lap time while obeying the vehicle's traction limits and track boundary constraints using a model predictive control controller.

This project aims to create a formula1-class AI driver developed to drive a high performance autonomous racing vehicle which is created to replace human drivers, in order to eliminate human error, and drive the vehicle to its fullest allowing racing car manufacturers to test the full performance potential of the vehicle. The project is divided into three main modules; Perception & Mapping, Trajectory Planning and Motion Control. The perception and mapping module is divided into two sub-modules; the perception sub-module which is concerned with detecting and localizing blue and yellow cones, this information is then fed to the mapping submodule, which constructs a full map of the track. The trajectory planning module is concerned with using the suitable algorithms to figure out a path around the track. The motion control module goal is to drive around the track as fast as possible while respecting the vehicle's model and track constraints.

1.1. Motivation and Justification

This project presents the algorithms and system architecture of a high performance autonomous racecar. The introduced vehicle is powered by a software stack designed for robustness, reliability, and extensibility. In order to autonomously race around a previously unknown track, the proposed project combines state-of-the-art techniques from different fields of robotics. Specifically, perception, estimation, and control are incorporated into one high-performance autonomous racecar.

This project aims to create a formula1-class AI driver, developed to drive a high performance autonomous racing vehicle. This AI driver is created to replace human drivers, in order to eliminate human error, and drive the vehicle to its fullest allowing racing car manufacturers to test the full performance potential of the vehicle.

The project gives a contribution to the racing motorsports industry which in turn serves for a development in the sense-plan-act cycle of the urban autonomous vehicles that is trying to reach full autonomy.

The need for High Performance Autonomous Vehicles

Higher levels of autonomy have the potential to reduce risky and dangerous driver behaviors. The greatest promise may be reducing the devastation of impaired driving, drugged driving, unbelted vehicle occupants, speeding and distraction. People with disabilities, like the blind, are capable of self-sufficiency, and highly automated vehicles can help them live the life they want. In a fully automated vehicle, all occupants could safely pursue more productive or entertaining activities, like responding to email or watching a movie. Autonomous vehicles maintain a safe



Figure1.1: Autonomous Formula Race Car

and consistent distance between vehicles, helping to reduce the number of stop and-go waves that produce road congestion. Thus, Self-driving vehicles promise significantly improved safety, universal access, convenience, efficiency, and reduced costs compared to conventional vehicles.

In order to fulfill the Society of Automation Engineers level 4 autonomy, no driver attention must be required, even in emergency situations and under challenging weather conditions. Although a major part of autonomous driving on the public roads will happen in standard situations, a crucial aspect to reach full autonomy is the ability to operate a vehicle close to its limits of handling, i.e. in avoidance maneuvers or in case of slippery surfaces.

In general, autonomous vehicles have lots of benefits as it helps in reducing traffic deaths as they eliminate human error, drop in harmful emissions as fewer accidents mean less traffic congestion which means drop in harmful emissions. It also will improve fuel economy and reduce travel time.

1.2. The Essential Question

According to a report by statista[], one in each ten cars will be autonomous by 2030. Billions have been spent on R&D in the field of self-driving vehicles, and car manufacturer giants, have all switched their interests into deploying self-driving cars. That's why, developing artificially intelligent agents capable of driving racing vehicles into unknown territory, with high performance and reliability; serves the goal of deploying autonomous vehicles into real roads.

We, in this project, are developing software and algorithms that create a driver capable of achieving high performance, reliability, and robustness. Which makes us competitive graduate engineers, indulged into international innovations and researches, ready to continue research and definitely learning in the field, effectively contributing to sustainable development in Egypt, and allowing us to serve individuals, society and the environment.

1.3. Project Objectives and Problem Definition

There are mainly two approaches that can be taken in order to develop an autonomous car. **An end-to-end approach** through imitation learning and learning-by demonstration and transfer learning. Such an approach firstly requires creating a test environment in order to engineer such a system is not only cumbersome and expensive, but also quite complex due to safety considerations. Secondly, building an end-to-end system requires several complex engineered systems, both in hardware and software, to work together - often such integrations are challenging, and such dependencies can greatly impede rapid development of individual systems (e.g., progress on autonomous software stack crucially depends upon bug-free hardware). Finally, much of the recent autonomous system modules depend upon the ability to collect a large amount of training data. Such data collection is not scalable in real-world due to both the complexity as well as resource requirements to carry out such training missions. These problems are alleviated by training and validating an autonomous stack in simulation. Such simulations allow to mitigate risks associated with safety, enable bypassing the need to access a closed off-road track and minimize the dependency on the development of the hardware stack. Most importantly, such simulations allow gathering data at scale, in a wide variety of conditions, which is not feasible in the real-world. The simulations can consist of a dynamic model of a custom car, a variety of racetrack scenes and synthesis of different weather conditions. Such simulations can indeed be very useful in building autonomous systems that can be deployed in the real-world.

The second approach combines state of the art techniques from different fields of robotics. Specifically, perception, estimation, and control are incorporated into one high performance autonomous car. Starting with the perception, systems can be developed to work using either a LiDAR, vision sensors or both. Next, the motion estimation subsystem fuses measurements from different sensors. For localization and mapping, a SLAM system shall be used, facilitating the detection and association of landmarks. Subsequently, an approach to plan paths can be taking the most likely track boundaries into account, given a map and/or on-board perception. Lastly, a control framework that directly minimizes lap time while obeying the vehicle's traction limits and track boundary constraints.

Project Objectives:

- Provide an AI Driver ready to race and adapt with a racing vehicle by driving it through the track as fast as possible to test and expose its dynamic power.
- Test the software required in autonomous transport in urban cities regarding redundant perception, failure detection, and control in challenging conditions.

To achieve these objectives, our project's goal is to create a formula1-class AI driver, developed to drive a high performance autonomous racing vehicle. WE create this AI driver to replace human drivers, in order to eliminate human error. This Project is divided into 3 main modules which we will talk about each one in full details in our document. these modules are:

1. Perception & mapping:

The goal of the perception pipeline is to efficiently provide accurate cone position and color estimates as well as their uncertainties in real-time with very high speed.

2. Trajectory planning:

Processing steps that are used to output the reference path of the vehicle should be always close to using cone colors and trigonometry which is the centerline of the track. If the vehicle kept to this path, it will be sure that the vehicle will never report an error by hitting a cone or getting out of its boundaries.

3. Motion Control:

The car now knows the track layout and can localize itself within the environment. Given this capability, we can race the car around a known track. Which brings us to our motion planning problem where the goal is to drive around the track as fast as possible.

1.4. Project Outcomes

The project outputs a complete simulated intelligent model that is capable of exploring an unknown track, detecting landmarks (track cones) of an unknown track with a full monocular perception pipeline with a stereo pipeline as a redundant perception, while simultaneously mapping the track and localizing itself using the K-means-based Global Mapping algorithm to build the map of the unknown track using the sensor fusion and Extended Kalman Filter (EKF), to finally generate a trajectory that enforces maximum performance along the track, and hopefully getting deployed in the previously developed electric racing vehicle by the team (Cairo University Racing Team) developing this project. The hardware platform of this project is an autonomous kit with embedded computer and a lot of sensors like Monocular camera, Stereo camera, IMU, and wheel odometry sensors.

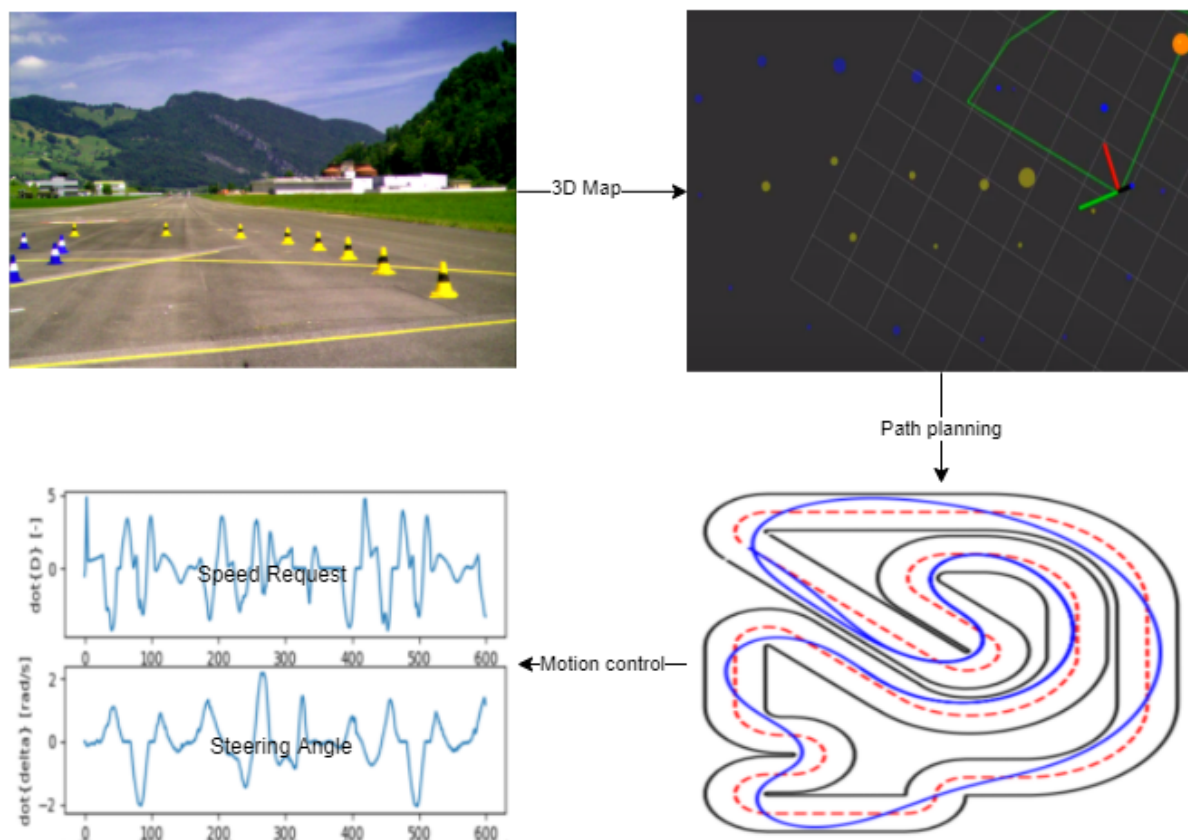


Figure 1.2 Project Main Outcomes

1.5. Document Organization

We will talk in the following chapters in full details about market visibility study such as targeted customers, market survey, and business case and financial analysis as in chapter two.

Then in chapter three we will talk deeply about literature survey. We will talk about each part of the three parts of our project which is perception and simultaneous localization and mapping and motion control, background about each part, implemented approach, and Comparative Study of Previous Work.

In chapter four we are going to discuss system design, architecture and algorithms which is the most important part of our report . We will talk in this chapter about overview and assumptions, functional description of architecture, modular decomposition which we will divide each module into submodules and talk in details about sensors and controllers and components that are used to build this architecture, design constraints and some block diagrams.

In chapter five we will explain testing setup, testing plan and strategy, test schedule, comparative results to previous work, faced challenges and future work. We also will explain how we test each module individually then integrate modules in addition to showing results of each module and in the overall system.

Chapter 2: Market Feasibility Study

The global autonomous/driverless car market was valued at USD 24.1 billion in 2019 and is expected to project a compound annual growth rate of 18.06%, during the forecast period, 2020-2025. Although Level 4 and Level 5 (as scaled by SAE) autonomous cars are unlikely to reach wide acceptance, by 2030, there will be a rapid growth for Level 2 and Level 3 autonomous cars, which have advanced driver assistance systems, like collision detection, lane departure warning, and adaptive cruise control. Fully autonomous cars are not going to reach a wide customer base, unless they are secure from cyber-attacks. If such concerns are addressed, the autonomous car market is estimated to reach USD 60 billion, by 2030. Major automaker companies, technology giants and specialist start-ups have invested more than USD 50 billion over the past five years, in order to develop autonomous vehicle (AV) technology, with 70% of the money coming from other than the automotive industry. At the same time, public authorities see that AVs offer huge potential economic and social benefits.

2.1 Targeted Customers

When it comes to marketing your business, it's all about defining your target. In this section, targeted customers or better known as targeted markets will be reviewed. Based on market needs and interests, two main domains of customers were focused on.

2.1.1 Automotive Companies (Valeo)

Due to the fact of self-driving cars being the future, most if not all of automotive companies around the globe are shifting their interests into the development of autonomous vehicles. Companies now have dedicated teams that are working day-in & day-out developing the needed software and hardware that enables the deployment of autonomous cars.

On a local level, **Valeo** is a main targeted company. Valeo, recently, added to their interests the development of self driving car software. They are currently working on the development of more robust and real time algorithms to be utilized into self driving vehicles.

Our project outputs software that will be deployed on racing vehicles, which makes it very robust, real time, and reliable. Additionally, our project paves the way for more research and additional development of both software and hardware needed in autonomous vehicles. We believe that our project can help Valeo accomplish their goals in the field.

2.1.2 Computer Vision & Robotics Companies

There are numerous companies in Egypt that are using deep learning techniques, image processing and computer vision algorithms in a very wide range of applications, like; real-time monitoring of manufacturing assembly lines, retails in monitoring malls traffic activities, and robots.

In our project, we were able to create a perception pipeline for the vehicle, that utilizes state-of-the-art computer vision and deep learning techniques, and works in a real-time manner. Through tweaking such a pipeline software, one can add a variety of features that enables such software to be used by such companies in a wide variety of applications.

These companies are; Avidbeam, Sypron, Avelabs, Affectiva, The D. GmbH and InnoVision Industries.

2.2. Market Survey

2.2.1. Roborace

Roborace is the world's first racing series for humans and artificial intelligence. It was created to accelerate the development of autonomous software by pushing the technology to its limits in a range of controlled environments. Roborace is a platform for the development of autonomous technology in an extreme environment and educates the public about the benefits and safety of these technologies when they make it onto the roads at scale.



Figure: 2.1 Roborace cars

Roborace are developing autonomous racing electric vehicles with the target of creating autonomous cars race competitions, where teams can excessively test their software and compete against each other in a real world environment. It gives organizations developing driverless technologies an extreme yet safe environment to test their software and hardware pushing them to the limits of their capabilities. Robocar's Nvidia Drive PX2 GPU "brain" is capable of up to 24 trillion A.I. operations per second.

Despite that Roborace can be seen as a competitor, we believe that it can be better seen as an opportunity. Using our software and hardware experience in developing a driverless racing vehicle, we can be among the teams participating in a huge autonomous racing competition.

2.3 Business Case and Financial Analysis

A business that develops reliable software for autonomous vehicles, and builds the base of research in the field, would be the first of its kind in Egypt. We expect such a business to boom in the entire middle east and the Arabic world due to it being the first of its kind in the area. Such a business creates huge chances for having autonomous software developed entirely in Egypt and by Egyptian engineers, only if it was studied and applied correctly.

2.3.1 Business Case

Based on our market survey, over the next five years we expect that we can get a collaboration deal with automotive company **Valeo**. We can work closely with them in their research regarding self-driving vehicles, directly developing software and algorithms in the field.

Regarding computer vision companies, we expect developing at least five applications in the field. Applications that utilize computer vision and deep learning techniques to provide solutions to modern problems.

2.3.2 Financial Analysis

- a. The Capex (Capital Expenditure):
 1. Two workstation machines:
 - 1.1. 2x GPU GTX 2080 TI (48000 EGP)
 - 1.2. 2x AMD Ryzen™ Threadripper™ 3970X (64000 EGP)
 - 1.3. 2x 32GP RAM Corsair Vengeance RGB Pro DDR4 (4600 EGP)
 - 1.4. 2x Case and other main peripherals (16000 EGP)
 2. Worksight, mainly an apartment in a reputable place that is around 120 meters squared (800,000 EGP)
 3. Office Equipment (chairs, disks, ...etc.) (20,000 EGP)
- b. The Opex (Operational Expenses):
 1. Monthly Salaries:
 - 1.1 Four Team Leaders: (12000 x 4 = 48000 EGP).
 - 1.2 Twelve software developers: (8000 x 12 = 96000 EGP).
 - 1.3 Three marketing & social media moderators (5000 x 3 = 15000 EGP).
 - 1.4 Two cleaners (2000 x 2 = 4000 EGP).
 2. Facility costs:
 - 2.1 Electricity (1000 EGP).
 - 2.2 Water (150 EGP).
 - 2.3 Maintenance costs (2500 EGP)
 3. Online marketing and sales (500 EGP).

Chapter 3: Literature Survey

Autonomous racing presents a unique opportunity to test commonly-applied, safety-critical perception, and autonomy algorithms in extreme situations at the limit of vehicle handling and provides the opportunity to test safety-critical perception pipelines at their limit. This section describes a literature background for each module in the autonomous racing platform, covering all required software modules reaching from environment perception to mapping and control. The following graph shows the track of our literature survey process to develop a driverless race car.

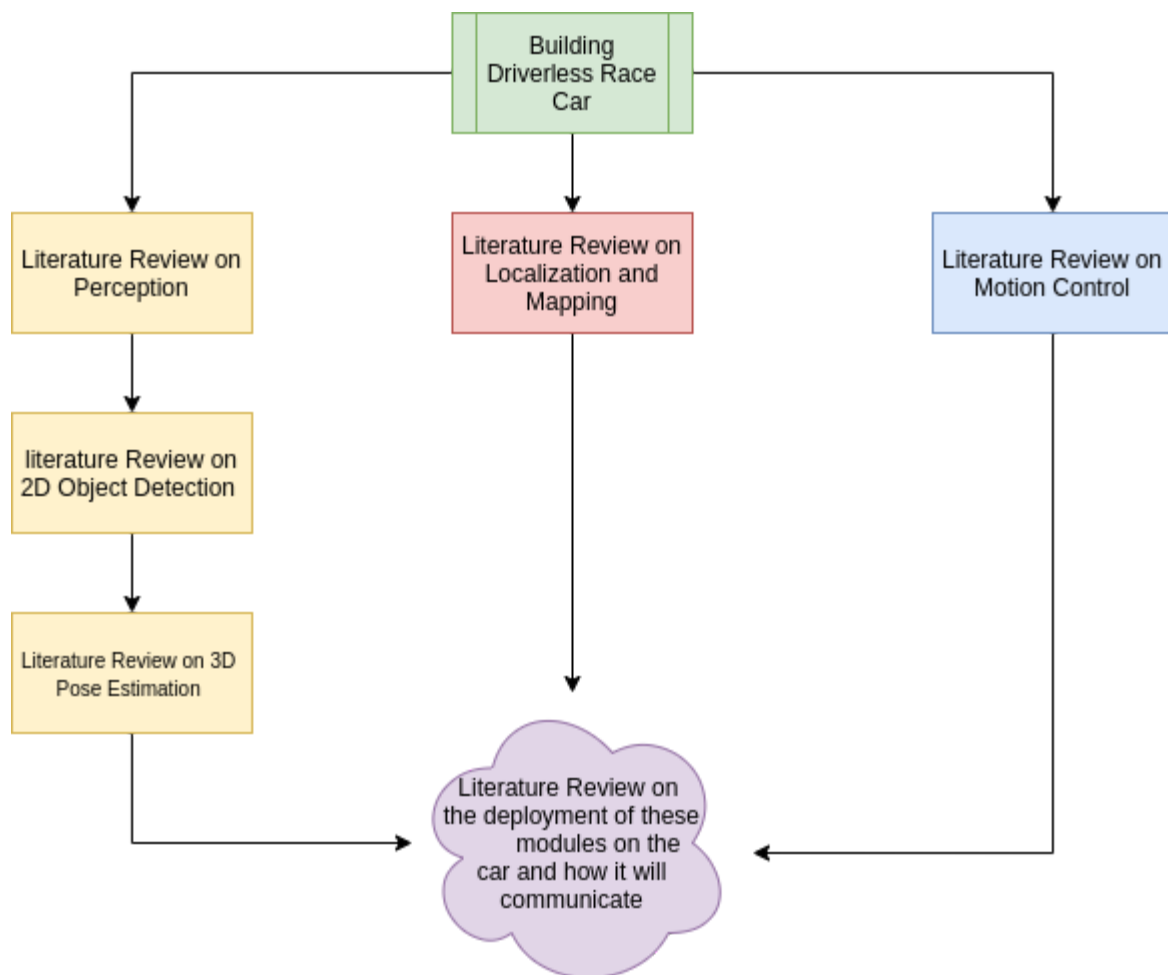


Figure 3.1: Literature survey steps overview graph

3.1 Literature Review on Perception techniques

For autonomous vehicles, perception is a crucial task to make decisions, plan, and operate in real-world environments, by means of numerous functionalities and operations from occupancy grid mapping to object detection. Nowadays, most perception systems use machine learning (ML) techniques, ranging from classical to deep-learning approaches. Machine learning for robotic perception can be in the form of unsupervised learning, or supervised classifiers using handcrafted features, or deep-learning neural networks (e.g., convolutional neural network (CNN)), or even a combination of multiple methods.

Regardless of the ML approach considered, data from sensor(s) are the key ingredient in autonomous vehicles perception. Data can come from a single or multiple sensors, usually mounted on board the vehicle, but can also come from the infrastructure or from another vehicle (e.g., cameras mounted on UAVs flying nearby). In multiple detection sensors more than one sensor can be used in the perception pipeline according to the design, and the data from these sensors can be fused in a certain way to use it to localize and map the environment around the vehicle. This section describes the different types of perception techniques and approaches and describes the chosen approach in our driverless race car.

3.1.1 Background information

The main goal of the perception pipeline in the formula racing car is to estimate the position and the color of the track cones in an accurate and fast way. These estimations and detections are used in the mapping module to build the map of the track. There are different ways to perceive the track environment:

1) LIDAR-based Sensing

LIDAR is a surveying technology that measures distance by illuminating a target with a laser light. LIDAR is an acronym of Light Detection And Ranging, (sometimes Light Imaging, Detection, And Ranging) and was originally created as a portmanteau of “light” and “radar.”

In order to perceive the track cones' position and color in this technique which based on LIDAR the perception process and the information from the LiDAR is capitalized in two ways:

- 1-The 3D information in the point cloud is used to detect cones on the track and find their position with respect to the car.
- 2-The intensity data is used to differentiate between the various colored cones.

In three main phases which are pre-processing, cone detection, and color estimation, the cones which draw the track are detected and classified using the LiDAR pipeline shown in the following figure.

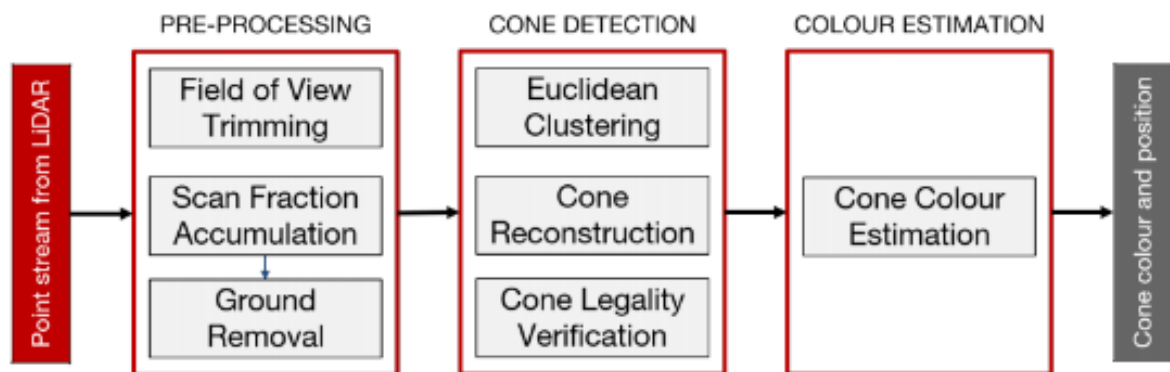
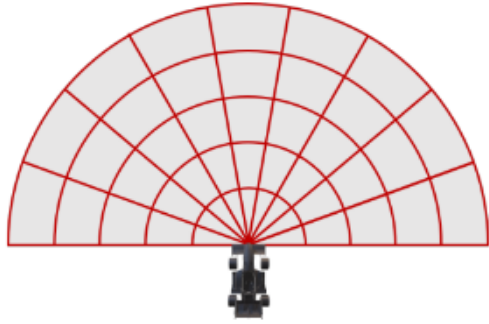


Figure3.2: The LiDAR pipeline used to detect cones and estimate their color

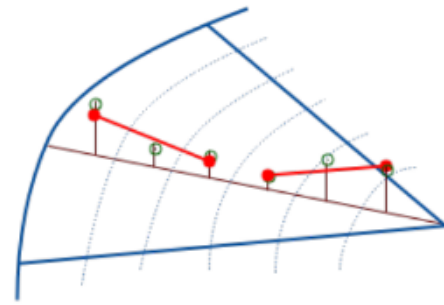
a) Pre-Processing

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

An adaptive ground removal algorithm (Himmelsbach et al., 2010) that adapts to changes in the inclination of the ground using a regression based approach is used to estimate the ground plane and distinguish between the ground and non-ground points, after which the ground points are discarded (Gosala et al., 2018). The ground removal algorithm works by dividing the FoV of the LiDAR into angular segments and splitting each segment into radial bins Figure 3.3-a. A line is then regressed through the lowermost points of all bins in a segment. All the points that are within a threshold distance to this line are classified as ground points and are removed see figure 3.3-b.



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



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

Figure 3.3: View of LiDAR

b) Cone Detection

The aforementioned ground removal algorithm removes a substantial amount of cone points in addition to those of the ground. This significantly reduces the already small number of return points that can be used to detect and identify cones. This is addressed by first clustering the point cloud after ground removal using

Euclidean-distance based clustering algorithm and then reconstructing a cylindrical area around each cluster center using points from the point cloud before ground removal. This recovers most of the falsely removed

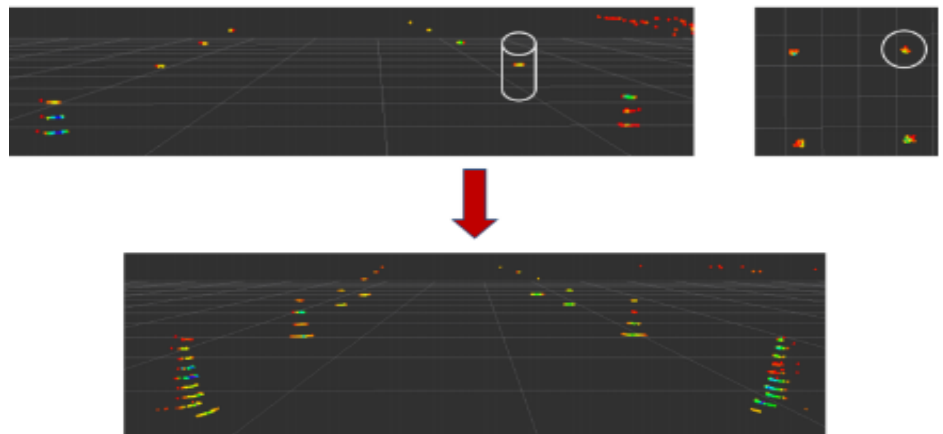


Figure 3.4: An illustration of the cone reconstruction process

points improving cone detection and color estimation. The reconstructed clusters are then passed through a rule-based filter that checks whether the number of points in that cluster is in accordance with the expected number of points in a cone.

In Figure 3.4, The top-left image shows the LiDAR point cloud after executing the adaptive ground removal algorithm. One can note the sparsity of the point cloud and the dearth of points in a cone. The cones are reconstructed by retrieving a cylindrical

area around each cluster center (top-right) resulting in the bottom image wherein the presence of cones is more evident.

c) Cone Pattern Estimation

According to the rule book (FSG, 2018) [1], a yellow cone has a yellow-black-yellow pattern whereas a blue cone has a blue-white-blue pattern which results in differing LiDAR intensity pattern as one moves along the vertical axis of the cone as shown in Figure 3.5. When reading the intensity values along the vertical axis, one denotes high-low-high and low-high-low patterns for the yellow and blue cones respectively. These differing intensity patterns are used to differentiate between yellow and blue cones.



Figure 3.5: The intensity gradients for the pre-defined yellow and blue cones along with their point cloud returns.

2) Camera based Sensing

Cameras are a crucial exteroceptive sensor for self-driving cars as they are low-cost and small, provide appearance information about the environment, and work in various weather conditions. They can be used for multiple purposes such as visual navigation and obstacle detection.

a) Stereo Camera

The stereo vision system is one of the popular computer vision techniques. The idea here is to use the parallax error to our advantage. A single scene is recorded from two different viewing angles, and depth is estimated from the measure of parallax error. This technique is more than a century old and has proven useful in many applications. This field has made a lot of researchers and mathematicians to devise novel algorithms for the accurate output of the stereo systems. This system is particularly useful in the field of robotics. It provides them with the 3D understanding of the scene by giving them estimated object depths. Stereo depth estimation can be made a lot more

efficient than the current techniques. The idea revolves around the fact that stereo depth estimation is not necessary for all the pixels of the image. This fact opens room for more complex and accurate depth estimation techniques for the fewer regions of interest in the image scene.

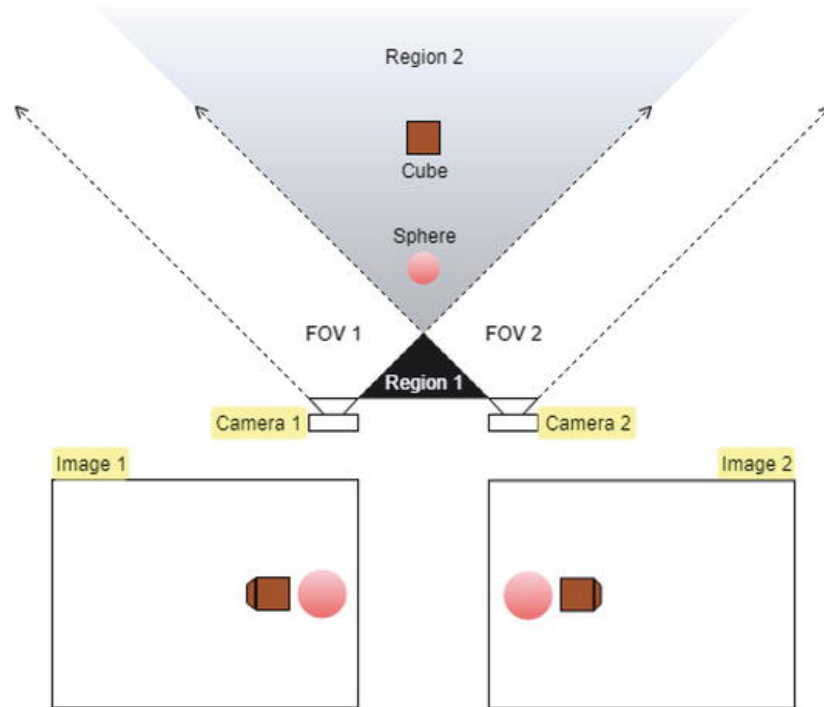


Figure: 3.6 The stereo Setup

This instrument was first described to us in 1838 by Charles Whitestone to view relief pictures. He called it the stereoscope [2]. A lot of other inventors and visionaries later used this concept to develop their versions of stereoscopes. It even led to the establishment of the London Stereoscopic Company in 1854. The concept of depth estimation using multiple views was used even for the estimation of the distance of the far away astronomical objects in the early times. The depth is also directly proportional to the distance between the two cameras of the stereo vision system, also called the baseline. Hence the estimation of such vast distances demanded us to use the longest possible baseline length that we could use. So the data was recorded from Earth being on either side of the sun, making the baseline length to be the same as the diameter of the Earth's orbit around the sun, and then the depth of the astronomical objects is measured. This method was called the stellar parallax or trigonometric parallax .

Using the concept of stereo vision it is shown that it is possible to use the stereo vision in the perception pipeline of the driverless racing vehicle and in this case the region of interest that we need to know its depth is the track cone and then to estimate the cone's position in the 3D by using object detection of the cones to extract the bounding boxes (Region of interest) and then using the feature matching and triangulation between the left and right frames the cone position in 3D will be estimated.

b) Mono camera

Unlike the stereo camera setup, mono camera setup can be used in the perception pipeline by using a single camera we can localize and estimate the 3D position of the object (track cones in our case). The literature survey of this problem is described in detail in Section 3.3.

3.1.2 Comprehensive Study

LIDAR systems are currently large and expensive systems which must be mounted outside of vehicles. The system Google uses is in the range of 80 kg and \$70,000, for example, and must be mounted on top of the vehicle with unobstructed sight lines. Due to their current limitations, the systems are not useful for detecting anything near the car. Current implementations have improved range substantially from early 30 meter ranges up to 150 to 200 meter ranges, with increases in resolution as well. At present, production systems with higher range and resolution continue to be expensive. LIDAR works well in all light conditions, but starts failing with increases in snow, fog, rain, and dust particles in the air due to its use of light spectrum wavelengths. LIDAR cannot detect colour or contrast, and cannot provide optical character recognition capabilities. Narrow-beam LIDAR has been used for 20 years, but current-generation LIDAR used on autonomous cars is less effective for real-time speed monitoring.

Camera image recognition systems have become very cheap, small, and high-resolution in recent years. They are less useful for very close proximity assessment than they are for further distances. Their colour, contrast, and optical-character recognition capabilities give them a full new capability set entirely missing from all other sensors. They have the best range of any sensor but that's in good light conditions. Their range and performance degrades as light levels dim, starting to depend — as human eyes do — on the light from headlights of the car. In very bright conditions, it is apparently possible for some implementations to not identify light objects against bright skies, which was reportedly a factor in the May 2016 Tesla Autopilot-related fatality in Florida. Digital signal processing makes it

possible to determine speed, but not at the level of accuracy of radar or LIDAR systems.

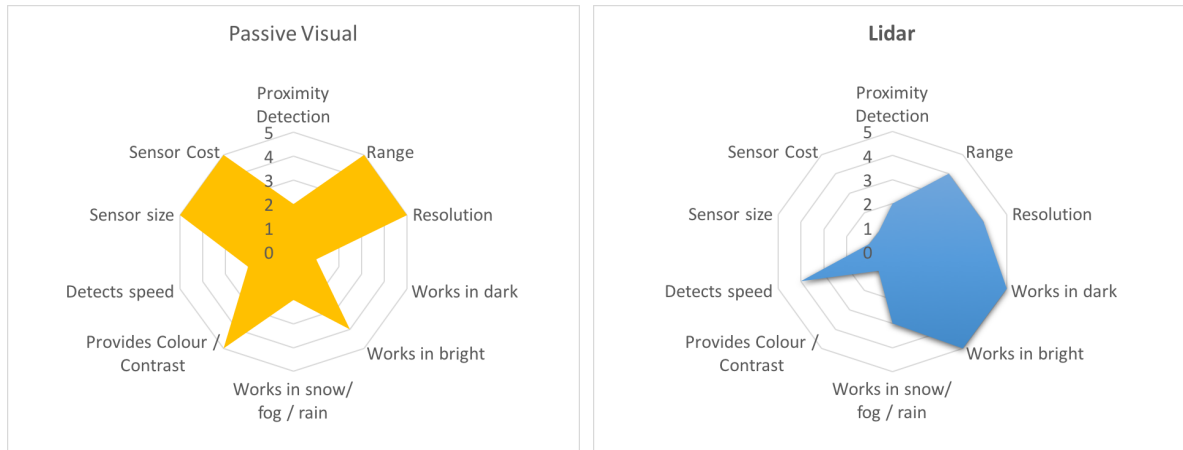


Figure 3.7: Camera vs LIDAR comparative study

3.1.3 Implemented Approach

Due to the logistics and financial situation of bringing the LIDAR as mentioned in Section 3.3.2, the LIDAR is excluded from the perception pipeline. For the perception pipeline two-camera architecture were deployed with a **stereo** camera used for long-range detections and a **monocular** camera for short-range detections. As shown in Figure 3.8. Because in our case we need a high performance perception pipeline the idea of redundant perception was used, and by fusing the data from the two camera setups we can build the track map in an accurate and fast way.

The rationale for using the monocular camera for short-range rather than long-range detections is that for a reasonable mounting height, a landmark's 3D location on a relatively flat surface is a much stronger function of pixel space location for short-range objects than long-range objects. This relieves some of the challenges for estimating landmark pose from a monocular camera. On the other hand, however, estimating 3D pose of an object using a single measurement, i.e. a single image from a monocular camera is an ill-posed problem. This is primarily due to ambiguity in the scale of the scene arising from limited information of the surroundings. This ill-posed problem of extracting pose information can be solved if a priori information about the 3D object in the scene is available. The 3D priors about an object, in addition to 2D information obtained from an image can be together leveraged to extract 3D pose of this object captured in any arbitrary image of the scene. On a real-time system, such as an autonomous race-car, it becomes even more crucial to detect and estimate multiple object positions extremely efficiently, with a little latency

and data overhead (in terms of transport and processing) as possible. The implementation of the perception pipeline is briefly mentioned in chapter 4.

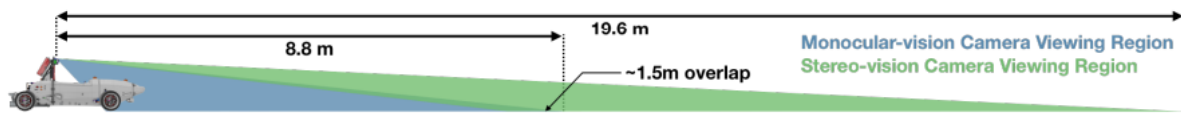


Figure 3.8: Monocular and Stereo camera ranges

3.2 Literature review on Object Detection and Recognition

Object detection is a crucial task for autonomous driving. In addition to requiring high accuracy to ensure safety, object detection for autonomous driving also requires real time inference speed to guarantee prompt vehicle control, as well as small model size and energy efficiency to enable embedded system deployment.

3.2.1 Background on Object Detection and Recognition

Evaluating metrics

Evaluation metrics used in classification are not enough when working with detection. In addition to evaluating how good a detector is at classifying an object, we need a way to quantify how well the bounding boxes fits the objects.

AP (Average precision)

It is a popular metric in measuring the accuracy of object detectors like Faster R-CNN, SSD, etc. Average precision computes the average precision value for recall value over 0 to 1. Precision measures how accurate your predictions are. i.e. the percentage of your predictions are correct.

$$\text{Precision} = \text{TruePositives} / (\text{TruePositives} + \text{FalsePositives})$$

Recall measures how good you find all the positives. For example, we can find 80% of the possible positive cases in our top K predictions.

$$\text{Recall} = \text{TruePositives} / (\text{TruePositives} + \text{FalseNegatives})$$

IoU (Intersection over union)

IoU measures the overlap between 2 boundaries. We use that to measure how much our predicted boundary overlaps with the ground truth (the real object boundary). In some datasets, we predefine an IoU threshold (say 0.5) in classifying whether the prediction is a true positive or a false positive.

$$\text{IoU}(\text{box1}, \text{box2}) = (\text{area}(\text{box1} \cap \text{box2}) / \text{area}(\text{box1} \cup \text{box2}))$$

Mean Average Precision (mAP)

It is an evaluation metric often used to compare different detection systems. It is calculated by taking the average of the maximum precisions at 11 recall levels evenly spaced between 0 and 1. Predictions are required to have an IoU > 0.5. All predictions with IoU < 0.5 are treated as a wrong prediction. The mAP is the average of the AP for each class.

Frames Per Second (FPS)

It is the number of frames the object detection module can infer per one second.

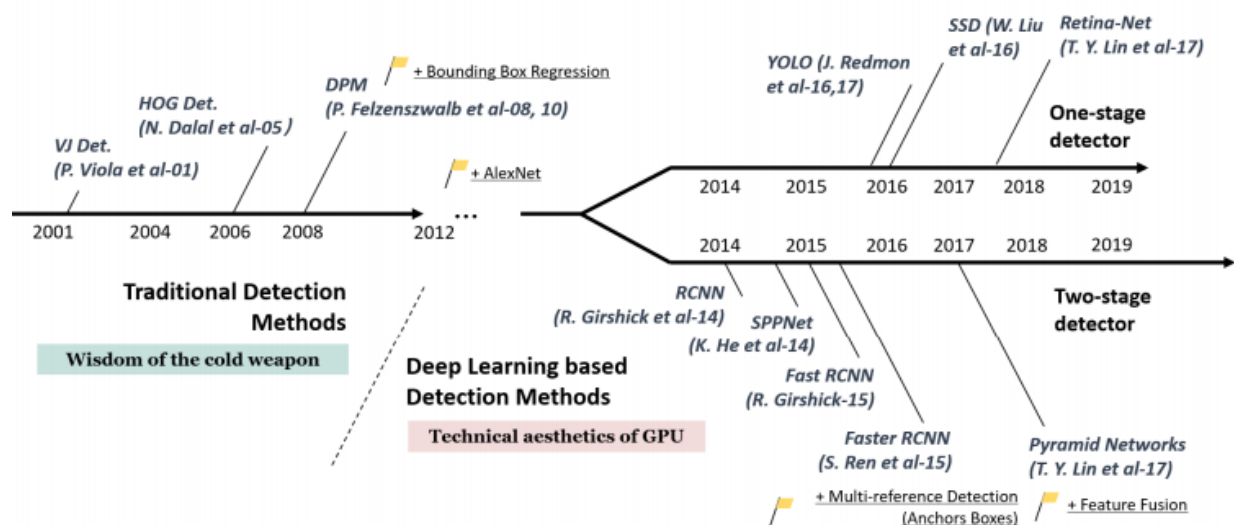


Figure 3.9 Different object detection approaches

Deformable Part-based Model (DPM)

It was a revolution of object detectors at its time. DPM was originally proposed by P. Felzenszwalb in 2008, and then a variety of improvements have been made by R. Girshick. The DPM is based on the “divide and conquer” method, where the training can be simply considered as the learning of a proper way of decomposing an object, and the inference can be considered as an ensemble of detections on different object parts. For example, detecting a person can be looked as detecting his head, two arms and two legs. R. Girshick has further extended the star-model to the “mixture models” to deal with the objects in the real world under more significant variations. A typical DPM detector consists of a root-filter and a number of part-filters. Instead of manually specifying the configurations of the part filters (e.g., size and location), a weakly supervised learning method is developed in DPM where all configurations of part filters can be learned automatically as latent variables. R. Girshick has further formulated this process as a special case of Multi-Instance learning, and some other important techniques such as “hard negative mining”, “bounding box regression”, and “context priming” are also applied for improving detection accuracy. To speed up the detection, he developed a technique for “compiling” detection models into a much faster one that implements a cascade architecture, which has achieved over 10 times acceleration without sacrificing any accuracy. Although today’s object detectors have far surpassed DPM in terms of the detection accuracy, many of them are still deeply influenced by its valuable insights.

RCNN

It starts with the extraction of a set of object proposals (object candidate boxes) by selective search, which is explained below. Then each proposal is rescaled to a fixed size image and fed into a CNN model trained on ImageNet (say, AlexNet) to extract features. Finally, linear SVM classifiers are used to predict the presence of an object within each region and to recognize object categories. RCNN yields a significant performance boost on VOC07, with a large improvement of mean Average Precision (mAP) from 33.7% (DPM-v5) to 58.5%. Although RCNN has made great progress, its drawbacks are obvious: the redundant feature computations on a large number of overlapped proposals (over 2000 boxes from one image) leads to an extremely slow detection speed (14s per image with GPU). Later in the same year, SPPNet was proposed and has overcome this problem.

Selective search

1. Generate initial sub-segmentation, we generate many candidate regions.
2. Use greedy algorithms to recursively combine similar regions into larger ones.
3. Use the generated regions to produce the final candidate region proposals.

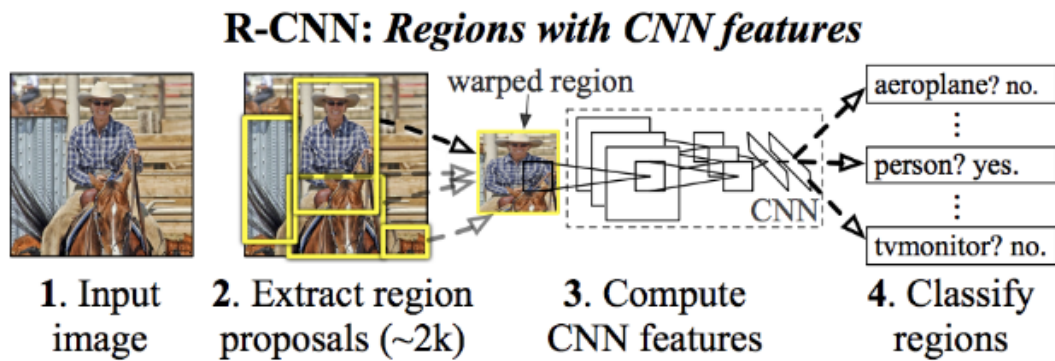


Figure 3.10 R-CNN object detection

Spatial Pyramid Pooling Networks (SPPNet)

The main contribution of SPPNet is the introduction of a Spatial Pyramid Pooling (SPP) layer, which enables a CNN to generate a fixed-length representation regardless of the size of image/region of interest without resizing it. The technique performs pooling (ex: Max pooling) on the last convolution layer (either convolution or sub sampling) and produces a $N \times B$ dimensional vector (where N =Number of filters in the convolution layer, B = Number of Bins). The vector is in turn fed to the FC layer. The number of bins is a constant value. Therefore, the vector dimension remains constant irrespective of the input image size. When using SPPNet for object detection, the feature maps can be computed from the entire image only once, and then fixed length representations of arbitrary regions can be generated for training the detectors, which avoids repeatedly computing the convolutional features. SPPNet is more than 20 times faster than R-CNN without sacrificing any detection accuracy. Although SPPNet has effectively improved the detection speed, there are still in a 2D single frame and assigns it to a class.

Now, due to the advancements in some drawbacks: first, the training is still multi-stage, second, SPPNet only fine-tunes its fully connected layers while simply ignores all previous layers.

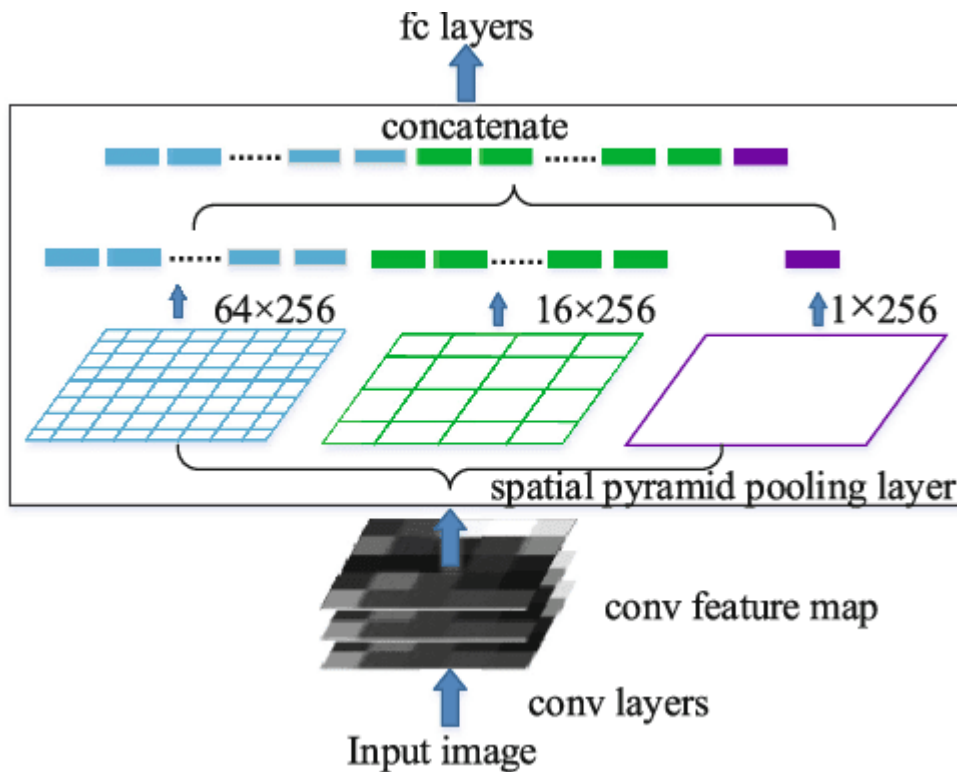


Figure 3.11 Spatial Pyramid Pooling Networks

Fast RCNN

In 2015, R. Girshick proposed Fast RCNN detector, which is a further improvement of R-CNN and SPPNet. The approach is similar to the R-CNN algorithm. But, instead of feeding the region proposals to the CNN, we feed the input image to the CNN to generate a convolutional feature map. From the convolutional feature map, we identify the region of proposals and warp them into squares and by using a RoI pooling layer we reshape them into a fixed size so that it can be fed into a fully connected layer. From the RoI feature vector, we use a softmax layer to predict the class of the proposed region and also the offset values for the bounding box. Fast RCNN enables us to simultaneously train a detector and a bounding box regressor under the same network configurations. On the VOC07 dataset, Fast RCNN increased the mAP from 58.5% (RCNN) to 70.0% while with a detection speed over 200 times faster than R-CNN. Although Fast-RCNN successfully integrates the advantages of R-CNN and SPPNet, its detection speed is still limited by the proposal detection.

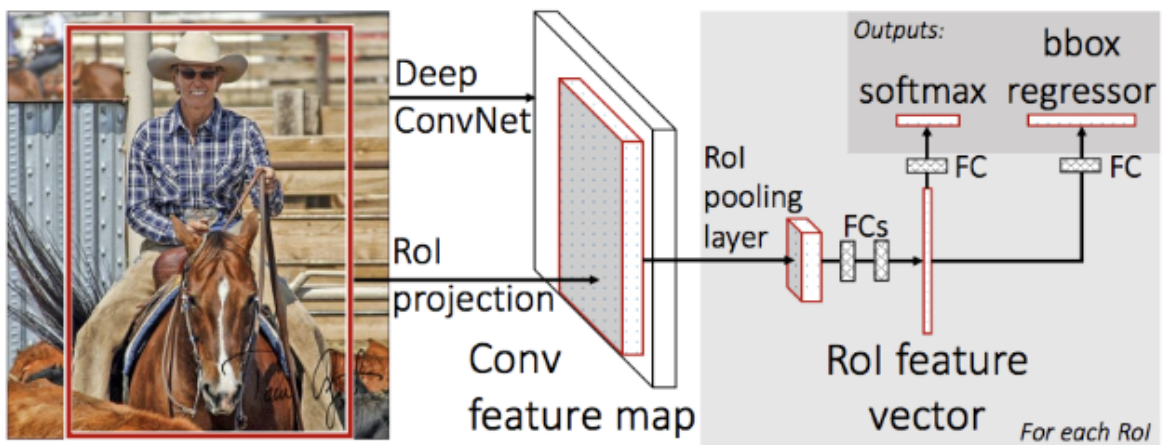


Figure 3.12 Fast RCNN

Faster RCNN

Faster RCNN is the first end-to-end, and the first near real time deep learning detector (COCO mAP@.5=42.7%, COCO mAP@[.5,.95]=21.9%, VOC07 mAP=73.2%, VOC12 mAP=70.4%, 17fps with ZFNet). The main contribution of Faster-RCNN is the introduction of Region Proposal Network (RPN) that enables nearly cost-free region proposals. From R-CNN to Faster RCNN, most individual blocks of an object detection system, e.g., proposal detection, feature extraction, bounding box regression, etc, have been gradually integrated into a unified, end-to-end learning

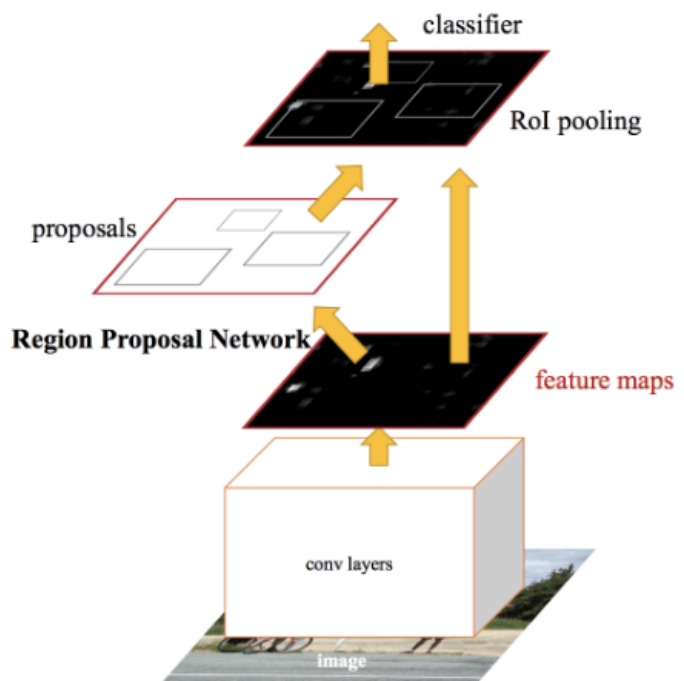


Figure 3.13 Faster R-CNN

framework. Similar to Fast R-CNN, the image is provided as an input to a convolutional network which provides a convolutional feature map. Instead of using a selective search algorithm on the feature map to identify the region proposals, a

separate network is used to predict the region proposals. The predicted region proposals are then reshaped using a RoI pooling layer which is then used to classify the image within the proposed region and predict the offset values for the bounding boxes.

YOLO

YOLO was proposed by R. Joseph et al. in 2015. It stands for “You Only Look Once”. It can be seen from its name that the authors have completely abandoned the previous detection paradigm of “proposal detection + verification”. YOLO uses a totally different approach. YOLO is a clever convolutional neural network (CNN) for doing object detection in real-time. The algorithm applies a single neural network to the full image, and then divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities.

YOLO is popular because it achieves high accuracy while also being able to run in real-time. The algorithm “only looks once” at the image in the sense that it requires only one forward propagation pass through the neural network to make predictions. After non-max suppression (which makes sure the object detection algorithm only detects each object once), it then outputs recognized objects together with the bounding boxes. YOLO suffers from a drop of the localization accuracy compared with two-stage detectors, especially for some small objects. YOLO’s subsequent versions and the latter proposed SSD have paid more attention to this problem. YOLO is extremely fast: a fast version of YOLO runs at 155fps with VOC07 mAP=52.7%, while its enhanced version runs at 45fps with VOC07 mAP=63.4% and VOC12 mAP=57.9%.

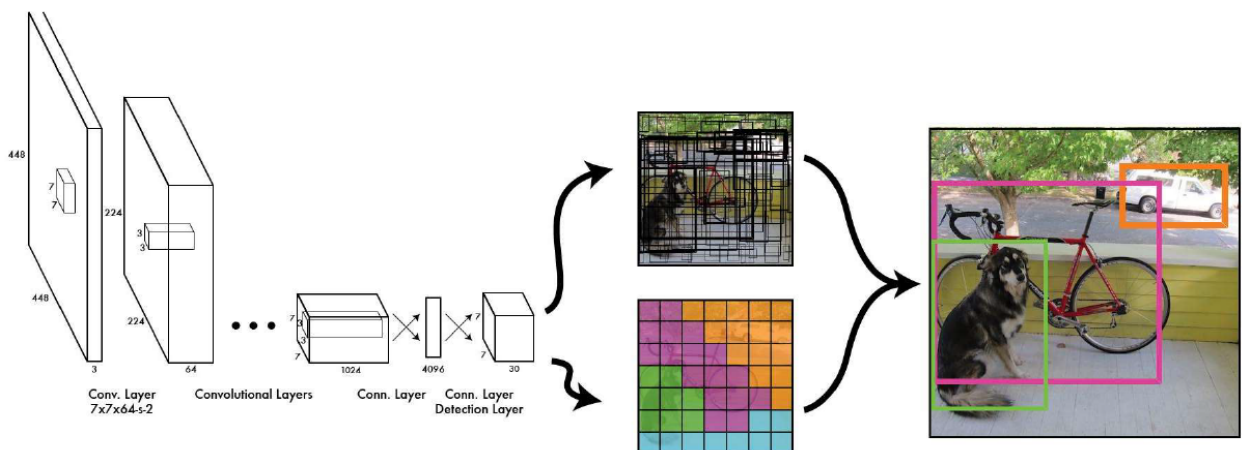
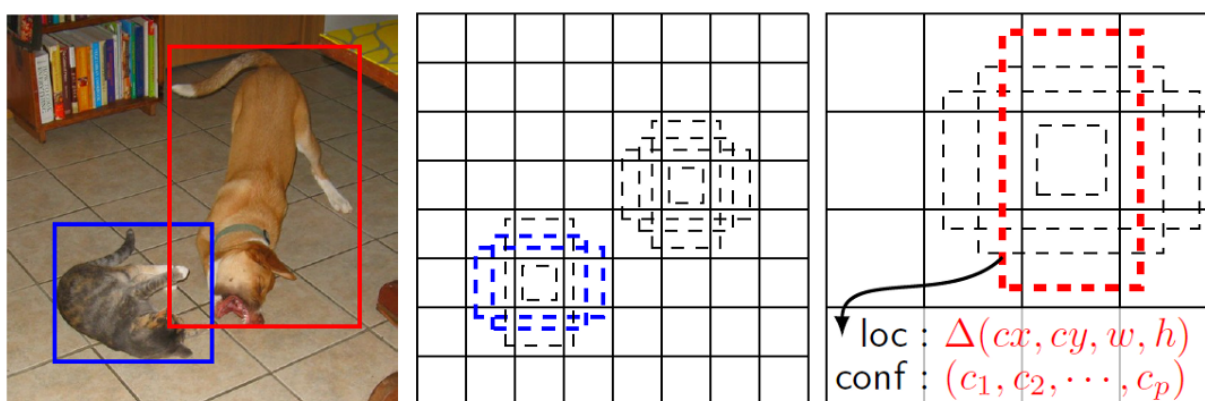


Figure 3.14 YOLO object detection

SSD

It was proposed by W. Liu et al. in 2015. The main contribution of SSD is the introduction of the multi-reference and multi-resolution detection techniques, which significantly improves the detection accuracy of a one-stage detector, especially for some small objects. The SSD is a multibox detector:

- After going through a certain convolutions for feature extraction, we obtain a feature layer of size $m \times n$ (number of locations) with p channels, such as 8×8 or 4×4 above. And a 3×3 conv is applied on this $m \times n \times p$ feature layer.
- For each location, we got k bounding boxes. These k bounding boxes have different sizes and aspect ratios. The concept is, maybe a vertical rectangle is more fit for humans, and a horizontal rectangle is more fit for cars.
- For each of the bounding box, we will compute c class scores and 4 offsets relative to the original default bounding box shape.
- Thus, we got $(c+4)kmn$ outputs.



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

Figure 3.15 SSD object detection

SSD has advantages in terms of both detection speed and accuracy (VOC07 mAP=76.8%, VOC12 mAP=74.9%, COCO mAP@.5=46.5%, mAP@[.5,.95]=26.8%, a fast version runs at 59fps). The main difference between SSD and any previous detectors is that the former one detects objects of 5 different scales on different layers of the network, while the latter ones only run detection on their top layers. The SSD, has fallen behind two stage detectors in terms of accuracy for years, till the RetinaNet was introduced.

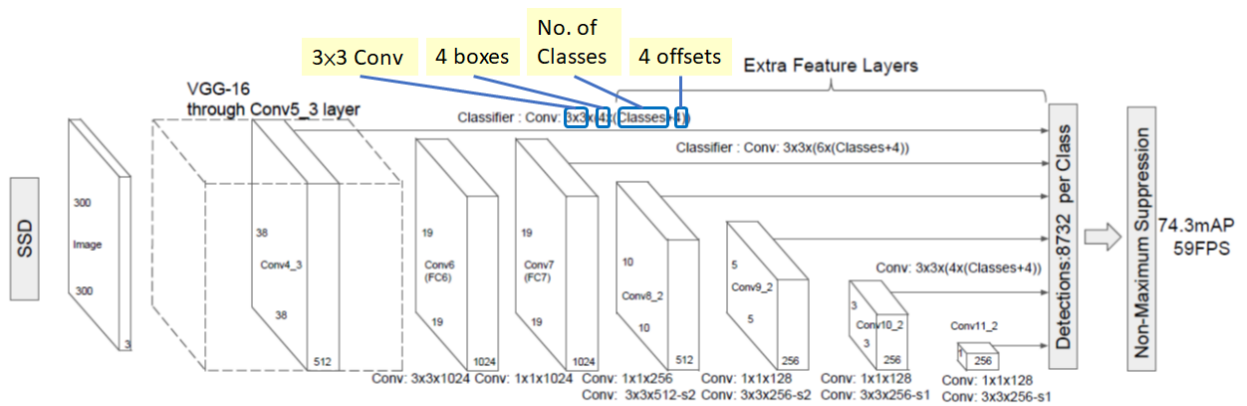


Figure 3.16 SSD object detection layers

RetinaNet

It was developed by Facebook AI Research (FAIR), and is reviewed. It is discovered that there is an extreme foreground-background class imbalance problem in one-stage detector. And it is believed that this is the central cause which makes the performance of one-stage detectors inferior to two-stage detectors. RetinaNet achieves state-of-the-art performance, outperforming Faster R-CNN, the well-known two-stage detectors. In essence, RetinaNet is a composite network composed of:

- A backbone network called Feature Pyramid Net, which is built on top of ResNet and is responsible for computing convolutional feature maps of an entire image;
- A subnetwork responsible for performing object classification using the backbone's output;
- A subnetwork responsible for performing bounding box regression using the backbone's output.

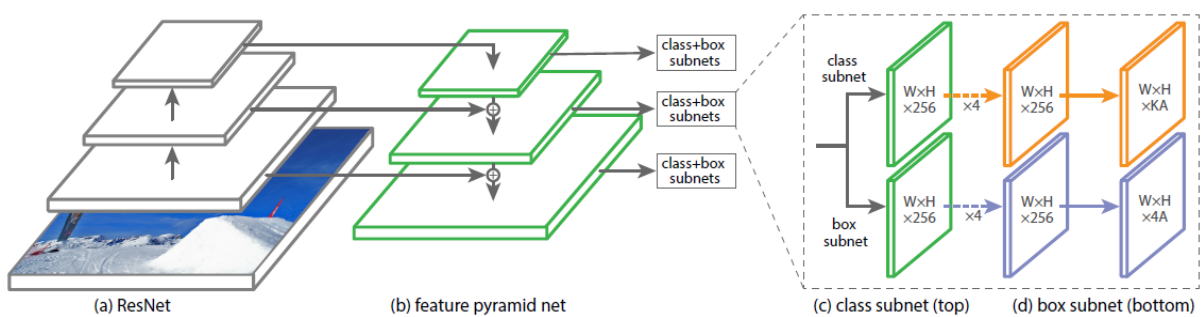


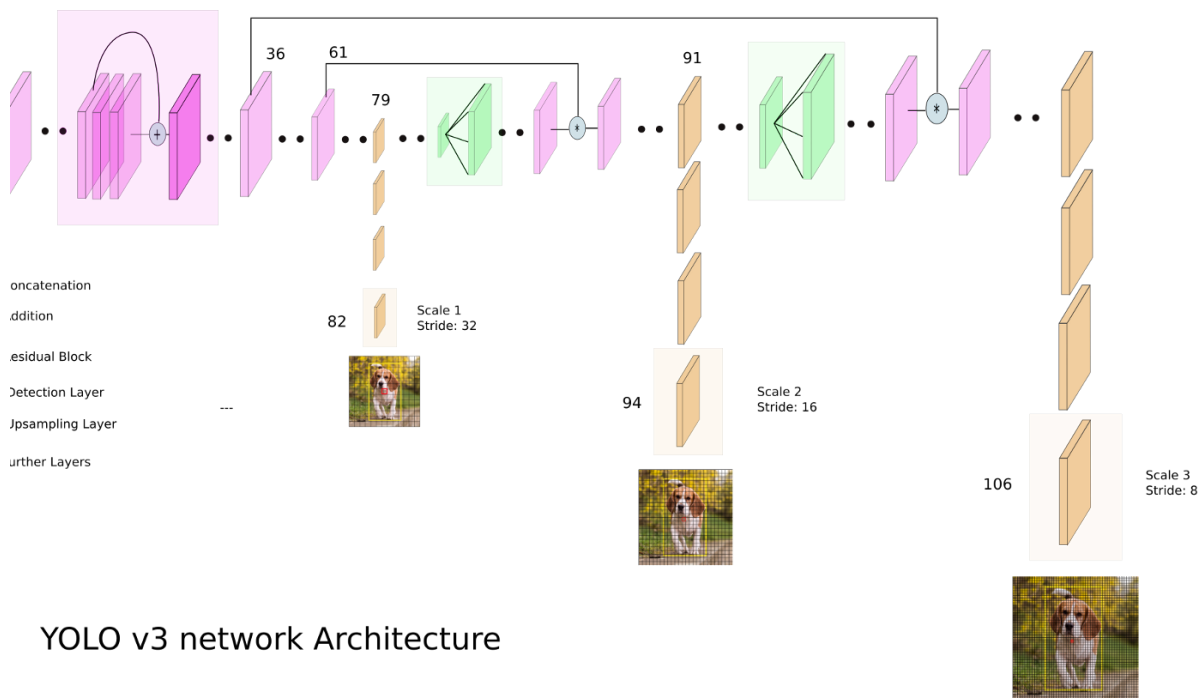
Figure 3.17 RetinaNet object detection

YOLOv3

YOLOv1, and YOLOv2 were the fastest and one of the most accurate object detectors. However, after their release, other algorithms like SSD and RetinaNet outperformed YOLO in terms of accuracy. YOLOv3, sorts out that issue by trading off some of its speed to increase detection accuracy.

Most of the trick in YOLOv3 increase in accuracy, lies in its backbone, the deep learning architecture, Darknet-53. Previous versions on YOLO using previous versions of darknet were still lacking some of the most important elements that are now staple in most of state-of-the-art algorithms, they had no residual blocks, no skip connections and no upsampling. YOLO v3 incorporates all of these.

YOLOv3 architecture has a 53 layer network trained on Imagenet. For the task of detection, 53 more layers are stacked onto it, giving us a 106 layer fully convolutional underlying architecture for YOLO v3. This is the reason behind the slowness of YOLO v3 compared to YOLO v2.



YOLO v3 network Architecture

Figure 3.18 YOLOV3 object detection

In YOLO v3, the detection is done by applying 1 x 1 detection kernels on feature maps of three different sizes at three different places in the network. The first detection is made by the 82nd layer. For the first 81 layers, the image is down

sampled by the network, such that the 81st layer has a stride of 32. If we have an image of 416 x 416, the resultant feature map would be of size 13 x 13. One detection is made here using the 1 x 1 detection kernel, giving us a detection feature map of 13 x 13 x 255.

Then, the feature map from layer 79 is subjected to a few convolutional layers before being sampled by 2x to dimensions of 26 x 26. This feature map is then depth concatenated with the feature map from layer 61. Then the combined feature map is again subjected to a few 1 x 1 convolutional layers to fuse the features from the earlier layer (61). Then, the second detection is made by the 94th layer, yielding a detection feature map of 26 x 26 x 255.

A similar procedure is followed again, where the feature map from layer 91 is subjected to few convolutional layers before being depth concatenated with a feature map from layer 36. Like before, a few 1 x 1 convolutional layers follow to fuse the information from the previous layer (36). We make the final of the 3 at 106th layer, yielding a feature map of size 52 x 52 x 255.

3.2.2 Comparative Study on Object Detection and Recognition

YOLOv3 massively outperforms other state of art detectors like RetinaNet, while being considerably faster, at COCO mAP 50 benchmark. It is also better than the SSD and it's variants. Here's a comparison of performances right from the paper (IoU = 0.5).

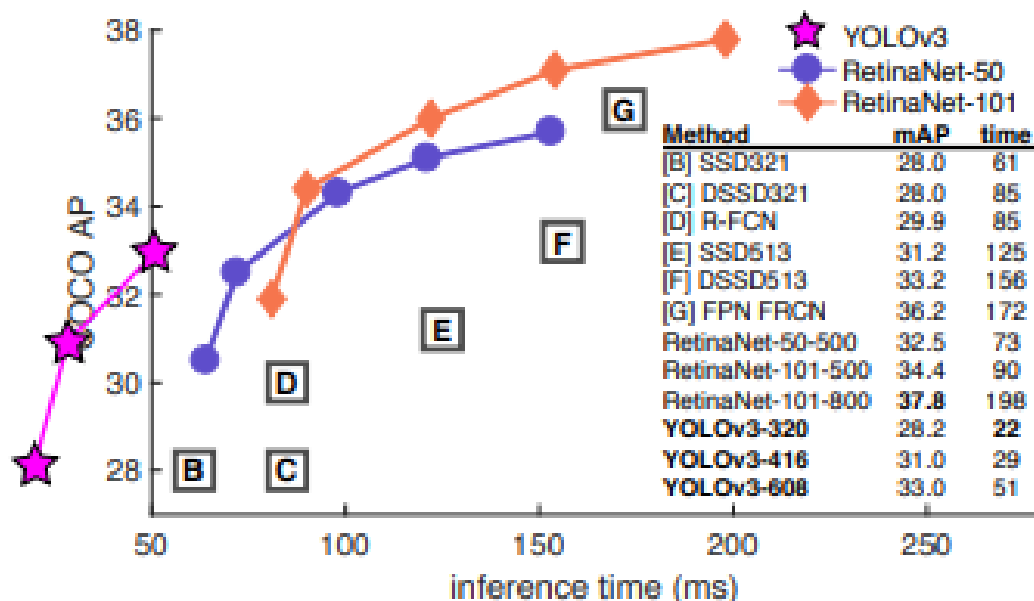


Figure 3.19 Comparative Study on Object Detection and Recognition

In benchmarks where the IoU is higher (say, COCO 75), the boxes need to be aligned more perfectly to be not rejected by the evaluation metric. Here is where YOLO is outdone by RetinaNet, as it's bounding boxes are not aligned as well as of RetinaNet. Here's a detailed table for a wider variety of benchmarks.

| | backbone | AP | AP ₅₀ | AP ₇₅ | AP _S | AP _M | AP _L |
|---------------------------|--------------------------|-------------|------------------|------------------|-----------------|-----------------|-----------------|
| <i>Two-stage methods</i> | | | | | | | |
| Faster R-CNN+++ [5] | ResNet-101-C4 | 34.9 | 55.7 | 37.4 | 15.6 | 38.7 | 50.9 |
| Faster R-CNN w FPN [8] | ResNet-101-FPN | 36.2 | 59.1 | 39.0 | 18.2 | 39.0 | 48.2 |
| Faster R-CNN by G-RMI [6] | Inception-ResNet-v2 [21] | 34.7 | 55.5 | 36.7 | 13.5 | 38.1 | 52.0 |
| Faster R-CNN w TDM [20] | Inception-ResNet-v2-TDM | 36.8 | 57.7 | 39.2 | 16.2 | 39.8 | 52.1 |
| <i>One-stage methods</i> | | | | | | | |
| YOLOv2 [15] | DarkNet-19 [15] | 21.6 | 44.0 | 19.2 | 5.0 | 22.4 | 35.5 |
| SSD513 [11, 3] | ResNet-101-SSD | 31.2 | 50.4 | 33.3 | 10.2 | 34.5 | 49.8 |
| DSSD513 [3] | ResNet-101-DSSD | 33.2 | 53.3 | 35.2 | 13.0 | 35.4 | 51.1 |
| RetinaNet [9] | ResNet-101-FPN | 39.1 | 59.1 | 42.3 | 21.8 | 42.7 | 50.2 |
| RetinaNet [9] | ResNeXt-101-FPN | 40.8 | 61.1 | 44.1 | 24.1 | 44.2 | 51.2 |
| YOLOv3 608 × 608 | Darknet-53 | 33.0 | 57.9 | 34.4 | 18.3 | 35.4 | 41.9 |

Table 3.1: Wider variety of object detection benchmarks

3.2.3 Implemented Approach of Object Detection and Recognition

From comparing benchmark results on various state-of-the-art object detection techniques, it can be concluded that there is a tradeoff between detection accuracy and inference time. In an implementation of a driverless racing vehicle, an object detection has to be accurate, but at the same time real time and able to process frames at high FPS; that's why a YOLOv3 based approach was selected. YOLOv3 maintains a very good accuracy level, but is considerably faster than any other object detection approach. Instead of using slow and computationally intensive cascade and sliding window approaches, We employ a quick, real-time and powerful object detector in our pipeline in the form of YOLOv3. Its ability to be fine-tuned with lesser data pre-trained weights and robust outputs made it the right fit in our system.

We divided our cone detection process along two YOLOv3 networks that work together simultaneously, one tinyYOLOv3 based network that detects the three cone types all as one class, this ensures that the most number of cones in each frame is detected and no cones are missed. The other one is a YOLOv3 based network that classifies only cones at close distances from the vehicle into one of three classes (yellow-blue-orange), ensuring maximum accuracy. Both the networks were implemented by the team members from scratch in Python using the PyTorch framework.

The network architecture by default each YOLO layer has 255 outputs: 85 values per anchor [4 box coordinates + 1 object confidence + 80 class confidences], times 3 anchors. The settings to filters=[5 + n] * 3 and classes=n, where n is the class count. This modification was made in all 3 YOLO layers in both networks.

We customized the first network by making a changes in YOLOV3 pipeline and reducing the number of classes that it detects, as “Our driverless high performance formula race car” does not really care about detecting cats, dogs, airplanes or bikes to name a few but needs to distinguish and detect ‘yellow’, ‘blue’ and ‘orange’ cones that provide information about the track. We reduce the classes of the pre-trained YOLOv3 to 3 classes which are ‘yellow’, ‘blue’ and ‘orange’ cones as shown in the following figure. For more information about the optimization and customization of the modified YOLOV3 to fit our high performance perception pipeline is described in chapter4.

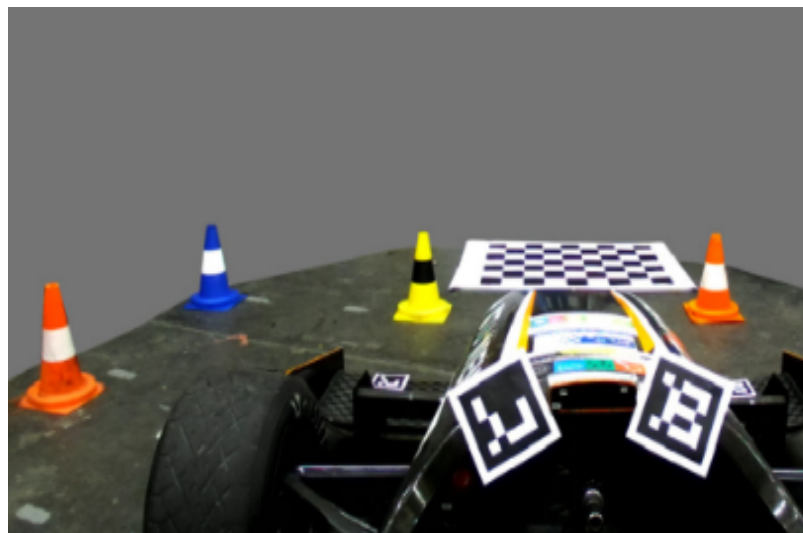


Figure 3.20 Yellow, blue, and orange cones

3.3 Literature Review on 3D Object Localization and Pose Estimation From Single Images

Object Detection and Recognition serves as an image 2D localization method where it finds where the object is in a 2D single frame and assigns it to a class. Now, due to the advancements in Computer Vision, 2D images can be used to identify the position of the object in the scene captured by the camera just by knowing extra information, and we are interested in introducing an accurate, robust and efficient way to estimate the 3D position of the objects from single frames in which it complies with our application i.e. mapping and finding cones in 3D space. This is to be the most important contribution which is to accurately estimate the 3D position of an object (Traffic cone in our application).

3.3.1 Background on 3D Object Localization and Pose Estimation From Single Images

We believe that human vision can estimate the depth of an object with one eye; this is due to the fact that the geometric features of different objects in life have been previously added to the knowledge of the human brain in addition to the geometric relationship between objects that serves to give a great estimate about where the object is. In this problem, we suppose that a priori knowledge of the 3D geometric information about the object is available alongside the camera intrinsic parameters used to capture the frames.

Pose estimation and key points regression have appeared in previous research work that is including [Viewpoint aware object detection and continuous pose estimation] and [3d generic object categorization, localization and pose estimation]. Glasner et al. estimate pose for images containing cars using an ensemble of voting SVMs, Tulsiani et al. [S. Tulsiani and J. Malik. Viewpoints and keypoints.] use features and convolutional neural networks to predict viewpoints of different objects. Their work captures the interplay between viewpoints of objects and key points for specific objects. We will briefly demonstrate their approaches and limitations.

Viewpoint Aware Object Detection Approach

Glasner et al. introduced a method to directly integrate 3D reasoning with an appearance based voting architecture. The method relies on a nonparametric representation of a joint distribution of shape and appearance of the object class. The voting method employs a novel parameterization of joint detection and viewpoint

hypothesis space, allowing efficient accumulation of evidence. In addition to combining this with a re-scoring and refinement mechanism, using an ensemble of view-specific support vector machines.



Figure 3.21 The voting process in Viewpoint Aware Object Detection Approach

This figure shows the voting process. Four patches from the test image (top left) are matched to database patches. The matching patches are shown with the corresponding color on the right column. Each match generates a vote in 6D pose space. The point in pose space is parameterized as a projection of designated points in 3D onto the image plane. These projections are shown here as dotted triangles. The red, green and blue votes correspond to a true detection, the cast pose votes are well clustered in pose space (bottom left) while the yellow match casts a false vote.

Viewpoints and Key Points Approach

In this approach S. Tulsiani and J. Malik characterize the problem of pose estimation for rigid objects in terms of determining viewpoint to explain coarse pose and keypoint prediction to capture the finer details, And address both these tasks in two different settings the constrained setting with known bounding boxes and the more challenging detection setting where the aim is to simultaneously detect and correctly

estimate pose of objects. And they present Convolutional Neural Network based architectures for these and demonstrate that leveraging viewpoint estimates can substantially improve local appearance based keypoint prediction.

The components of the proposed model is viewpoint prediction and Local Appearance based Keypoint Activation. In the viewpoint prediction they trained a CNN based architecture which can implicitly capture and aggregate local evidence for predicting the euler angles to obtain a viewpoint estimate. In the Local Appearance based Keypoint Activation they proposed a fully convolutional CNN based architecture to model local part appearance. And capture the appearance at multiple scales and combine the CNN responses across scales to obtain a resulting heatmap which corresponds to a spatial log-likelihood distribution for each keypoint.

The following figure will illustrate an overview on this approach:

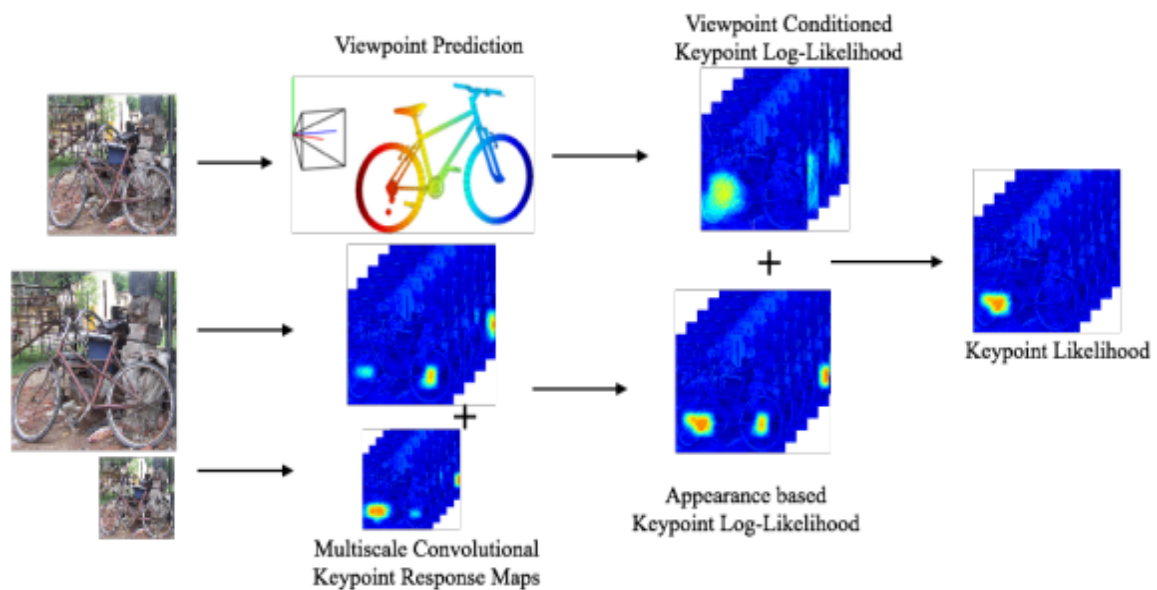


Figure 3.22 Viewpoints and Key Points Approach

To recover an estimate of the global pose, they use a CNN based architecture to predict viewpoint. For each keypoint, a spatial likelihood map is obtained via combining multiscale convolutional response maps and it is then combined with a likelihood conditioned on predicted viewpoint to obtain the final predictions.

3.3.2 Comparative Study on 3D Object Localization and Pose Estimation From Single Images

Although these approaches are considered solutions to the given problem, these are not enough for our application as they don't provide real time, accuracy, and robustness.

The limitations of the viewpoint aware object detection approach is that the pose estimates generated in the voting stage are not always accurate. When the pose estimate is incorrect the wrong viewpoint specific classifier will be applied which leads to that the 3D position will not be measured accurately. The experiments for the approach gave a maximum average precision (AP) of 0.32 on the Pascal VOC 2007 cars dataset using the 3D voting and 8view-SVM detector.

| | 2D voting | 2D voting + SVM | 3D voting | 3D voting + SVM | 3D voting + 8view-SVM |
|----|-----------|-----------------|-----------|-----------------|-----------------------|
| AP | 15.86% | 24.34% | 16.29% | 27.97% | 32.03% |

Table 3.2: Pascal VOC 2007 cars. Average precision achieved by our detectors compared to a 2D baseline.

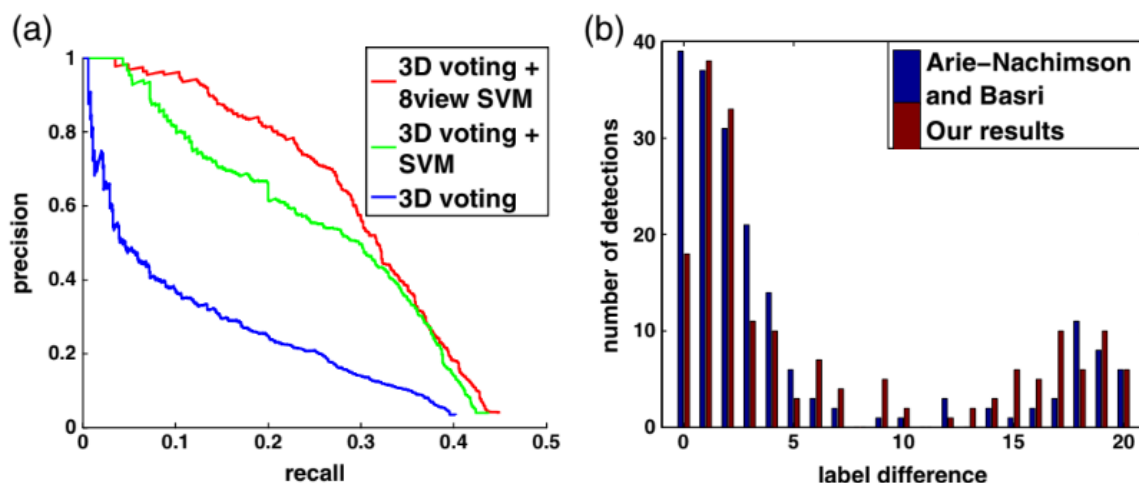


Figure 3.23: Pascal VOC 2007 cars

(a) Recall-precision. 3D voting followed by 8-view SVM (red) outperforms 3D voting (blue) and 3D voting followed by SVM (green). The approach achieved an average precision of 32.03% without using positive training examples from Pascal. (b) Pose estimation. A subset of the cars was annotated with one of 40 different labels corresponding to approximately uniform samples of the azimuth range. They showed their label differences alongside those reported in a similar old approach proposed by [M].

Arie-Nachimson, R. Basri, Constructing implicit 3D shape models for pose estimation].

In addition, the limitations of Viewpoints and Key Points approach in our case of designing a race driverless car in some climate conditions if the frame image has a problem the detection of the viewpoint will not be accurately predicted. The detection candidate has an associated viewpoint and the detection is labeled correct if it has a correct predicted viewpoint bin as well as a correct localization because that the two stages have a strong relation and the error will be highly propagated. The second limitation is that this model is relatively slow compared to the realtime race vehicle perception system. We must say that the fraction of error this approach proposes is away from our targeted performance. The metrics used to evaluate the performance are the median error and Accuracy at given viewport angle.

Median Error

The common confusions for the task of viewpoint estimation often are predictions which are far apart (eg. left facing vs right facing car) and the median error (MedErr) is a widely used metric that is robust to these if a significant fraction of the estimates are accurate.

Accuracy at θ

A small median error does not necessarily imply accurate estimates for all instances, a complementary performance measure is the fraction of instances whose predicted viewpoint is within a fixed threshold of the target viewpoint. Denoted this metric by Acc_{θ} where θ is the threshold. The results use $\theta = \pi/6$.

| | aero | bike | boat | bottle | bus | car | chair | table | mbike | sofa | train | tv | mean |
|------------------------------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| $Acc_{\frac{\pi}{6}}$ (Pool5-TNet) | 0.27 | 0.18 | 0.36 | 0.81 | 0.71 | 0.36 | 0.52 | 0.52 | 0.38 | 0.67 | 0.7 | 0.71 | 0.52 |
| $Acc_{\frac{\pi}{6}}$ (fc7-TNet) | 0.5 | 0.44 | 0.39 | 0.88 | 0.81 | 0.7 | 0.39 | 0.38 | 0.48 | 0.44 | 0.78 | 0.65 | 0.57 |
| $Acc_{\frac{\pi}{6}}$ (ours-TNet) | 0.78 | 0.74 | 0.49 | 0.93 | 0.94 | 0.90 | 0.65 | 0.67 | 0.83 | 0.67 | 0.79 | 0.76 | 0.76 |
| $Acc_{\frac{\pi}{6}}$ (ours-ONet) | 0.81 | 0.77 | 0.59 | 0.93 | 0.98 | 0.89 | 0.80 | 0.62 | 0.88 | 0.82 | 0.80 | 0.80 | 0.81 |
| $MedErr$ (Pool5-TNet) | 42.6 | 52.3 | 46.3 | 18.5 | 17.5 | 45.6 | 28.6 | 27.7 | 37 | 25.9 | 20.6 | 21.5 | 32 |
| $MedErr$ (fc7-TNet) | 29.8 | 40.3 | 49.5 | 13.5 | 7.6 | 13.6 | 45.5 | 38.7 | 31.4 | 38.5 | 9.9 | 22.6 | 28.4 |
| $MedErr$ (ours-TNet) | 14.7 | 18.6 | 31.2 | 13.5 | 6.3 | 8.8 | 17.7 | 17.4 | 17.6 | 15.1 | 8.9 | 17.8 | 15.6 |
| $MedErr$ (ours-ONet) | 13.8 | 17.7 | 21.3 | 12.9 | 5.8 | 9.1 | 14.8 | 15.2 | 14.7 | 13.7 | 8.7 | 15.4 | 13.6 |

Table3.3: Viewpoint Estimation with Ground Truth box Performance.

Accordingly, we looked for using state of the art solutions in computer vision and image processing to accurately localize the landmark while racing on the track, and we came up with the pipeline that provides the required performance.

3.3.3 Implemented Approach on 3D Object Localization and Pose Estimation From Single Images

3D Pose estimation from a single frame has always been an ill-posed problem, but it can be solved accurately when there is previous geometric knowledge about the objects introduced in the model. In this work, we have decided to build a DNN based on ResNet to extract accurately the key points from single image cone patches, these key points are mapped to 3D known model points in the local frame of the cone. At this point, a perspective n-points algorithm runs to find the 6 DOF of the camera with respect to the cone position that includes 3D position and 3 orientation angles, we are only interested in the XZ position that represents the top view location in the base map with respect to the vehicle. These reactive cone mappings will be used by the SLAM algorithm to get the global position of the cone in the world frame F_w . This approach will be discussed in details in chapter 4.

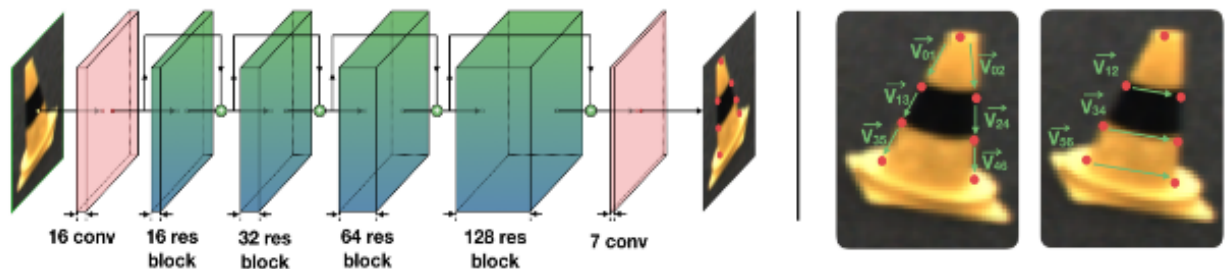


Figure 3.24: The developed DNN to extract cone key points.

3.4 Literature review on Mapping and Localization

At this instance of our progress in the full autonomous driving kit literature review, we have concluded multiple decisions that provided us with the landmarks (cones) estimates from our sensors. Now, we will research and discuss how these estimates can be fused in order to get an accurate landmark 3D position. In addition, these landmarks positions are referenced to the local vehicle frame that we need to translate to a global map which we are interested in constructing in order to generate accurate paths for the vehicle to drive itself through as accurately as possible and with extreme performance. Absolutely, this translation needs extra input information to be done which is the vehicle position and orientation that also needs to be the output of a sensor fusion process from the vehicle input sensors. Lastly, we need to combine the two past mentioned tasks which are Mapping and Localization into one algorithm in order to manipulate uncertainty in the environment with a chosen algorithm for Synchronous Localization and Mapping (SLAM). With this in mind, we started our review with the work of Sebastian Thrun, Wolfram Burgard, and Dieter Fox in the famous basis of Robotics which is presented in the reference book Probabilistic Robotics [19].

3.4.1 Background on Mapping and Localization

One of the main motives in this work is our understanding of uncertainty, and how we need to control the random process of what we are trying to achieve which is a vehicle that has to drive itself accurately and with extreme performance on a given track but the thing is, the way this vehicle act and react to the environment is not absolute. This means that the place of the landmarks the vehicle receive is not an absolute location but either the best estimate of where the cone is in the environment, in addition, how the AI agent knows where is the vehicle position with respect to the environment is not absolute but rather a good estimate of where the vehicle could be. This understanding helps in introducing redundant perception; this includes how we can fuse multiple sensor estimates to establish a system state that best describes the vehicle and how it should react to the environment.

Robot Uncertainty

Uncertainty arises if the robot lacks critical information for carrying out its task. It arises from five different factors.

1. **Environments.** Physical worlds are inherently unpredictable. While the degree of uncertainty in well-structured environments such as assembly lines is small, environments such as highways and private homes are highly dynamic and unpredictable.
2. **Sensors.** Sensors are inherently limited in what they can perceive. Limitations arise from two primary factors. First, range and resolution of a sensor is subject to physical laws. For example, Cameras can't see through walls, and even within the perceptual range the spatial resolution of camera images is limited. Second, sensors are subject to noise, which perturbs sensor measurements in unpredictable ways and hence limits the information that can be extracted from sensor measurements.
3. **Robots.** Robot actuation involves motors that are, at least to some extent, unpredictable, due effects like control noise and wear-and-tear. Some actuators, such as heavy-duty industrial robot arms, are quite accurate. Others, like low-cost mobile robots can be extremely inaccurate.
4. **Models.** Models are inherently inaccurate. Models are abstractions of the real world. As such, they only partially model the underlying physical processes of the robot and its environment. Model errors are a source of uncertainty that has largely been ignored in robotics, despite the fact that most robotic models used in state-of-the-art robotics systems are rather crude.
5. **Computation.** Robots are real-time systems, which limits the amount of computation that can be carried out. Many state-of-the-art algorithms are approximate, achieving timely response through sacrificing accuracy.

All of these factors give rise to uncertainty. Traditionally, such uncertainty has mostly been ignored in robotics. However, as robots are moving away from factory floors into increasingly unstructured environments, the ability to cope with uncertainty is critical for building successful robots.

To address the problem of mapping and localization we need to discuss some basics regarding **Recursive State Estimation**, **Gaussian Filters**, and **Simultaneous Localization and Mapping**.

A. Recursive State Estimation

In order to deploy an accurate autonomous system that can map its surrounding and localize itself in order to know what to do next, we should consider it as a mobile wheeled robot in which we should discuss the core definition of probabilistic robotics which is how to estimate useful information and quantities from multiple sensors data which can't be used directly because of the uncertainty by which each sensor implies with its readings. Like the rest of robotics applications, once this information is known it is easy to know what to do next. Let's say, we have a moving robot, it is easy to know where to go if at each time instance we know exactly where all the landmarks are and where the robot is. Unfortunately, these variables can't be known directly from the sensors data. Sensors import only partial information about these variables, because their measurements are accompanied with noise. Accordingly, Thrun and the rest introduced **State estimation**. A method used to recover what they called state variables in which it generates and computes the possible world states which the robot may be in, this is called **belief distributions** according to them.

Next, we will discuss what is concerned with the robot process while interacting with its environment.

1) **State**. Environments can be identified by its state. It will be nice to think of the state as the group of all elements that defines the robot and its environment that can affect the future. The state may include variables regarding the robot itself, such as its pose (position and orientation) and velocity. In the book, they denoted the state by x ; although the specific variables included in x will depend on the context. The state at time t will be denoted x_t . Typical concerned state variables are:

- ❑ **The robot pose**, simply it is the location and orientation relative to the global coordinate frame.
- ❑ **The location and features of surrounding objects in the environment**. These objects (aka landmarks) may be a cone, tree, or a wall. Robot environments may have a few dozen of these landmarks and up to hundreds and billions of state variables.
- ❑ **The location and velocities of moving objects and people**. Sometimes, there are other moving actors with the robot. These actors have their own kinematic and dynamic state. In our work the dynamic objects are irrelevant as we are only considering static objects (cones).
- ❑ There can be a huge number of other state variables such as temperature, robot battery level, etc.

2) **Environment Interaction**, We can classify the interactions of the mobile vehicle with the environment into two main types; The vehicle can affect the state of its environment through its motion. In addition, it can collect information about the state by using its sensors. Both types of interactions may occur at the same time.

- ❑ **Sensor measurements.** Perception is the process by which the robot uses its sensors to obtain information about the state of its environment. The result of such a perceptual interaction will be called a measurement, although we will sometimes also call it observation or percept. In fact, sensor measurements arrive with some delay. They provide information about the state a few moments ago.
- ❑ **Control actions.** These are used by the robot to change the state of the world. They do so by actively asserting forces on the robot's environment. Even if the robot does not perform any action itself, state usually changes. Thus, for consistency, we will assume that the robot always executes a control action, even if it chooses not to move any of its motors. In practice, the robot continuously executes controls and measurements are made **concurrently**.

Hypothetically, a robot may keep a record of all past sensor measurements and control actions. We will refer to such a collection as the data (regardless of whether they are being memorized). In accordance with the two types of environment interactions, the robot has access to two different data streams.

- ❑ **Measurement data** provides information about a momentary state of the environment. Examples of measurement data include camera images, range scans, and so on. For most parts, we will simply ignore small timing effects (e.g., most lidar sensors scan environments sequentially at very high speeds, but we will simply assume the measurement corresponds to a specific point in time). The measurement data at time t will be denoted as Z_t .
- ❑ **Control data** carry information about the change of state in the environment. In mobile robotics, a typical example of control data is the velocity of a robot. Setting the velocity to 10 cm per second for the duration of five seconds suggests that the robot's pose, after executing this motion command, is approximately 50 cm ahead of its pose before command execution. Thus, its main information regards the change of state. An alternative source of control data are odometers. Odometers are sensors that measure the revolution of a robot's wheels. As such they convey information about the change of the state. Even though

odometers are sensors, we will treat odometry as control data, since its main information regards the change of the robot's pose.

Control data will be denoted u_t . The variable u_t will always correspond to the change of state in the time interval $[t-1; t]$. As before, we will denote sequences of control data by $u_{t_1:t_2}$, for $t_1 \leq t_2$:

$$u_{t_1:t_2} = u_{t_1}, u_{t_1+1}, u_{t_1+2}, \dots, u_{t_2}$$

Since the environment may change even if a robot does not execute a specific control action, the fact that time passed by constitutes, technically speaking, control information. Hence, we assume that there is exactly one control data item per time step t .

Important Note

The distinction between measurement and control is a crucial one, as both types of data play fundamentally different roles in the material yet to come. Perception provides information about the environment's state, hence it tends to increase the robot's knowledge. Motion, on the other hand, tends to induce a loss of knowledge due to the inherent noise in robot actuation and the stochasticity of robot environments; although sometimes a control makes the robot more certain about the state. By no means is our distinction intended to suggest that actions and perceptions are separated in time, i.e., that the robot does not move while taking sensor measurements. Rather, perception and control takes place concurrently; many sensors affect the environment; and the separation is strictly for convenience.

Another key concept in probabilistic robotics is that of a **belief**. A belief reflects the robot's internal knowledge about the state of the environment. We already discussed that state cannot be measured directly. For example, a robot's pose might be $x = (14.12, 12.7, 0.755)$ in some global coordinate system, but it usually cannot know its pose, since poses are not measurable directly (not even with a GPS!). Instead, the robot must infer its pose from data. We therefore distinguish the true state from its internal belief, or state of knowledge with regards to that state. Probabilistic robotics represents beliefs through conditional probability distributions. A belief distribution assigns a probability (or density value) to each possible hypothesis with regards to the true state. Belief distributions are posterior probabilities over state variables conditioned on the available data. We will denote belief over a state variable x_t by $bel(x_t)$

, which is an abbreviation for the posterior

$$bel(x_t) = p(x_t | z_{1:t}, u_{1:t})$$

This posterior is the probability distribution over the state x_t at time t , conditioned on all past measurements $z_{1:t}$ and all past controls $u_{1:t}$. Generally, the belief for any system is modeled and updated using **Filters** with the help of the received sensor measurements and the control inputs in the previous instant of time.

Bayes Filters

The most general algorithm for calculating beliefs is given by the Bayes filter algorithm. This algorithm calculates the belief distribution from measurement and control data.

| | |
|----|--|
| 1: | Algorithm Bayes_filter ($bel(x_{t-1}), u_t, z_t$): |
| 2: | for all x_t do |
| 3: | $\overline{bel}(x_t) = \int p(x_t u_t, x_{t-1}) bel(x_{t-1}) dx$ |
| 4: | $bel(x_t) = \eta p(z_t x_t) \overline{bel}(x_t)$ |
| 5: | endfor |
| 6: | return $bel(x_t)$ |

Table 3.4: Algorithm for Bayes Filter Update Rule

The Bayes filter is recursive, that is, the belief $bel(x_t)$ at time t is calculated from the belief $bel(x_{t-1})$ at time $t-1$. Its input is the belief bel at time $t-1$, along with the most recent control u_t and the most recent measurement z_t . Its output is the belief $bel(x_t)$ at time t .

The above table only depicts a single step of the Bayes Filter algorithm: the update rule. This update rule is applied recursively, to calculate the belief $bel(x_t)$ from the belief $bel(x_{t-1})$, calculated previously.

The Bayes filter algorithm possesses two essential steps. In Line 3, it processes the control u_t . It does so by calculating a belief over the state x_t based on the prior belief over state x_{t-1} and the control u_t . In particular, the belief $bel(x_t)$ that the robot assigns to state x_t is obtained by the integral (sum) of the product of two distributions: the prior assigned to x_{t-1} , and the probability that control u_t induces a

transition from x_{t-1} to x_t . This update step is called the control update, or prediction.

The second step of the Bayes filter is called the measurement update. In Line 4, the Bayes filter algorithm multiplies the belief $bel(x_t)$ by the probability that the measurement z_t may have been observed. It does so for each hypothetical posterior state x_t . The resulting product is generally not a probability, that is, it may not integrate to 1. Hence, the result is normalized, by virtue of the normalization constant η . This leads to the final belief $bel(x_t)$, which is returned in Line 6 of the algorithm. [Refer to derivations in the appendix]

B. Gaussian Filters

Gaussian filters are efficient Bayes filter algorithms that represent the posterior by multivariate Gaussians. It is known that Gaussians can be represented in two different ways: The **moments representation** and the **canonical representation**.

The **moments representation** consists of the **mean (first moment)** and the **covariance (second moment)** of the Gaussian.

$$p(x) = \det(2\pi\Sigma)^{-\frac{1}{2}} \exp \left\{ -\frac{1}{2}(x - \mu)^T \Sigma^{-1}(x - \mu) \right\}$$

This density over the variable x is characterized by two sets of parameters: The mean and the covariance. The mean is a vector that possesses the same dimensionality as the state x . The covariance is a quadratic matrix that is symmetric and positive semidefinite. Its dimension is the dimensionality of the state x squared. Thus, the number of elements in the covariance matrix depends quadratically on the number of elements in the state vector. After this definition we are no longer in need to deal with the complex gaussian in each update we only need to deal with the mean and covariance to update the belief $bel(x_t)$ in what is called a state transition function that updates the new distribution moments from the old one.

The **canonical, or natural, representation** consists of an information matrix and an information vector. Both representations are duals of each other, and each can be recovered from the other via matrix inversion.

Bayes filters can be implemented for both representations. When using the moments representation, the resulting filter is called **Kalman filter**. The dual of the Kalman filter is the **information filter**, which represents the posterior in the canonical representation. Updating a Kalman filter based on a control is computationally

simple, whereas incorporating a measurement is more difficult. The opposite is the case for the information filter, where incorporating a measurement is simple, but updating the filter based on a control is difficult.

For both filters to calculate the correct posterior, three assumptions have to be fulfilled. **First**, the initial belief must be Gaussian. **Second**, the state transition probability must be composed of a function that is linear in its argument with added independent Gaussian noise. **Third**, the same applies to the measurement probability. It must also be linear in its argument, with added Gaussian noise.

Systems that meet these assumptions are called **linear Gaussian systems**. Both filters can be **extended to nonlinear problems**. This technique calculates a tangent to the nonlinear function. Tangents are linear, making the filters applicable. The technique for finding a tangent is called **Taylor expansion**. Performing a Taylor expansion involves calculating the first derivative of the target function, and evaluating it at a specific point. The result of this operation is a matrix known as the **Jacobian**. The resulting filters are called “**extended**.”

The accuracy of Taylor series expansions depends on two factors: The degree of nonlinearity in the system, and the width of the posterior. Extended filters tend to yield good results if the state of the system is known with relatively high accuracy, so that the remaining covariance is small. The larger the uncertainty, the higher the error introduced by the linearization.

One of the primary advantages of Gaussian filters is computational: The update requires time polynomial in the dimensionality of the state space. The primary disadvantage is their confinement to unimodal Gaussian distributions. Within the multivariate Gaussian regime, both filters, the Kalman filter and the information filter, have orthogonal strengths and weaknesses. However, **The Kalman filter** and its nonlinear extension, the **Extended Kalman Filter (EKF)**, are vastly more popular than the information filter.

Kalman Filter

This filter represents the state transition function that solves for a linear system model, where the probability distribution for state estimation based on motion is $p(x_t | u_t, x_{t-1})$ and based on the measurements is $p(z_t | x_t)$ where both updates the belief distribution $bel(x_t)$ in a consequent manner according to the bayes filter

algorithm line 3 and 4, motion increases uncertainty and measurements decreases it.

The KF applies three basic assumptions.

First

Linear Kalman Filter algorithm models the next state probability $p(x_t | u_t, x_{t-1})$ as a linear function in its arguments with added Gaussian noise. Consider this in the following expression.

$$x_t = A_t \cdot x_{t-1} + B_t \cdot u_t + \varepsilon_t$$

In this notation, x_t and x_{t-1} are state vectors, and u_t is the control vector at time t.

A_t and B_t are matrices. A_t is a square matrix of size $n \times n$, where n is the dimension of the state vector x_t . B_t is of size $n \times m$, with m being the dimension of the control vector u_t . By multiplying the state and control vector with the matrices A_t and B_t , respectively, the state transition function becomes linear in its arguments. Thus, Kalman filters assume linear system dynamics. The random variable ε_t is a **Gaussian random vector** that models the **randomness** in the state transition. It is of the same dimension as the state vector. Its mean is zero and its covariance will be denoted R_t . The mean of the posterior state is given by:

$$A_t \cdot x_{t-1} + B_t \cdot u_t$$

and the covariance by R_t :

$$p(x_t | u_t, x_{t-1}) = \det(2\pi R_t)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(x_t - A_t \cdot x_{t-1} - B_t \cdot u_t)^T R_t^{-1} (x_t - A_t \cdot x_{t-1} - B_t \cdot u_t)\right)$$

Until now we can predict the new distribution of the state based on the previous state and the control inputs this will be used in evaluating the step stated in line 3 in the bayes filter algorithm, this increases the uncertainty because of the uncertainty factors stated above in the Robot Uncertainty section.

Second

Next, KF algorithm models the measurement probability $p(z_t | x_t)$ also as a linear combination of its arguments, with added Gaussian noise.

$$z_t = C_t \cdot x_t + \delta_t$$

C_t is a matrix of size $k \times n$, where k is the dimension of the measurement

vector Z_t . The vector δ_t describes the **measurement noise**. Again, the distribution of δ_t is a multivariate Gaussian with zero mean and covariance Q_t . Consider the following multivariate normal distribution that describes the measurement probability.

$$p(z_t | x_t) = \det(2\pi Q_t)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(z_t - C_t \cdot x_t)^T Q_t^{-1} (z_t - C_t \cdot x_t)\right\}$$

This step depicts line 4 in the Bayes Filter which is the correction step, as it reduces the uncertainty of the state by using the sensor measurements.

The next algorithm shows how to apply KF algorithm for state estimation, by updating the moments representation of the belief; the mean of the distribution μ_t , and the covariance Σ_t .

[refer to the appendix for derivations]

- 1: **Algorithm Kalman filter**($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):
- 2: $\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$
- 3: $\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$
- 4: $K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$
- 5: $\mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t)$
- 6: $\Sigma_t = (I - K_t C_t) \bar{\Sigma}_t$
- 7: return μ_t, Σ_t

Table: Algorithm for Linear Kalman Filter

Kalman filters represent the belief $bel(x_t)$ at time t by the mean μ_t and the covariance Σ_t . The input of the Kalman filter is the belief at time t - 1, represented by μ_{t-1} and Σ_{t-1} . To update these parameters, Kalman filters require the control u_t and the measurement z_t . The output is the belief at time t, represented by μ_t and Σ_t .

In Lines 2 and 3, the predicted belief $\bar{\mu}$ and $\bar{\Sigma}$ is calculated representing the belief $\overline{bel(x_t)}$ one time step later, but before incorporating the measurement z_t . This belief is obtained by incorporating the control u_t .

The mean $\bar{\mu}$ is calculated using the above mentioned equation, with the mean μ_{t-1} substituted for the state x_{t-1} . The update of the covariance considers the fact that states depend on previous states through the linear matrix A_t . This matrix is multiplied twice into the covariance, since the covariance is a quadratic matrix.

The belief $\overline{bel(x_t)}$ is subsequently transformed into the desired belief $bel(x_t)$ in Lines 4 through 6, by incorporating the measurement z_t . The variable K_t computed in Line 4 is called Kalman gain. It specifies the degree to which the measurement is incorporated into the new state estimate. Line 5 manipulates the mean, by adjusting it in proportion to the Kalman gain K_t and the deviation of the actual measurement, z_t , and the measurement predicted according to the measurement probability mentioned above.

Finally, the new covariance of the posterior belief is calculated in Line 6, adjusting for the information gain resulting from the measurement.

For sure, there must be an initial state for this algorithm defined by the initial belief distribution $bel(x_0)$ denoted by the mean of the distribution μ_0 , and the covariance Σ_0 .

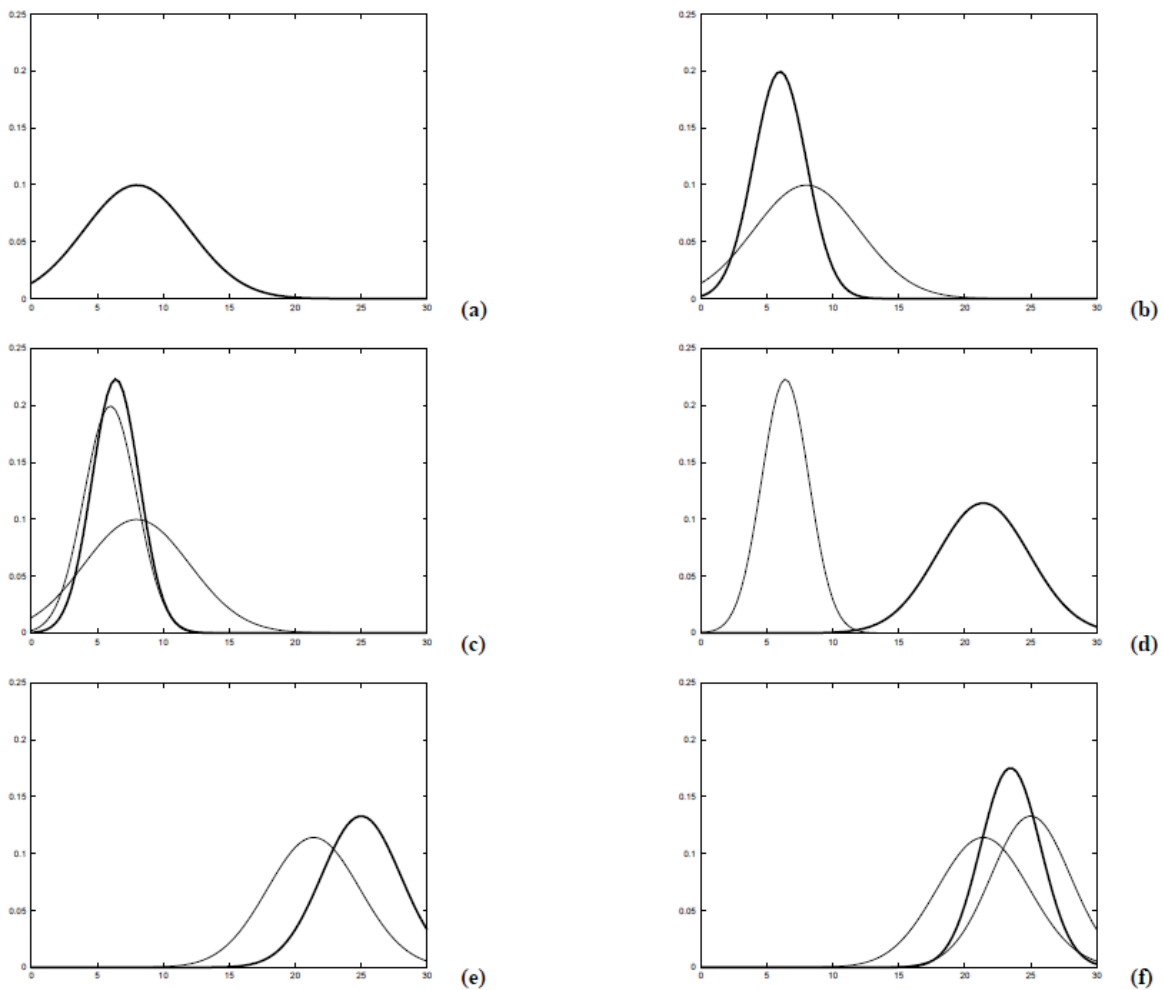


Figure 3.25: Illustration of Kalman filters: (a) initial belief, (b) a measurement (in bold)

with the associated uncertainty, (c) belief after integrating the measurement into the belief using the Kalman filter algorithm, (d) belief after motion to the right (which introduces uncertainty), (e) a new measurement with associated uncertainty, and (f) the resulting belief.

Example

In order to make things more clear, we provide a simple example on applying the Bayes Filter using Kalman Filter Algorithm.

We consider a simple construction of a car that exists in a one dimensional environment where its motion is considered only in the x direction and the state estimation transition is assumed to be linear. Consider the following figure.

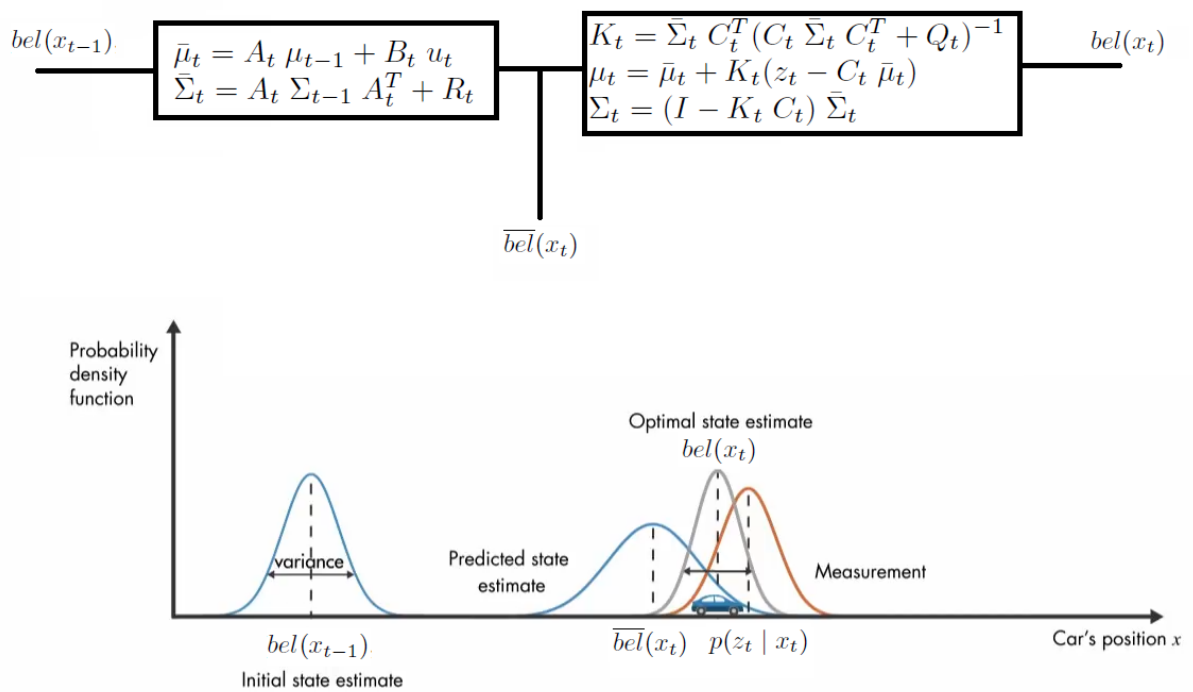


Figure 3.26: Simple example on applying Bayes Filter by using Kalman Filter Algorithm.

Each probability distribution function is modeled using the mean μ and covariance Σ (Variance in this case as we are considering one dimensional state only which is the distance in x), the algorithm states how we can translate from one state of the vehicle to the next state, the state here is the distance in the x direction only but it may contain more information such as the 3D position of the vehicle and the landmarks around it as mentioned in the state description section above.

In this example, let's say that car started at 0 m, and in the next state it reached 5 m which is unknown to us and we need to find.

The algorithm is divided into two parts the first is used to predict the next state using the motion model as it is known what are the previous motion commands of the vehicle such as distance request to the car motor u_t . This outputs the belief $\overline{bel(x_t)}$ which is a distribution of Let's say a mean of 3.6m which shows how this belief suffers from uncertainties because of the factors mentioned before. The second step is the usage of sensors to improve the predicted state estimation using the sensor model let's say that the sensors provide a probability distribution with the mean of 5.4m which again shows the uncertainties depicted in the estimation. The algorithm then uses the both motion and measurements distributions to output the optimal state estimation of the next state which would be also a probability distribution in which the mean is 5.01 m or something like that. The Kalman Gain K_t decides whether to rely more on the measurements or on the prediction based on motion.

Note: In the motion command we only stated the distance as a command i.e. requesting the motor controller to move the car for 5 m. This is because the Kalman Filter algorithm assumes that the system is linear in state transitioning, as the mean and covariance is updated using matrices A, B, and C in a system of linear equations. Accordingly, we can't include commands such as velocity and acceleration requests. In addition, in measurements we can only receive position data as again we can't estimate position from acceleration received from accelerometers for example. That is why we need to consider the Extended Kalman Filter.

Extended Kalman Filter

The Extended Kalman filter (EKF) calculates an approximation to the true belief. It represents this approximation by a Gaussian. In particular, the belief $bel(x_t)$ at time t is represented by a mean u_t and a covariance Σ_t . Thus, the EKF inherits from the Kalman filter the basic belief representation, but it differs in that this belief is only approximate, not exact as was the case in Kalman filters.

The representation here is no longer a linear system of matrices. That is, the linear predictions in Kalman filters are replaced by their nonlinear generalizations in EKFs. This is done by linearizing the nonlinear state transformation (nonlinear motion and nonlinear observation) at the current time step. Here the assumption is that the next state probability and the measurement probabilities are governed by nonlinear functions g and h , respectively. Where,

$$x_t = g(u_t, x_{t-1}) + \varepsilon_t, \quad z_t = h(x_t) + \delta_t$$

The main idea that constructs the EKF algorithm is linearization as we need to deal with non linear relations between variables. For instance, suppose we need to estimate the position from velocity or acceleration, this defines a nonlinear function. In KF, the task would be to predict the position from a linear combination of the velocity or acceleration command but we can't as this violates the assumption of the algorithm. By linearization, we can estimate a linear relation from the nonlinear function as we go step by step in time.

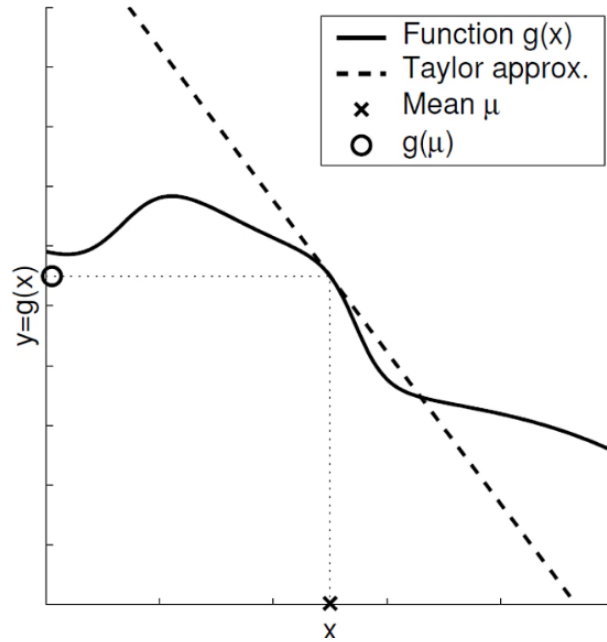


Figure 3.27: The linearization process of the Extended Kalman Filter algorithm.

EKF does function linearization by computing jacobians using Taylor Expansion and then prunes the expression to the linear part only in which it also uses a matrix representation to model both the prediction and the correction steps. The algorithm again updates the mean and covariance to update the probability distribution of the belief state.

EKFs use Jacobians G_t and H_t instead of the corresponding linear system matrices A_t , B_t , and C_t in Kalman filters. The Jacobian G_t corresponds to the matrices A_t and B_t , and the Jacobian H_t corresponds to C_t .

```

1:   Algorithm Extended Kalman filter( $\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$ ):
2:      $\bar{\mu}_t = g(u_t, \mu_{t-1})$ 
3:      $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$ 
4:      $K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$ 
5:      $\mu_t = \bar{\mu}_t + K_t (z_t - h(\bar{\mu}_t))$ 
6:      $\Sigma_t = (I - K_t H_t) \bar{\Sigma}_t$ 
7:     return  $\mu_t, \Sigma_t$ 

```

Table 3.5: Algorithm for Extended Kalman Filter

C. Simultaneous Localization and Mapping (SLAM)

□ Mobile Robot Localization

Mobile robot localization is the problem of determining the pose of a robot relative to a given map of the environment. It is often called position estimation or position tracking. Mobile robot localization is an instance of the general localization problem, which is the most basic perceptual problem in robotics. This is because nearly all robotics tasks require knowledge of the location of the robots and the objects that are being manipulated (although not necessarily within a global map).

Localization can be seen as a problem of coordinate transformation. Maps are described in a global coordinate system, which is independent of a robot's pose. Localization is the process of establishing correspondence between the map coordinate system and the robot's local coordinate system. Knowing this coordinate transformation enables the robot to express the location of objects of interests within its own coordinate frame—a necessary prerequisite for robot navigation. As the reader easily verifies, knowing the pose $x_t = (x, y, \Theta)^T$ of the robot is sufficient to determine this coordinate transformation, assuming that the pose is expressed in the same coordinate frame as the map.

Localization problems are characterized by the type of knowledge that is available initially and at run-time. We distinguish three types of localization problems with an increasing degree of difficulty.

Position tracking. Position tracking assumes that the initial robot pose is known. Localizing the robot can be achieved by accommodating the noise in robot motion. The effect of such noise is usually small. Hence, methods for position tracking often rely on the assumption that the pose error is small. The pose uncertainty is often approximated by a unimodal distribution (e.g., a Gaussian). The position tracking problem is a local problem, since the uncertainty is local and confined to a region near the robot's true pose.

Global localization. Here the initial pose of the robot is unknown. The robot is initially placed somewhere in its environment, but it lacks knowledge of where it is. Approaches to global localization cannot assume boundedness of the pose error. As we shall see later in this chapter, unimodal probability distributions are usually inappropriate. Global localization is more difficult than position tracking; in fact, it subsumes the position tracking problem.

Kidnapped robot problem. This problem is a variant of the global localization problem, but one that is even more difficult. During operation, the

robot can get kidnapped and teleported to some other location. The kidnapped robot problem is more difficult than the global localization problem, in that the robot might believe it knows where it is while it does not. In global localization, the robot knows that it doesn't know where it is. One might argue that robots are rarely kidnapped in practice. The practical importance of this problem, however, arises from the observation that most state-of-the-art localization algorithms cannot be guaranteed never to fail. The ability to recover from failures is essential for truly autonomous robots. Testing a localization algorithm by kidnapping it measures its ability to recover from global localization failures.

Another aspect that characterizes different localization problems pertains to the fact whether or not the localization algorithm controls the motion of the robot.

Passive localization. In passive approaches, the localization module only observes the robot operating. The robot is controlled through some other means, and the robot's motion is not aimed at facilitating localization. For example, the robot might move randomly or perform its everyday's tasks.

Active localization. Active localization algorithms control the robot so as to minimize the localization error and/or the costs arising from moving a poorly localized robots into a hazardous place.

Markov Localization

Probabilistic localization algorithms are variants of the Bayes filter. The straightforward application of Bayes filters to the localization problem is called Markov localization. This algorithm is derived from the algorithm Bayes filter. Notice that Markov localization also requires a map m as input. The map plays a role in the measurement model $p(z_t | x_t, m)$ (Line 4). It often, but not always, is incorporated in the motion model. Just like the Bayes filter, Markov localization transforms a probabilistic belief at time $t - 1$ into a belief at time t . Markov localization addresses the global localization problem, the position tracking problem, and the kidnapped robot problem in static environments.

EKF Robot Localization

The extended Kalman filter localization algorithm, or EKF localization, is a special case of Markov localization. EKF localization represents beliefs $bel(x_t)$ by their first and second moment, that is, the mean μ_t and the covariance Σ_t . The basic EKF algorithm was stated before.

□ Mapping

In fact, mapping is one of the core competencies of truly autonomous robots. Acquiring maps with mobile robots is a challenging problem for a number of reasons.

The hypothesis space, that is the space of all possible maps, is huge. Since maps are defined over a continuous space, the space of all maps has infinitely many dimensions. Even under discrete approximations, such as the grid approximation which shall be used in this chapter, maps can easily be described 10^5 or more variables. The sheer size of this high-dimensional space makes it challenging to calculate full posteriors over maps; hence, the Bayes filtering approach that worked well for localization is inapplicable to the problem of learning maps, at least in its naive form discussed thus far.

Learning maps is a “chicken-and-egg” problem, for which reason is often referred to as the simultaneous localization and mapping (SLAM) or concurrent mapping and localization problem. When the robot moves through its environment, it accumulates errors in odometry, making it gradually less certain as to where it is. Methods exist for determining the robot’s pose when a map is available, as we have seen in the previous chapter. Likewise, constructing a map when the robot’s poses are known is also relatively easy—a claim that will be substantiated by this chapter and subsequent chapters. In the absence of both an initial map and exact pose information, however, the robot has to do both; estimating the map and localizing itself relative to this map.

| Mapping | Localization | SLAM |
|-------------------------------------|----------------------------------|---|
| Building a map for the environment. | Estimating the robot’s location. | Building a map and localizing the robot simultaneously. |

Table 3.6: Differentiating between Mapping, Localization, and SLAM.

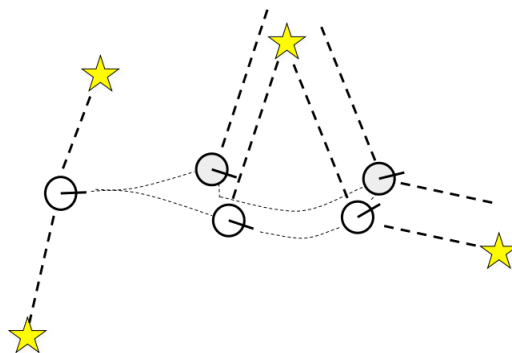


Figure 3.28: Localization example, finding the robot's pose from landmarks location and sensor measurements. (Having a prior knowledge about the environment map)

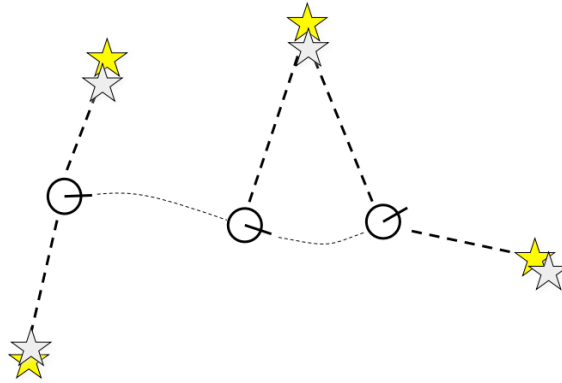


Figure 3.29: Mapping example, finding landmarks location from robot's pose and sensor measurements. (Having a prior knowledge about the position of the robot's position and orientation)

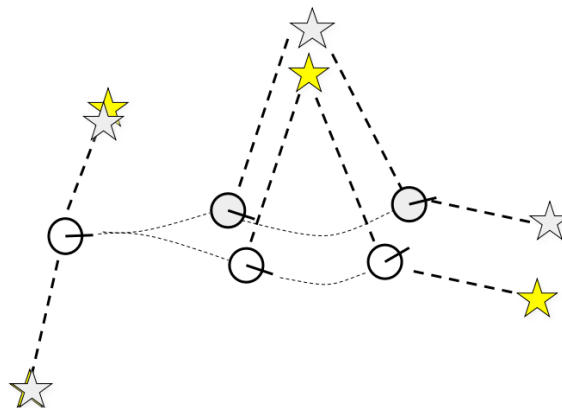


Figure 3.30: SLAM example, finding landmarks locations and robot's pose at the same time using sensor measurements. (Robot's pose and landmarks locations are evaluated with uncertainty attached in a consecutive manner)

3.4.2 Implemented approach for Mapping and Localization

Vehicle pose state estimation is an essential part of any mobile robotic application as it enables the robust operation of other system components. Several sensors are fused to estimate the pose and velocity of the ground vehicle. To take advantage of

redundancy in state estimation, the contribution of each sensor input to the overall estimated state has to be quantified in function of the sensor's accuracy and previous state knowledge. This is why we have chosen **The Extended Kalman Filter (EKF) localization** for the vehicle as it is the state-of-the-art estimator for fast, mildly nonlinear systems. For systems with white zero-mean additive gaussian noise corrupting the sensors and the motion model, it is a good approximation of the optimal solution. (i.e vehicle position and orientation in the map)

Localization is achieved by combining and fusing the readings from **visual odometry, wheel speed encoders** and **IMU** sensors. All the sensors will be fed to the Extended Kalman Filter (EKF) algorithm which estimates the current location of the car. The wheel speed sensor will send the current velocity of the car, the IMU sensor sends the lateral movement that is the steering angle of the car thus using these value state estimation will be done and thus the data can be filtered and a more accurate localization is processed.

For **mapping**, to create a map, we use **k-means clustering** to estimate the position of each cone in the track. This is performed by evaluating cone locations from repeated sampling and identifying a cone as being at the average location of a cluster of positions (Understanding k-means clustering in machine learning. Towards Data Science). Sampled cone locations can be determined to be part of a previously identified cone if it is found to be too close to it. Similarly, if a cluster becomes too large, it can be inferred that it is in fact a cluster containing position data for two cones instead of one, so we can then split it into two clusters for more accurate cone placement. In addition, if the cone position samples received are too sparse then the uncertainty of its location is increased which is expressed visually by increasing the size of the ellipse around the average values of the samples until the landmark is removed from the map, in the other hand, if the samples lies at the same position the uncertainty decreases and also the size of the ellipse, indicating how certain are we about the position of the landmark. This will be discussed later in section 4.6

3.5 Literature Survey on Motion Control

In this chapter we will talk about a very important topic in our project which is motion control. After the car knows the track layout and it can localize itself within the environment after finishing the first lap and mapping the complete track using Simultaneous localization and mapping algorithm. The main objective that we use motion control for is to compute trajectory of future control inputs with states of plant to optimize future behavior of plant output or in other words we can race the car

around a known track with maximum speed and how to make this need without oversteering in corners or slipping in acceleration.

3.5.1 Background on Motion control

In the last few years there has been a huge improvement and advancement in algorithms and controller techniques used in motion control on autonomous vehicles on track which we will talk about in this chapter. The most important controller technique used in motion control is Model Predictive Contouring Control (MPCC) or also known as Model Predictive Control (MPC). Using an MPC has proved to generalize to all scenarios where it keeps pushing the vehicle to its limits safely where the vehicle is about to reach its traction circle (Oversteering in Corners, Slipping in acceleration, Braking and propelling timing).

In this chapter we will talk about engineering and non-engineering backgrounds that we see important for complete understanding of part of our project which is motion control, discuss any pivotal knowledge to our project and give short literature review of the latest publications related to our project.

Controller Techniques

The motion control of an autonomous system, also known as execution competency, as we said before can be defined as the ability to race the car around a known track with maximum speed and how to make this need without oversteering in corners or slipping in acceleration or in other words it is the process of converting intentions into actions; its main purpose is to execute the planned intentions by providing necessary inputs to the hardware level that will generate the desired motions. Controllers map the interaction in the real world in terms of forces, and energy, on the other hand the cognitive navigation and planning algorithms in an autonomous system are usually concerned with the velocity and position of the vehicle with respect to its environment. Measurements inside the control system play an important role in controlling motion of the plant (Vehicle). Measurements inside the control system can be used to determine how well the system is behaving, and therefore the controller can react to reject disturbances and alter the dynamics of the system to the desired state. Models of the system can be used to describe the desired motion in greater detail, which is essential for satisfactory motion execution.

Let us talk in this part of the chapter about some controllers that might be used in controlling motion of our vehicle, know the weakness and strength points of each controller and which one is better in describing and dealing with our project.

We will talk about Proportional-Integral-Derivative (PID) and Model predictive control (MPC) and explain why we use Model predictive control instead of using Proportional-Integral-Derivative.

3.5.2 Comparative Study on Motion Control

Classical Control

Before talking about how to use Proportional-Integral-Derivative in controlling motion of vehicles we have to clarify some points about feedback control.

- Feedback control is the most common controller structure found in many applications.
- Feedback control uses the measured system response and actively compensates for any deviations from the desired behavior.
- Feedback control can reduce the negative effects of parameter changes, modelling errors, as well as unwanted disturbances.
- Feedback control can also modify the transient behavior of a system, as well as the effects of measurement noise.

Proportional-Integral-Derivative (PID) controller is the most common form of classical feedback control. The Proportional-Integral-Derivative controller is the most widely used controller in the process control industry. PID controllers are easy to understand and the concept of PID control is relatively simple. It requires no system model, and the control law is based on the error signal as:

where e is the error signal, K_p , K_i , and K_d are the proportional, integral, and derivative gains of the controller, respectively.

However, we can't use only feedback terms as the use of only feedback terms in a controller may suffer from several limitations. The first significant limitation of a feedback only controller is that it has delayed response to errors, as it only responds to errors as they occur. In addition to purely feedback controllers also suffer from the problem of coupled response, as the response to disturbances, modelling error, and measurement noise are all computed by the same mechanism. It is more logical then to manipulate the response to a reference independently from the response to errors.

$$u(t) = K_d \dot{e} + K_p e + K_i \int e(t)dt$$

where e is the error signal, K_p , K_i , and K_d are the proportional, integral, and derivative gains of the controller, respectively.

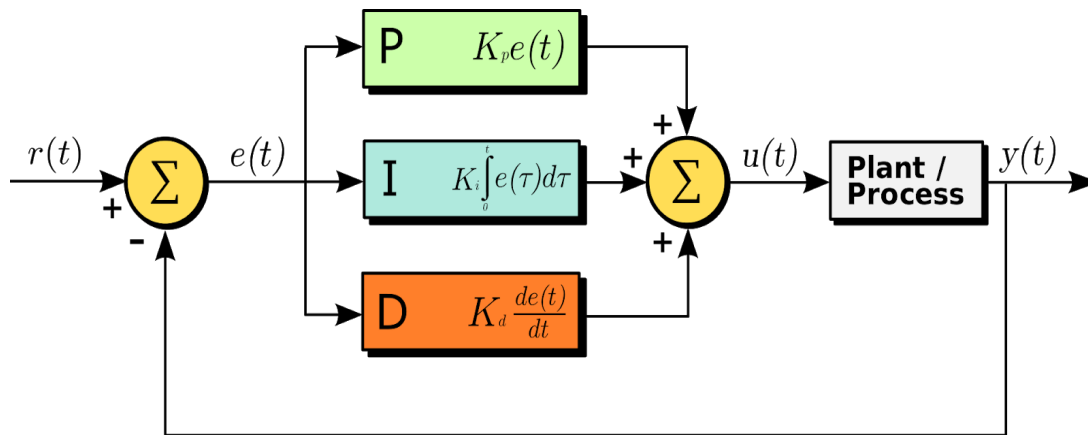


Figure 3.31: PID Controller

However, we can't use only feedback terms as the use of only feedback terms in a controller may suffer from several limitations. The first significant limitation of a feedback only controller is that it has delayed response to errors, as it only responds to errors as they occur. In addition to purely feedback controllers also suffer from the problem of coupled response, as the response to disturbances, modelling error, and measurement noise are all computed by the same mechanism. It is more logical then to manipulate the response to a reference independently from the response to errors.

Feedback Control

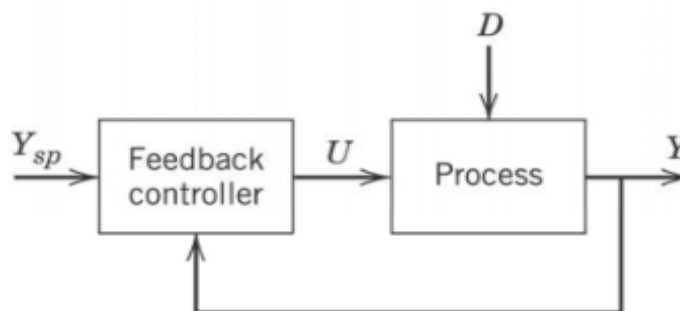


Figure 3.32: Feedback Control

We can add another degree of freedom to the controller. This is done by including a feedforward term to the controller. Feedback control and Feedforward control complete each other as the addition of a feedforward term in the controller can help to overcome the limitations of feedback control. The feedforward term is added to the control signal without considering any measurement of the controlled system. However, the feedforward term may involve the measurement of disturbances. A

model reference is used for the feedforward controller as designing a feedforward control requires a more complete understanding of the physical system.

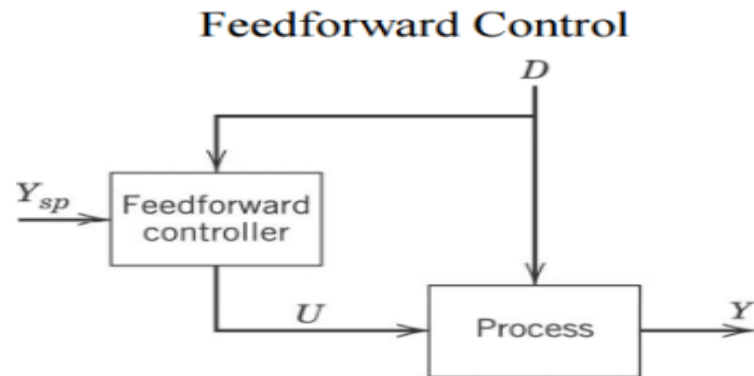


Figure 3.33: Feedforward Control

We can combine feedforward term and feedback term in the controller. The method of combining a feedforward and a feedback term in the controller is also known as two degree of freedom controller.

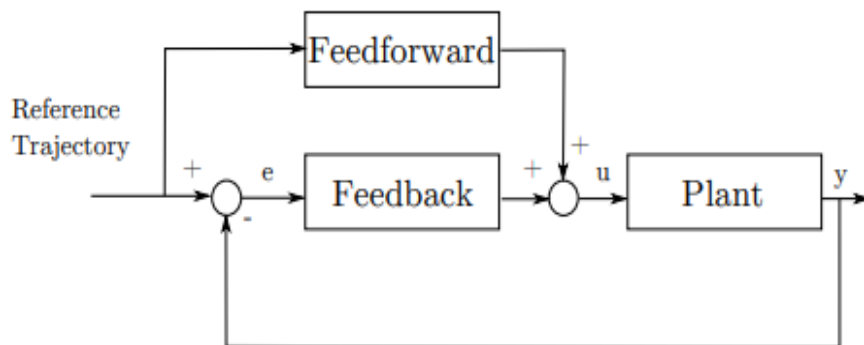


Figure 3.34: Feedforward and Feedback Control

To sum up the roles of feedforward and feedback control:

Feedback:

- Removes Unpredictable Errors and Disturbances
- Does not remove Predictable Errors and Disturbances
- Does not remove Errors and Disturbances Before They Happen
- Cannot remove Errors and Disturbances Before They Happen

- Does not require Model of a System
- Affects Stability of the System

On the other hand, Feedforward:

- Does not remove Unpredictable Errors and Disturbances
- Removes Predictable Errors and Disturbances
- Removes Errors and Disturbances Before They Happen
- Requires Model of a System
- Does not affect Stability of the System.

State space control, often referred to as modern control, is a technique that tries to control the entire vector of the system as a unit by examining the states of the system. a state-space representation is a mathematical model of a physical system as a set of input, output and state variables related by first-order differential equations or difference equations. State variables are variables whose values evolve through time in a way that depends on the values they have at any given time and also depends on the externally imposed values of input variables. Output variables' values depend on the values of the state variables.

A linear state space model can be written as:

$$\dot{X}(t) = A(t) * x(t) + B(t) * u(t) \quad y(t) = C(t) * x(t) + D(t) * u(t)$$

where $x(t)$ is the system state vector, $u(t)$ is the control input vector, and $y(t)$ is the output of the system.

The observations in an autonomous system are mostly nonlinear due to change of dynamics of the car as a result of changes of velocity, and therefore a linear model of the nonlinear system may have to be produced by first linearizing the state space equation of the system.

$$\dot{X}(t) = f(x(t), u(t)) \quad y(t) = h(x(t), u(t))$$

The feedback and feedforward (two degree of freedom controller) can also be applied to nonlinear systems. Feedforward is used to generate a reference trajectory, while the feedback is used to compensate for disturbances and errors.

The nonlinear system can be linearized about a reference trajectory to produce linearized error dynamics.

$$\delta \dot{X}(t) = A(t) * \delta x(t) + B(t) * \delta u(t) \quad \delta y(t) = C(t) * \delta x(t) + D(t) * \delta u(t)$$

where A, B, C, and D are the appropriate Jacobians. If there exists a trajectory generation process that can be designed to produce a reference input, such that reference input generates a feasible trajectory which satisfies the nonlinear system dynamics of the system, state space controllers can be configured to perform feedback compensation for the linearized error dynamics.

Model Predictive Control

After talking about classical control especially Proportional-Integral-Derivative (PID), now we will talk about another controller which we can say that it is one of the most important controllers nowadays and the widely used controller in autonomous vehicles. Autonomous systems need motion models for planning and prediction purposes. Models can also be used in control execution. A control approach which uses system modelling to optimize over a forward time horizon is commonly referred to in the literature as Model Predictive Control (MPC).

Model predictive control has seen tremendous and massive success in the industrial process control applications, due mainly to its simple concept and its ability to handle complicated process models with input constraints and nonlinearities and because it has multi inputs multi outputs (MIMO) and easy to be understood.

Model predictive control has been developed to integrate the performance of optimal control and the robustness of robust control. Typically the prediction is performed for a short time horizon called the prediction horizon, where the goal of the model predictive controller is to compute the optimal solution over this prediction horizon. The model, and thus the controller can be changed online to adapt to different conditions. The basic structure of MPC is shown in the figure below.

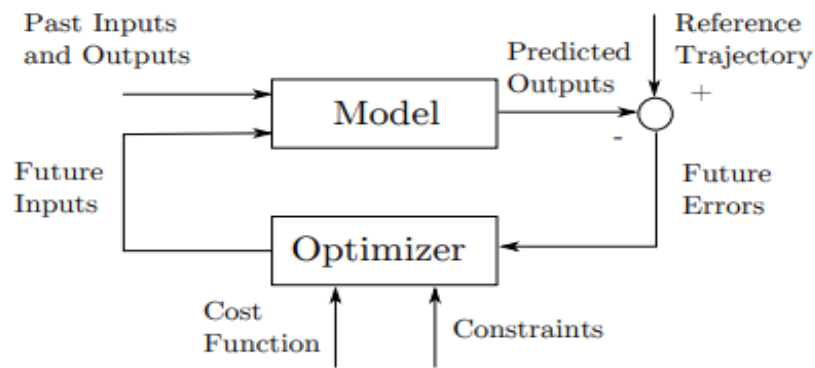


Figure 3.35: The basic structure of MPC

using an MPC has proved to generalize to all scenarios where it keeps pushing the vehicle to its limits safely where the vehicle is about to reach its traction circle (Oversteering in corners, Slipping in acceleration, Braking and propelling timing).

Model predictive control has several other attractive features, such as the simplicity of designing a multi variable feedback controller. It also allows for easy specification of system inputs, states, and outputs that must be enforced by the controller. MPC furthermore permits specification of an objective function to optimize the control effort. MPC can also address time delay, rejecting measured and unmeasured disturbances and taking advantage of previously stored information of expected future information. This feature can be very useful for repeated tasks, such as following a fixed path. MPC embodies both optimization and feedback adjustment, thus mimicking natural processes. Model predictive control has also been widely adapted to automotive applications.

The operations of the overall vehicle system must be optimal throughout the operating range. However, applying a model predictive controller in an automotive system meets different challenges than those faced in the process control industry.

In the process control industry, the sampling time is relatively longer, and the computing resources available are huge. The sampling period for processes in an automobile is a few milliseconds, and the amount of computing resources available is limited due to space constraints. Pushing the adoption of MPC into greater spread in the automotive industry is due to advances in processor speed and memory, as well as development of new algorithms.

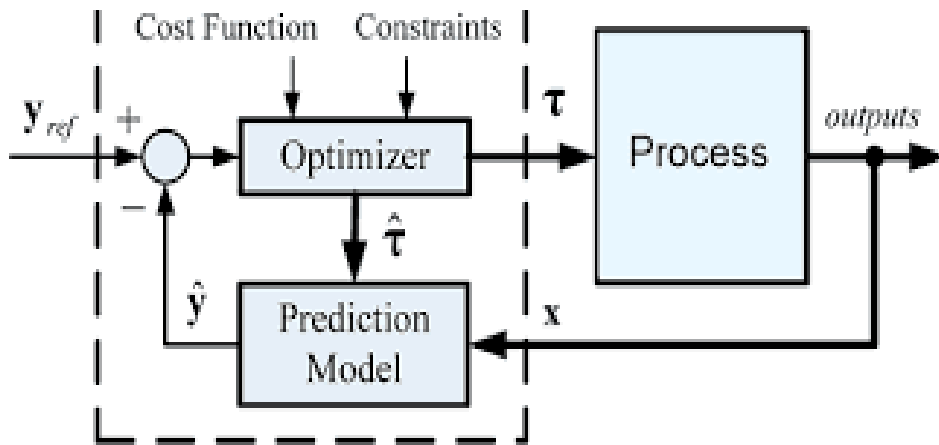


Figure 3.36: MPC Main Components

MPC has already been applied in several automotive control applications, including traction control, braking and steering, lane.

$$x(k + 1 | t) = f(x(k | t), u(k | t)) \quad y(k | t) = h(x(k | t), u(k | t))$$

These two equations are the discrete time model of the system dynamics with sampling period T_s where x is the system's state, u is the control input, and y is the system output. t is the discrete time index. The notation for a vector $v(h|t)$ denotes the value for v predicted at h time steps as referenced from time t , based on information up to t . The optimizer is the control input sequence $U(t) = (u(0|t), \dots, u(N - 1|t))$, where N is the prediction horizon. .

$$\text{minimize } U(t) \quad F(x(N|t)) + [N-1 \sum k = 0] L(x(k|t), y(k|t), u(k|t))$$

The cost function represents the performance objective that consists of the stage cost L and the terminal cost F . The constraints on the states and outputs are enforced along the horizons N_c and N_{cu} , respectively. The control horizon N_u is given as the number of optimized steps before the terminal control law is applied. At any control cycle t , the model predictive control strategy for the general problem operates as follows: system outputs are measured and the state $x(t)$ is estimated. This state estimation is acquired to initialize the above equation . Once the MPC optimization problem is solved and the optimal input sequence $U^*(t)$ is obtained, the first element of the optimal input sequence is then applied to the system. At the following cycle, the process is repeated using the newly acquired state estimate, thus applying the feedback.

Vehicle Model

It is challenging to drive a vehicle at its operational limits due to the highly nonlinear behavior in this operation range. So we have to model the dynamics of our vehicle as a dynamic bicycle model with nonlinear tire force laws where the car is modeled as one rigid body, and the symmetry of the car is used to reduce it to a bicycle. Only the in-plane motions are considered, i.e. the pitch and roll dynamics as well as load changes are neglected.

The advantages of using a bicycle model to model our car is that the model is able to match the performance of the car even in racing conditions, while at the same time being simple enough to allow the MPC problem to be solved in real-time.

Our vehicle model is derived under the following assumptions:

- the vehicle drives on a flat surface
- load transfer can be neglected
- combined slip can be neglected
- the longitudinal drive-train forces (from motor) act on the center of gravity

The Matrix below shows the equation of motion

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

where the car has a mass m and an inertia I_z , l_R and l_F represent the distance from the center of gravity to the rear and the front wheel respectively, $F_{R,y}$ and $F_{F,y}$ are the lateral tire forces of the rear/front wheel, F_x is the combined force produced by the drive-train and τ_{TV} the additional moment produced by the torque vectoring system.

X , Y , ϕ , v_x , v_y , r represent the state of the model where (X, Y) represent the position ϕ represent heading angle, (v_x, v_y) represent the longitudinal and lateral velocities and r represent the Yaw rate.

While $[\delta, D]$ represent the control inputs where δ represent the steering angle and D represent driving command which is like pedals of a driver where $D = 1$ corresponds to full throttle and $D = -1$ to full braking.

This model is the dynamic model where the rate of change of states is the function of inputs and states.

This model is used in high velocities and can not be used in slow velocities. The model is ill defined for slow velocities due to slip angles.

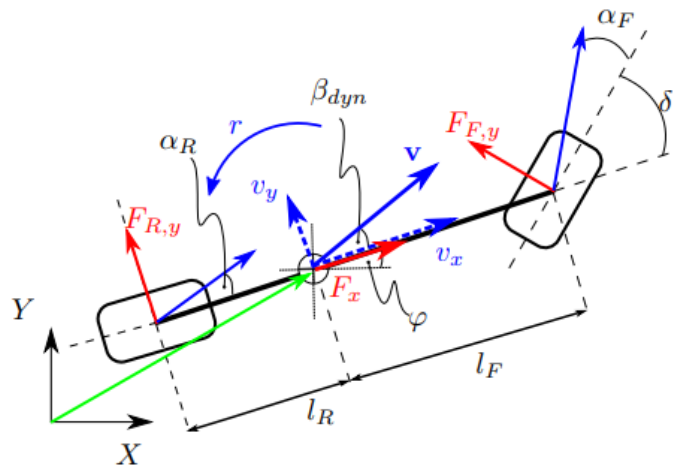


Figure 3.37: Tire Model

In the above figure the green colour represents the position vectors, while red colors represent forces where $F_{r,y}$ and $F_{f,y}$ represent the interaction between tires and track surface where r is the rear wheel and f is the front wheel and the third force which is F_x which represent the longitudinal force acting on the car which depends on input D (Driver command). In addition to the last color which is blue which represents the velocities.

Equations of forces can be shown below:

$$F_{r,y} = D_r \sin (C_r \arctan (B_r * \alpha_r))$$

$$F_{f,y} = D_f \sin (C_f \arctan (B_f * \alpha_f))$$

where

$$\alpha_R = \arctan ((v_y - l_r * r) / v_x) , \alpha_F = \arctan ((v_y + l_f * r) / v_x) - \delta$$

Where B,C,D are coefficients of the mode and α are rear and front angles.

In addition to third force which is longitudinal force which is equal

$$F_x = C_m * D - C_{r0} - C_{r2} * V_x^2.$$

Where $C_m * D$ is motor model C_{r0} is rolling resistance V_x^2 is drag.

As we said that the problem of the dynamical bicycle model is that the model is ill-defined for slow velocities due to the slip angles. However, slow velocities are important for race start and in sharp corners. For slow driving normally kinematic models are used which do not depend on slip angles. However, kinematic models are not suited for fast driving as they neglect the interaction of the tires and the ground. So, to get the best of both models within one formulation we propose a novel vehicle model combining a dynamic and a kinematic vehicle model.

we first formulate the kinematic model using the state of the dynamic model

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

$\cos(\delta)^2 \approx 1$ and $\tan(\delta) \approx \delta$. Is due to the assumption that δ is very small. The resulting model is the kinematic model or in other words the model that does not depend on forces. Where the rate of change of states does not only depend on states and inputs of the system but also on the rate of change of input.

Both kinematic and dynamic models are formulated using the same states which allows us to combine them. The resulting vehicle model is generated by linearly blended the two models. Below a certain velocity V_{min} we use purely the kinematic model while for velocities above certain velocities V_{max} we use purely the dynamic

model, and we combined the models and use the combined model in range of velocity between V_{max} and V_{min} .

3.5.3 Implemented Approach on motion control

From the above two controller techniques that we present. We can say that both Proportional-Integral-Derivative (PID) controller and Model Predictive Control (MPC) controller can be used to control motion of our vehicle. Each controller has its pros and cons or in another words there is a tradeoff between them.

In our project we use Model Predictive Control (MPC) to control motion of our autonomous vehicle because many teams all over the world use Model Predictive Control (MPC) instead of adaptive PID in there vehicles such as: Akademischer Motorsportverein Zürich (AMZ) which is one of the biggest formula students racing teams in the world, and many other teams.

Proportional-Integral-Derivative (PID) control has such advantages as a simple structure, good control effect and robust and easy implementation. Unfortunately, this method does not deal with parameter optimization and automatically adapts to the environment caused by the complexity of vehicle dynamics, uncertainty of the external environments and the non-holonomic constraint of the vehicle. But in solving the problem of trajectory tracking of unmanned vehicles, also the reference model of adaptive PID control based on the model reference is hard to ascertain because the motion model of the vehicle is influenced greatly by environments. The design of fuzzy adaptive PID control requires much priori knowledge. The vehicle finds it hard to obtain comprehensive priori knowledge when the vehicle travels in unknown environments. Adaptive PID control based on a neural network generally uses supervised learning to optimize the parameters, so it is also limited by some application conditions, for instance, the teacher signal of supervised learning is hard to obtain exactly. Although the design of adaptive PID control based on evolutionary algorithms requires less priori knowledge, it has the disadvantage of long computing times, i.e., not real time on line optimization.

That is why we use Model Predictive Control (MPC) to control the motion of our vehicle as The two main advantages of the proposed MPC is the direct consideration of the vehicle limits when computing the command and that the algorithm does not need any pre-determined logic, only the track layout, and the vehicle model.

In addition to it is flexible, open and intuitive formulation in time domain., solve problems with linear and non linear systems or variable and multivariable systems without change the controller formulation this is shown in the figure below which

shows us the a multivariable system with multi inputs and multi outputs (MIMO) , and it is the only controller that deals with constraints. This approach will be discussed in details in chapter 4.

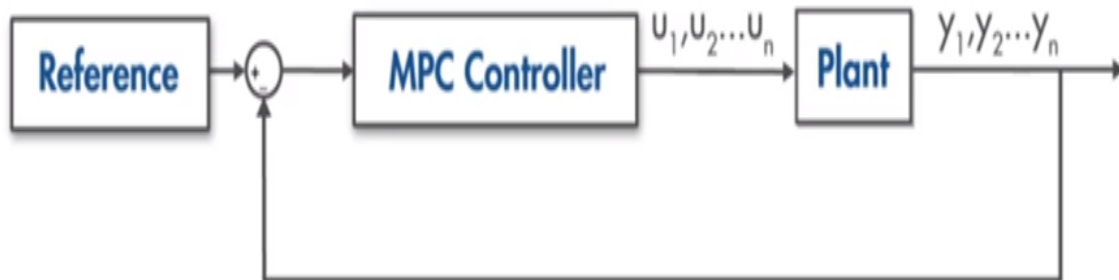


Figure 3.38: Model Predictive Controller

In the next chapter we will talk in some details about model predictive control (MPC) , how it works, its parameters, why it is important in autonomous vehicles and how it is used to optimize output to be similar to a reference path with maximum speed and without slipping or oversteering in corners.

3.6 Literature Survey on the Deployment of the modules and communication between them

In order to deploy and integrate all of our modules on a real car there must be a framework that will run all of the implemented modules and ensure that the communication between them is fast and accurate, so in this section we will discuss different types of frameworks that we faced and by the end of this section we will choose our framework which ensures the fast communication between the high performance driverless race car modules.

3.6.1 Background Information

Robot Operating System (ROS): ROS [3] is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms.

1) Message Passing

A communication system is often one of the first needs to arise when implementing a new robot application. ROS's built-in and well-tested messaging system saves you time by managing the details of communication between distributed nodes via the anonymous publish/subscribe mechanism. Another benefit of using a message passing system is that it forces you to implement clear interfaces between the nodes in your system, thereby improving encapsulation and promoting code reuse. The structure of these message interfaces is defined in the message IDL (Interface Description Language).

2) Recording and Playback of Messages

Because the publish/subscribe system is anonymous and asynchronous, the data can be easily captured and replayed without any changes to code. Say you have Task A that reads data from a sensor, and you are developing Task B that processes the data produced by Task A. ROS makes it easy to capture the data published by Task A to a file, and then republish that data from the file at a later time. The message-passing abstraction allows Task B to be agnostic with respect to the source of the data, which could be Task A or the log file. This is a powerful design pattern that can significantly reduce your development effort and promote flexibility and modularity in your system.

3) Standard Robot Messages

The asynchronous nature of publish/subscribe messaging works for many communication needs in robotics, but sometimes you want synchronous request/response interactions between processes. The ROS middleware provides this capability using services. Like topics, the data being sent between processes in a service call are defined with the same simple message IDL.

4) Robot Geometry Library

A common challenge in many robotics projects is keeping track of where different parts of the robot are with respect to each other. For example, if you want to combine data from a camera with data from a laser, you need to know where each sensor is, in some common frame of reference. This issue is especially important for humanoid

robots with many moving parts. We address this problem in ROS with the tf (transform) library, which will keep track of where everything is in your robot system.

Designed with efficiency in mind, the tf library has been used to manage coordinate transform data for robots with more than one hundred degrees of freedom and update rates of hundreds of Hertz. The tf library allows you to define both static transforms, such as a camera that is fixed to a mobile base, and dynamic transforms, such as a joint in a robot arm. You can transform sensor data between any pair of coordinate frames in the system. The tf library handles the fact that the producers and consumers of this information may be distributed across the network, and the fact that the information is updated at varying rates.

5) Diagnostics

ROS provides a standard way to produce, collect, and aggregate diagnostics about your robot so that, at a glance, you can quickly see the state of your robot and determine how to address issues as they arise.

6) Tools

One of the strongest features of ROS is the powerful development toolset. These tools support introspecting, debugging, plotting, and visualizing the state of the system being developed. The underlying publish/subscribe mechanism allows you to spontaneously introspect the data flowing through the system, making it easy to comprehend and debug issues as they occur. The ROS tools take advantage of this introspection capability through an extensive collection of graphical and command line utilities that simplify development and debugging.

a- Command-Line Tools:

Do you spend all of your time remotely logged into a robot? ROS can be used 100% without a GUI. All core functionality and introspection tools are accessible via one of our more than 45 command line tools. There are commands for launching groups of nodes; introspecting topics, services, and actions; recording and playing back data; and a host of other situations. If you prefer to use graphical tools, rviz and rqt provide similar (and extended) functionality.

b- rviz

Perhaps the most well-known tool in ROS, rviz provides general purpose, three-dimensional visualization of many sensor data types and any URDF-described robot.

rviz can visualize many of the common message types provided in ROS, such as laser scans, three-dimensional point clouds, and camera images. It also uses information from the tf library to show all of the sensor data in a common coordinate frame of your choice, together with a three-dimensional rendering of your robot. Visualizing all of your data in the same application not only looks impressive, but also allows you to quickly see what your robot sees, and identify problems such as sensor misalignments or robot model inaccuracies.

c- rqt

ROS provides rqt, a Qt-based framework for developing graphical interfaces for your robot. You can create custom interfaces by composing and configuring the extensive library of built-in rqt plugins into tabbed, split-screen, and other layouts. You can also introduce new interface components by writing your own rqt plugins .

The rqt_graph plugin provides introspection and visualization of a live ROS system, showing nodes and the connections between them, and allowing you to easily debug and understand your running system and how it is structured.

With the rqt_plot plugin, you can monitor encoders, voltages, or anything that can be represented as a number that varies over time. The rqt_plot plugin allows you to choose the plotting backend (e.g., matplotlib, Qwt, pyqtgraph) that best fits your needs.

For monitoring and using topics, you have the rqt_topic and rqt_publisher plugins. The former lets you monitor and introspect any number of topics being published within the system. The latter allows you to publish your own messages to any topic, facilitating ad hoc experimentation with your system.

For data logging and playback, ROS uses the bag format. Bag files can be created and accessed graphically via the rqt_bag plugin. This plugin can record data to bags, playback selected topics from a bag, and visualize the contents of a bag, including display of images and plotting of numerical values over time.

3.6.2 Implemented approach

By Implementing our pipeline from perception to mapping and localization of the vehicle it is found that most of pipeline modules needs to interact and send data to each other starting from camera sensing ending with producing the torque request and steering angle, so the pipeline's modules are run as nodes using Robot Operating System or ROS as the framework that eases handling of communication

and data messages across multiple systems as well as different nodes. Different modules communicate via messages, they receive data and output processed information. Another important aspect is that ROS is open-source and provides tools for visualization, monitoring and simulation, making it easy to integrate, test, diagnose and develop the complete software system. More information about the deployment of the pipeline on the ROS framework is described in chapter 4.

Chapter 4: System Design and Architecture

4.1. Overview and Assumptions

Autonomous racing presents a unique opportunity to test commonly-applied, safety-critical perception, and autonomy algorithms in extreme situations at the limit of vehicle handling and provides the opportunity to test safety-critical perception pipelines at their limit. Accurate and low-latency visual perception is applicable to many domains, from autonomous driving to augmented reality. We present challenges that require us to optimize known solutions to develop designs optimized for Formula Driverless. This section describes the practical challenges and solutions to applying state-of-the-art computer vision algorithms to build a low latency, high-accuracy perception system for a high performance formula student racing vehicle. The key components of the car modules include YOLOv3-based object detection, pose estimation and time synchronization on its dual stereo vision/monovision camera setup. We highlight modifications required to adapt perception CNNs to racing domains, improvements to loss functions used for pose estimation, and methodologies for sub-microsecond camera synchronization among other improvements. And describes an overview on the entire autonomous racing platform, covering all required software modules reaching from environment perception to mapping and control. Starting with the perception pipeline, the developed system works using a mono camera and stereo camera and the motion estimation subsystem fuses the measurements from different sensors using Extended Kalman Filter (EKF) to estimate the odometry of the vehicle and landmarks to use it in the localization and mapping algorithm.

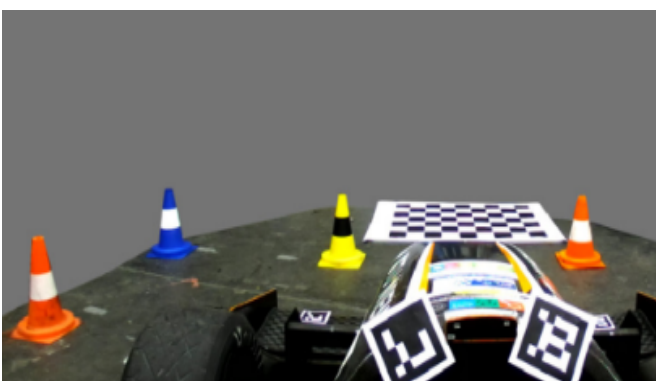


Figure 4.1: The restricted environment (Track and cones)

For the perception pipeline two-camera architecture were deployed with a **stereo** camera used for long-range detections and a monocular camera for short-range detections. As shown in Figure 4.2 the camera system will be mounted within the roll hoop of the vehicle, which satisfies the constraints from the competition rules and allows the cameras to be as high as possible to limit the effects of occlusions between landmarks.

The rationale for using the monocular camera for short-range rather than long-range detections is that for a reasonable mounting height, a landmark's 3D location on a relatively flat surface is a much stronger function of pixel space location for short-range objects than long-range objects. This relieves some of the challenges for estimating landmark pose from a monocular camera. On the other hand, however, estimating 3D pose of an object using a single measurement, i.e. a single image from a monocular camera is an ill-posed problem. This is primarily due to ambiguity in the scale of the scene arising from limited information of the surroundings. This ill-posed problem of extracting pose information can be solved if a priori information about the 3D object in the scene is available. The 3D priors about an object, in addition to 2D information obtained from an image can be together leveraged to extract 3D pose of this object captured in any arbitrary image of the scene. On a real-time system, such as an autonomous race-car, it becomes even more crucial to detect and estimate multiple object positions extremely efficiently, with as little latency and data overhead (in terms of transport and processing) as possible.

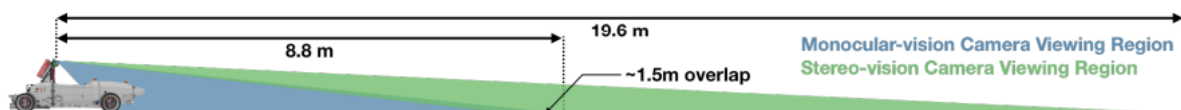


Figure 4.2: Monocular and Stereo camera ranges

In the two vision pipelines (Mono and Stereo) the landmarks (Track Cones) are detected using 2D customized, trained, and optimized detection algorithm which is YOLOV3 the detection algorithm not only fastly detects the 2D position of the cones but also used to detect the cones colors for mapping purposes, then the detected cones enter to the implemented keypoints extraction Residual neural network to extract the seven points for each cone to use it in the PNP algorithm which is used to estimate the 3D position for each cone relatively to the car position. We believe that it is possible to estimate the 3D position from a single frame mono camera but in our case we assume that this perception pipeline

works in a restricted environment in our case the restricted environment is a track with a prior knowledge of its cones 3D models.

For localization and mapping algorithms, we utilize a K-means-based Global Mapping algorithm to construct the global map of the track using the 3D Cones' local positions which are estimated in the perception pipeline and fuse sensors' readings to build a relation between the 3D local positions of cones with the odometry of the car to construct the 3D global map of the track.

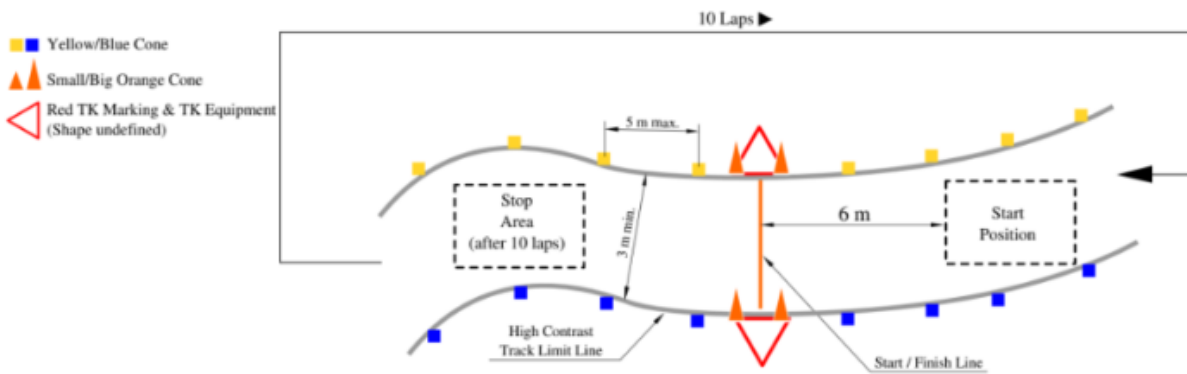


Figure 4.3 : The track layout of a trackdrive discipline (FSG, 2018). Blue and yellow cones mark the track boundaries

Lastly, we present a control framework that directly minimizes lap time while obeying the vehicle's traction limits and track boundary constraints. Figure 4.3 shows the track layout of a trackdrive discipline. Blue and yellow cones mark the track boundaries. It is important to note that the track is completely unknown before starting the race which increases the challenge considerably. In addition, all computations and sensing are required to happen on-board.

This pipeline is integrated over a very powerful operating system which is the robot operating system (ROS) and each stage in the pipeline is represented with a node in ROS to easily communicate with each other and transfer the data in an accurate and fast way.

4.2. System Architecture

In this section, the architecture of the autonomous software system is discussed that is used to drive the vehicle in real time. Referring to the block diagram figure below.

Sensors

The input sensors (blue boxes) are attached to a well known location in the vehicle in order to be relatively translated during operation.

Monocular Camera

A sensor selected under the design constraints in order to provide a stream of 2D image frames which are fed to the monocular pipeline to extract the near objects in scene (cones) 3D positions with respect to the vehicle.

Stereo Camera

A sensor selected under the design constraints in order to provide two streams of 2D image frames from two cameras which are placed with very specific distance between each other which are fed to the stereo camera pipeline to generate a depth map and extract the far objects in scene (cones) 3D positions with respect to the vehicle.

Inertial Measurement Unit (IMU)

A sensor selected under the design constraints in order to provide acceleration in x, y, z and orientation about them in order to be used by the vehicle localization extended kalman filter algorithm (EKF) to evaluate the position of the vehicle at each time frame.

Wheels Encoders

A sensor selected under the design constraints in order to provide the velocity of the vehicle's wheels which is used to estimate the velocity vector of the vehicle (i.e. Longitudinal and Lateral velocities) in order to be used by the vehicle localization extended kalman filter algorithm (EKF) to evaluate the position of the vehicle at each time frame.

Pipelines

The diagram also shows the pipelines used in the vehicle operating system (dashed lines) where inside each pipeline there exist the modules that compose the pipeline (red boxes).

Monocular Pipeline

The pipeline responsible for getting the 3D position of near cones from a single frame, it consists of three modules that are responsible for extracting the 3D positions of the cones from the scene information provided by the camera, in addition it uses the 3D model of the cone and the intrinsic parameters of the camera.

- 1) **2D Space Localization Cone Detection** used to find the cones in the image frame by a bounding box and recognize their colors.
- 2) **Keypoints Extraction DNN** a developed DNN used to estimate specific set of 7 keypoints for each cone coming in the shape of image batches
- 3) **3D Space Localization PNP algorithm** uses the 7 points detected by the DNN for each batch and the corresponding points in the 3D model

provided to the algorithm to extract the 6 DOF of the cone with respect to the camera in the vehicle.

Stereo Pipeline

The pipeline responsible for acquiring 3D position of far cones from 2 image frames specifically distant from each other, it is divided into two processing modules according to the input frame as the stereo camera inputs two frames left and right. The pipeline includes 4 modules.

- 1) **2D Space Localization Cone Detection** used to find the cones in the left image frame by a bounding box and recognize their colors.
- 2) **Feature Extraction** used to extract the features of the left frame image batches that represent cones in order to be shifted to match that of the right frame.
- 3) **Bounding Box Propagation** uses the features extracted from the left frame boxes to extract the location of the corresponding bounding box in the right frame and then propagate the right frame bounding boxes features.
- 4) **SIFT Feature Matching and Triangulation** uses the received left and right features to match and triangulate between them using the known distance between the cameras to evaluate the depth map of the received cones which are used to evaluate the 3D position of the cones.

Localization and Mapping Pipeline is responsible for fusing the input stream of cones 3D position from the monocular and stereo pipeline while receiving vehicle odometry inputs to produce a 3D World Map Construction that describes the track and the cones position. The pipeline consists of 4 modules.

- 1) **K-means Cone Mapping Fusion** used to fuse the two cone estimates from monocular and stereo pipelines by using sampled data in a k-means clustering scheme.
- 2) **Visual Tracking Pose Estimation** continuous mapping between frames to give estimates about the position of the vehicle.
- 3) **EKF Robot Localization** fuses the IMU, wheels encoders, and the position estimation from the visual tracking module to produce the final position of the vehicle in the global map.
- 4) **Global Mapping** uses the final fused reactive cones mapping from the k-means Mapping and the final position from the EKF Robot Localization to produce 3D map of the track and cones.

Path Planning and Motion Control Pipeline is responsible for processing the received track map and generates the trajectory that the vehicle should follow and uses a model predictive control algorithm (MPC) to generate the actuation commands which are sent to the vehicle control unit.

Actuators

Steering Angle is the instant angle the steering wheel should be for the vehicle to follow the generated trajectory.

Longitudinal Velocity is the instant speed command sent to the motor controller to run the vehicle on the generated trajectory as fast as possible.

4.2.1. Block Diagram

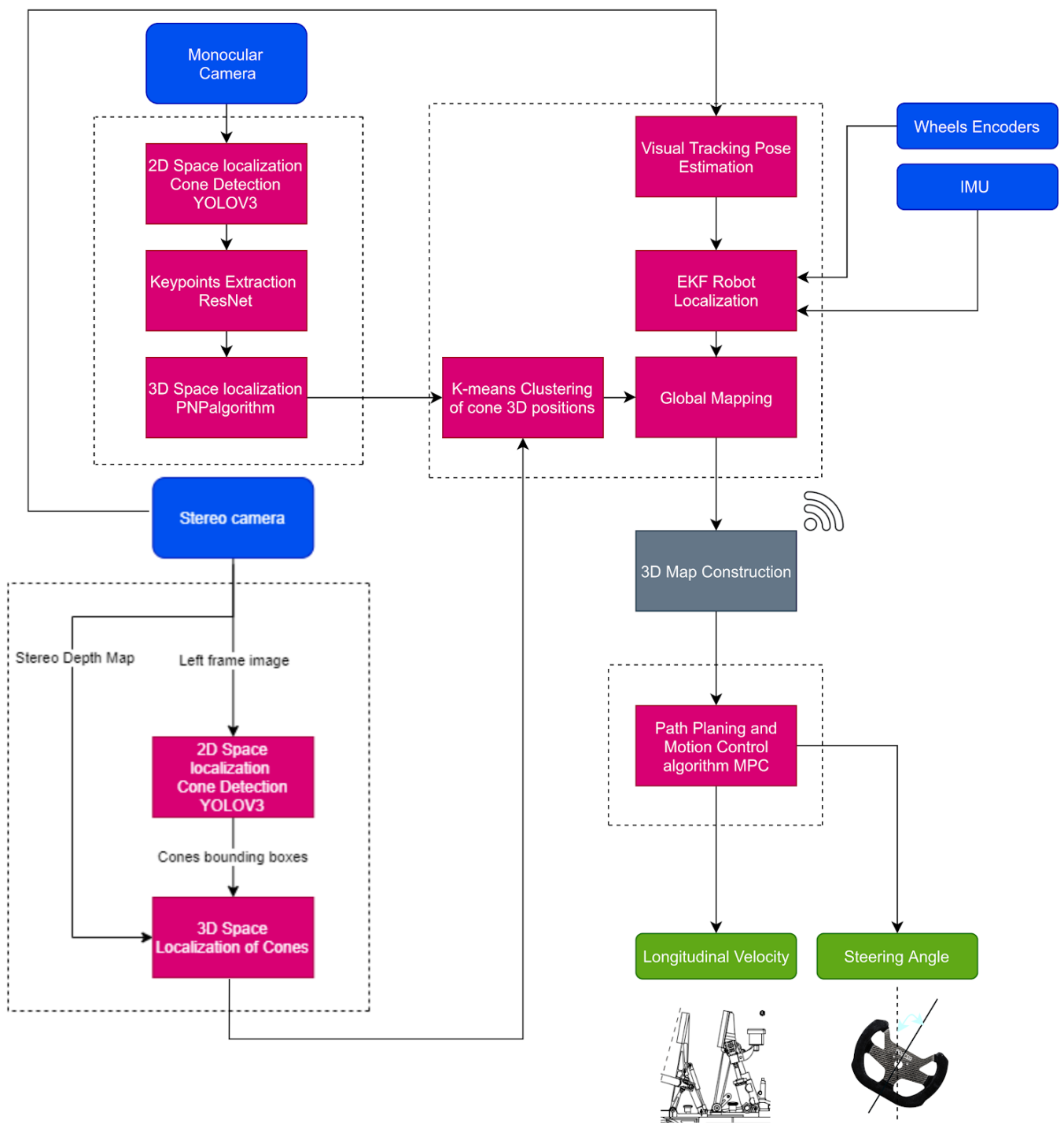


Figure4.4: Overall system block diagram

4.3 2D Space Localization / Cone Position and Color Detection

4.3.1. Functional Description

The object detector should be efficient in that it is fast, requires lesser memory and is still decently accurate in its detections to cope with our high performance perception pipeline of the racing driverless vehicle. We choose and customize YOLOv3 for the purpose of detecting different colored cones. Thresholds for it are chosen such that false positives, incorrect detections and misclassification are avoided at any cost; even if that translates to not being able to detect all cones in a given image. We customized YOLOv3 by reducing the number of classes that it detects, as “Our driverless high performance formula race car” does not really care about detecting cats, dogs, airplanes or bikes to name a few but needs to distinguish and detect ‘yellow’, ‘blue’ and ‘orange’ cones that provide information about the track. We reduce the classes of the pre-trained YOLOv3 to 3 classes which are ‘yellow’, ‘blue’ and ‘orange’ cones. More than 50k formula student track cones were collected and manually labelled by drawing boundary boxes around the cones.

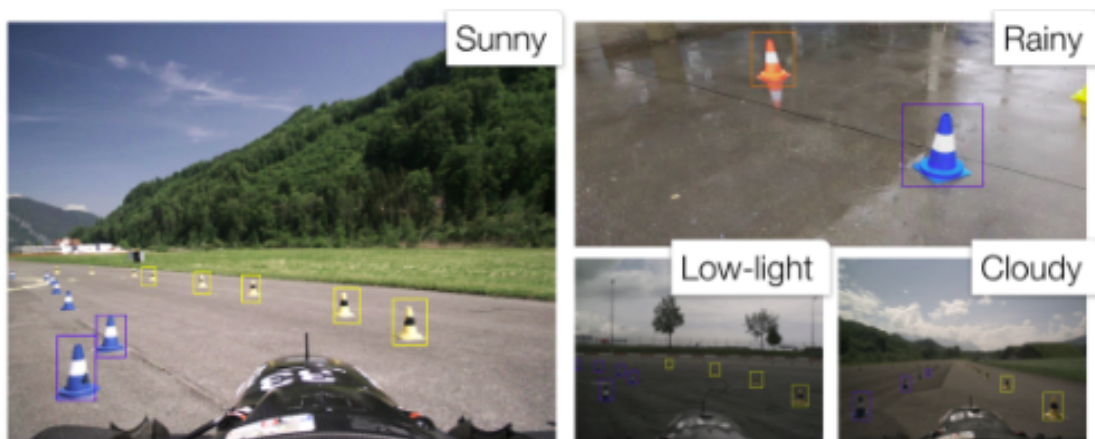


Figure 4.5: Exemplary images from varying lighting and weather conditions

We then developed a fast and accurate labeling tool which labels the color of the boundary boxes in each frame using the keyboard and produces the CSV file which contains the bounding box and the color of each cone in each frame to use it in the training of the object detection module and producing the CFG files and the CNN weights of the model. Figure showing robust performance from the object detection to detect ‘yellow’, ‘blue’ and ‘orange’ cones. It is imperative to have a robust cone

detection as it is the first module in the pipeline and directly affects performance of the modules that follow because they depend on it, eventually the final output as well.

These images in diverse weather and lighting conditions which are in Figure 4.5 show the robustness of this particular object detection module, customized to detect colored cones.

4.3.2. Modular Decomposition

Diving into this module and to divide the module into smaller fine ones the work in this module is divided according to the next five submodules:

1) Dataset Collection

To ensure our concept of a fast and accurate perception pipeline of our Driverless racing vehicle, we collected a dataset for the perception neural network that can generalize across different domains (weather, lighting, scenery) to produce more robust detections and in turn localize landmarks with higher accuracy. To ensure this with our networks, training data was collected on multiple image sensors and lenses from various perspectives in different settings.

The dataset for the object detection module is more than 50k formula student track cones. The data for training and testing was collected by contributing 600 images to Formula Student Objects in Context (FSOCO) with the concept of Sharing is caring. This dataset lives from the contribution of all formula student teams who want to access a dataset of more than **50k cones** of formula student track cones. The data is divided as a sequence of on board track drive video frames which collected in the whole formula student competitions and some of the data which is shared by the teams like us was in a park or a college it is not matter where the images were captured but it matters that the cones of the formula student competition are the same in all images in the dataset. Here are samples from the dataset of the track cones in Figure 4.6



Figure 4.6 Samples from the object detection dataset of the track cones

2- Self Developed Labeling Tool

Bounding Box Labeling Tool:

Eight thousand images of on-track footage containing about 50 thousand cones were manually labeled using a self-developed labeling tool that exploits similar structure between consecutive frames in a video sequence. Objects of interest are easily labeled by drawing rectangle through click-drag click using a mouse. To speed up the tedious labeling procedure, the tool tracks annotated rectangles over frames and propagates them to prevent re-labeling for future frames, treating propagated bounding boxes as annotations. After some frames, the trackers may lose their objects due to fast movement or change in view points. At such points, one can refresh and re-label again. Since the annotations for cones are long and thin rectangular bounding boxes, we exploit such prior information by re-calculating the anchor boxes used by YOLOv3. This is done by performing k-means clustering on the aspect-ratio of the rectangle annotations in the dataset and improves the object detector's performance.

2) Cone Color Labeling Tool:

An annotation tool was developed to label the cones into one of three classes; yellow, blue, or orange. The tool begins by reading the boundary box coordinates for each cone in each image, shows the boundary box around the corresponding cone, and by clicking one of three buttons (A for blue, D for yellow, S for orange) a cone can

be classified into one of the three classes. The boundary boxes and the classified classes are then saved together in the correct format to be used in training the YOLOv3 network.

3) YoloV3 customization and optimization

For accurate 2D localization, the synchronized images are batched together and passed through a full YOLOv3 [4] neural network using the TensorRT inference framework. Weights were calibrated to int-8 precision using 50k pictures from the network training dataset resulting in inference speeds 10x faster than a similar PyTorch implementation.

Non-maximal suppression thresholds were set to 10% to filter out occluded landmarks as these were found to be problematic for depth estimation later in the pipeline.

A perception neural network that can generalize across different domains (weather, lighting, scenery) will produce more robust detections and in turn localize landmarks with higher accuracy. To ensure this with our networks, training data was collected on multiple image sensors and lenses from various perspectives in different settings.

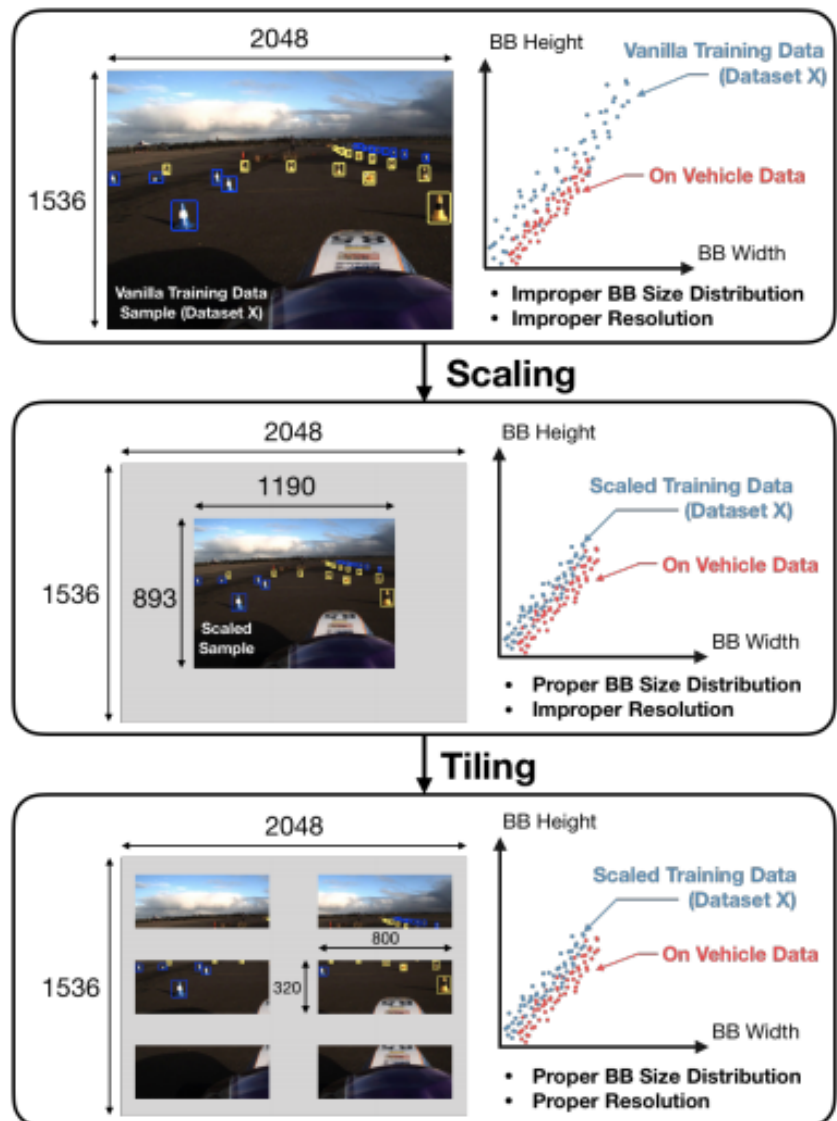


Figure 4.7 CV-C Data Loader Pre-processing Stages

A drawback of this process is that the distribution of landmark bounding box (BB) sizes (in pixels) in the training set no longer was representative of what would be seen by the network in the wild. To mitigate this, each set of training images from a specific sensor/lens/perspective combination was uniformly rescaled such that their

landmark size distributions matched that of the camera system on the vehicle. Each training image was then padded if too small or split up into multiple images if too large. This process is illustrated in Figure 4.7.

Since YOLOv3 makes width and height predictions of detections by resizing predefined bounding boxes, k-means clustering was done on the post-scaled training data in height and width space to give the network strong priors. An additional modification made during the training process was tuning the hyperparameters in front of each of the terms in the loss function.

Each bounding box prediction consisted of estimates at x, y, width, height, class (foreground or background), and confidence, which are penalized differently during training as follows:

$$L_{total} = \gamma_{cls}L_{cls-ce} + \gamma_{BG}L_{BG-bce} + \gamma_{FG}L_{FG-bce} + \gamma_{xy}(L_{x-mse} + L_{y-mse}) + \gamma_{wh}(L_{w-mse} + L_{h-mse})$$

A distributed Bayesian hyperparameter search for the five coefficients resulted in significant gains in precision when weighting the foreground loss two-orders of magnitude greater than the background loss.

The converged upon values from the optimization process are $\gamma_{BG}=25.41$, $\gamma_{FG}=0.09$, $\gamma_{XY}=1.92$ and $\gamma_{WH}=1.33$.

Further gains were obtained by switching from the SGD optimizer in the initial implementation to Adam [5]. The compounding benefits for each of these changes are shown in Figure 4.8

as precision-recall curves. The resulting final mAP was 85.1% with 87.2% recall and 86.8% precision.

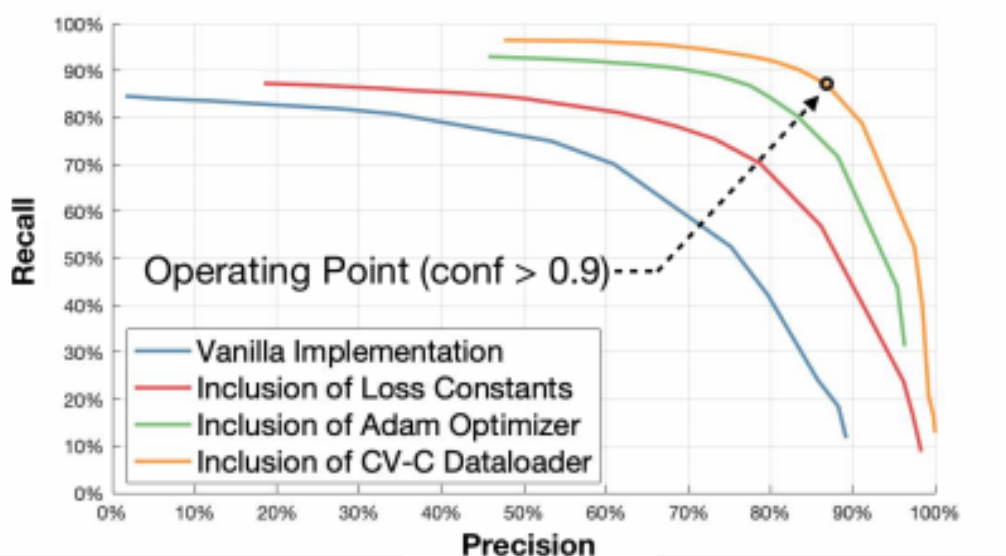


Figure 4.8: The compounding benefits for the changes in YOLOv3

4.3.3. Design Constraints

Similar to most perception systems for autonomous racing, the goal of our perception system is to accurately localize environment landmarks (traffic cones) that demarcate the racetrack. The track is delineated by blue cones on the left, yellow cones on the right, and orange cones at the start and finish. Downstream mapping and planning systems use these landmarks to create and update the track map with a sample illustrated in Figure 4.9. Our perception system adheres to regulations set by Formula Student Driverless.

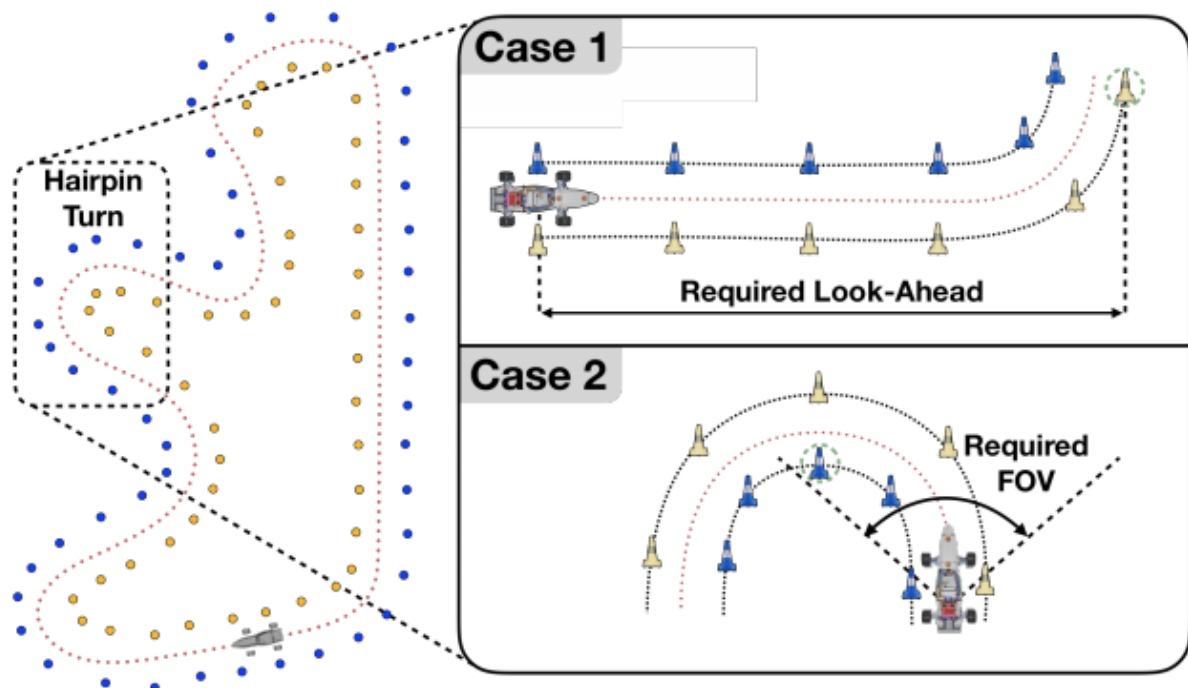


Figure 4.9: Sample of landmarks which are used to create and update the track map

Our perception system was designed to meet four high-level requirements:

- 1) Mapping Accuracy: accuracy of landmark localization.
- 2) Latency: total time between a landmark coming into view of the perception system to the time at which it is localized.
- 3) Look-ahead Distance: longest straight-line distance in which accuracy is maintained.
- 4) Horizontal Field-of-View (FOV): arc of visibility in front of the car, related to visibility through a hairpin.

To guide design choices, we derived quantitative targets for each requirement. Mapping accuracy was driven by the error tolerances of the mapping and motion control algorithms that will be illustrated in the next modules. This dictated maximum tolerable localization error of $<0.5\text{m}$ at the maximum look-ahead distance.

For latency the object detector YOLOV3 was optimized to meet our requirement of designing a high performance perception pipeline of a racing vehicle and the communication between the object detection modules itself and with the other systems constructed over ROS which minimize the communication losses and improves the speed of the processes communication time.

Horizontal FOV is lower-bounded by unmapped hairpin turns Figure 4.9. In such turns, the system must perceive landmarks on the inside apex of a hairpin turn from the start of the turn in order to plan an optimal trajectory. Given legal track dimensions, this results in a minimum FOV of 101° .

Look-ahead requirements depend on full-stack-latency, car dynamics, and camera properties. In particular, minimum landmark size, the number of pixels required to detect a landmark, plays a crucial role in determining look-ahead. To understand this more clearly, the camera and kinematics models generate a characterization of the relationship between the minimal landmark size, camera focal length, and the physical size of pixels on the camera sensor.

We assume a conservative minimum detectable landmark size of 20 pixels (consistent with state-of-art work [4,6]); this is represented by the dashed line in Figure 4.10. The optimal solution is a system that:

- (1) stays close but above this line,
- (2) maximizes pixel size (i.e., maximizing light capture), and
- (3) minimizes focal length (i.e., maximizing FOV). Figure 4.10 illustrates the trade-off between these properties. Based on these

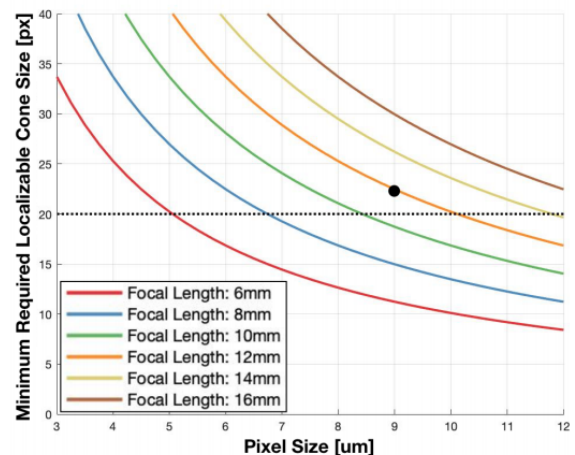


Figure 4.10 Cone size vs focal length

models and fixed values of prior constraints, we derived a look-ahead requirement of 19.6m. As there is an inherent trade-off between look-ahead distance and FOV in camera-based perception systems, the two cases outlined in Figure 4.9 have mutually exclusive requirements: wide FOV while maintaining long look-ahead. To meet this design requirement for the redundant perception pipeline with stereo and monovision the chosen on board hardware of our autonomous kit is specified in the Appendix.

4.4. Key Points extraction Residual Network And 3D localization of cones -PnP algorithm

4.4.1. Functional Description

Till now we have the cone batches in each frame Figure 4.12 so in this module we will take another step to solve the problem of estimating the 3D position of an object (the track cone) using a single measurement, i.e. a single image/frame of a monocular camera. Estimating the 3D pose of an object using a single measurement is an ill-posed problem. This is primarily due to ambiguity in the scale of the scene arising from limited information of the surroundings. This ill-posed problem of extracting pose information can be solved if a priori information about the 3D object in the scene is available. The 3D priors about an object, in addition to 2D information obtained from an image can be together leveraged to extract 3D pose of this object captured in any arbitrary image of the scene. The priori information about the 3D object in our case is the 3D model of the track cone as shown in Figure 4.11 we have a prior knowledge of the formula student standard track cones' 3D model and we defined a 7 key points on the 3D model in a defined locations as in Figure 4.11 and for the detected cone batches from the previous module

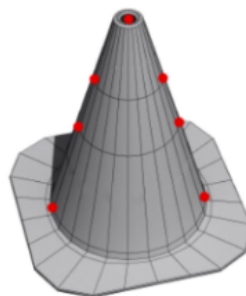


Figure 4.11: Cone's 3D Model with the 7 key points

We detect the 7 key points for each cone batch at the same location of the points which are in the 3D mode as in Figure 4.12 I of the cone, so we developed an accurate and fast Deep Neural Network (DNN) to detect the seven points for each cone batch. The 2D information which we talked about is obtained from the output of the keypoint extraction DNN, and it will be used in the next module in the pipeline with the prior knowledge of the 3D model of the track cone to estimate the 3D position of each cone batch. The “keypoint regression” scheme that exploits prior information about the object’s shape and size to regress and find specific feature points on the image. Further, in addition to the “keypoint regression” which provides 2D information, a priori 3D information about the object is used to match 2D-3D correspondences which will be explained in the next module. To extract depth estimates from a single camera a residual neural network is implemented and

trained with 3.2k images based off of work by [7] is run to detect seven key points on each batch detected with the object detection module for use in a Perspective-n-Point (PnP) algorithm which is described in the next module.



Figure 4.12: Cone batches (left) and Cone batches with key points detected (right)

4.4.2. Modular Decomposition

As explained in the previous section, with the help of an object detector, one can find multiple objects of interest in a single image. The question here is to go from objects on the image to their positions in 3D. This in itself is not solvable from a single view of the scene, because of ambiguities due to scale. However, since there is prior information about the 3D shape, size and geometry of the cone, one has hope to recover 3D pose from a single measurement as will be discussed in the next module, but now we are interested in extracting the key points from the landmarks (cones) which were detected in the object detection module so, in the next submodules we will describe the process of extracting the key points and the detailed architecture of the developed DNN and the modifications which we made to it.

1) Design and architecture

In the context of classical computer vision, there are mainly three kinds of features. The least informative ones are the “flat features” which in the vicinity are not distinguishable at all, for instance the patch on a plain, flat wall is one such example. “Edges” are a little more interesting as they have a gradient in a particular direction (crossing-over the edge). However, if one moves in a direction perpendicular to this gradient one is unable to distinguish; this is also known as the aperture problem. By far, the most interesting features are the “corners”. They have change in gradient in two major directions and are quite distinguishable from areas in the vicinity, making them unique and fascinating. With this in mind, we design a convolutional neural network (CNN) inspired by finding “corner” like points given a patch of the image. The primary difference between this scheme and any other feature extraction process is that this is very specific as compared to commonly used techniques. This does not mean that it cannot be used for other objects. This “keypoint Extraction” scheme works for a specific object but can be easily extended to different types of objects. In our case, we want to find positions of very specific points on the image that correspond to 3D counterparts whose locations can be measured in 3D from an arbitrary world frame F_w .

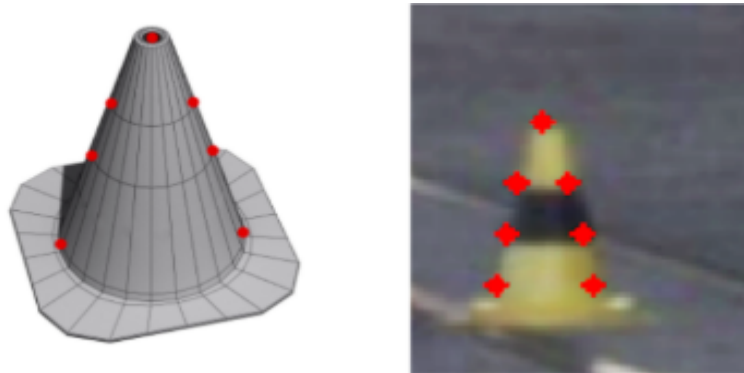


Figure 4.13 :3D model of the cone and a representative sub-image patch with the image of the cone. The red markers correspond to the 7 specific “key points” the “keypoint network” regresses to given an image patch with a cone in it.

As depicted in Figure 4.13 , the key points on the 3D and its corresponding 2D image are very specific. There are two primary reasons to have these key points at those places. First, the key-point regressor locates the position of 7 very specific features that are also visually distinct and can be considered as “corners” such as points between the merging of distinct textures and points at the interface of the foreground and the background. Second, and more importantly, these 7 points are relatively easy to measure in 3D from a fixed world frame F_w . For convenience F_w is chosen to be the base of the 3D cone, enabling easy measurement of 3D position of these 7 points in this world frame, F_w . The 7 keypoints are the apex of the cone, two points (one on either side) at the base of the cone, 4 points where the center stripe, background and upper or lower stripes meet. The customized CNN inspired from “corner” features takes as input a $80 \times 80 \times 3$ sub-image patch which presumably contains a cone, as detected by the object detector in the previous module and maps

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times padding[0] - dilation[0] \times (kernel_size[0] - 1) - 1}{stride[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times padding[1] - dilation[1] \times (kernel_size[1] - 1) - 1}{stride[1]} + 1 \right\rfloor$$

it to R14.

The input dimensions are chosen as 80×80 spatially, as this was the average size of bounding boxes detected. The output vector of R14 are the (x, y) coordinates of the 7 key points relative to the patch. The architecture of the convolutional neural network consists of basic residual blocks inspired from ResNet [8]. The reasoning here is that since the convolution operation reduces spatial dimensions, we apply ‘same’ convolutions that result from a 3×3 kernel with padding=1 and stride=1 via a residual block. As analyzed in [9], with more layers, the tensor volume has more channels and fewer spatial dimensions, implying the tensors contain more generic, global information than specific, local information. Since, we eventually care about

location of keypoints which is extremely specific and local. Using such an architecture prevents loss of spatial information as it is crucial to predict the position of keypoints accurately as the input volume is processed deeper into the network. Also, the residual blocks can easily learn identity transforms drastically reducing the chance of over-fitting.

The first block in the network is a convolution layer with a batch norm (BN) followed by rectified linear units (ReLU) as the non-linear activation. The next 4 blocks are basic residual blocks with increasing channels $C = 64$, $C = 128$, $C = 256$ and $C = 512$ as depicted in Figure 4.14. Finally, there is a fully-connected layer that regresses the location of the keypoints.

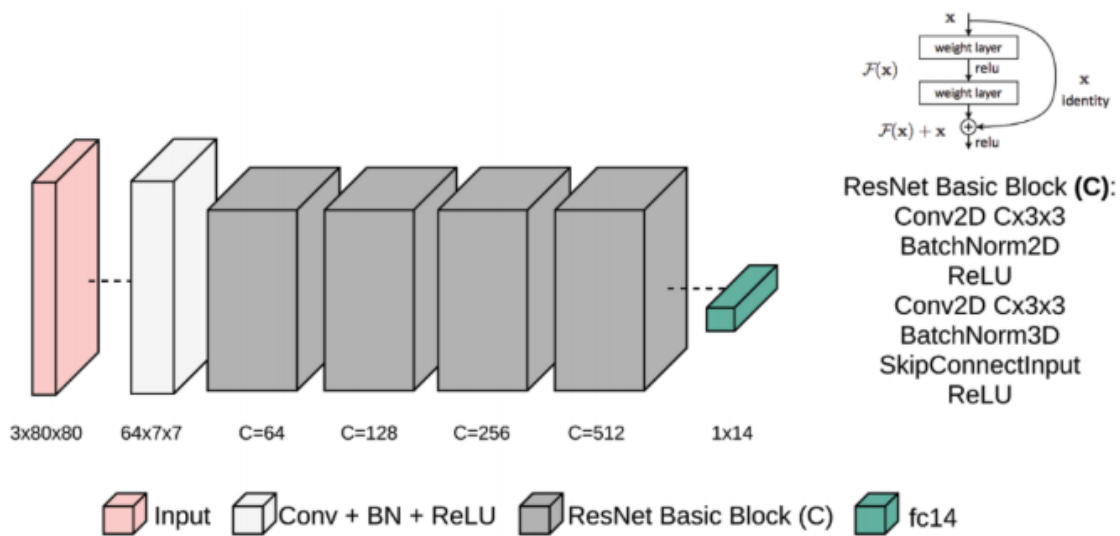


Figure 4.14: Architecture of the “keypoint network”. It takes a sub-image patch of $80 \times 80 \times 3$ as input and maps it to R14, the (x, y) coordinates for the 7 key points.

The cross-ratio (Cr) is a scalar quantity and can be calculated using 4 collinear points or 5 or more non-collinear points [10].

Since it is invariant under a projection and a camera in essence is a projective transform, this implies that the cross-ratio is preserved.

$$Cr(p_1, p_2, p_3, p_4) = (\Delta_{13}/\Delta_{14})/(\Delta_{23}/\Delta_{24}) \in \mathbb{R}$$

$$\Delta_{ij} = \sqrt{\sum_{n=1}^D (x_i^{(n)} - x_j^{(n)})^2}, D \in \{2, 3\}$$

It is preserved irrespective of the viewpoint of the scene and whether it is calculated in 3D or in 2D (on the image plane, after the projective transform).

In our case, we use 4 collinear points p_1, p_2, p_3, p_4 to calculate the cross-ratio as defined in Equation. Depending on whether the value is calculated for 3D points ($D = 3$) or their projected 2D counterparts ($D = 2$), the distance Δ_{ij} , between two points p_i and p_j is defined.

Jointly minimizing the squared error and the cross ratio

In addition to the cross-ratio to act as a regularizer, the loss has a squared error term. This forces the output to be as close as possible to the ground-truth annotation of the keypoints. The effect of the cross-ratio is controlled by the factor γ .

$$\begin{aligned} \sum_{i=1}^7 (p_i^{(x)} - p_{i_groundtruth}^{(x)})^2 + (p_i^{(y)} - p_{i_groundtruth}^{(y)})^2 \\ + \gamma (Cr(p_1, p_2, p_3, p_4) - Cr_{3D})^2 \\ + \gamma (Cr(p_1, p_5, p_6, p_7) - Cr_{3D})^2 \end{aligned}$$

Equation: represents the Loss function (L_{mse}) minimized while training the “keypoint regressor”..

The second and third term minimize the error between the cross-ratio measured in 3D (Cr_{3D}) and the cross-ratio calculated in 2D based on the “keypoint regression” output, indirectly having an influence on the locations output by the CNN. The second term in the equation represents the left arm of the cone while the third term is for the right arm, as illustrated in Figure 4.7. For the cross-ratio, we choose to minimize the squared error term between the already known 3D estimate ($Cr_{3D} = 1.3940842428872968$) and its 2D counterpart.

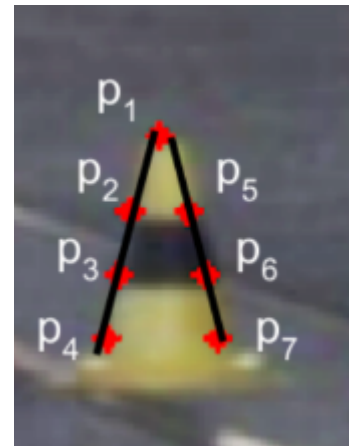


Figure4.15: An exemplary 80×80 cone patch with extracted “key points” overlaid in red. Depiction of the left (p_1, p_2, p_3, p_4) and right arm (p_1, p_5, p_6, p_7) of the cone. Both of which are used to calculate the cross-ratio terms and minimize the error between themselves and the cross-ratio on the 3D object (Cr_{3D}).

2) Modifications applied to the network architecture: Along with a sample output, is shown in Figure 4.16 To make the algorithm robust to single keypoint outliers all subset permutations of the keypoints with one point removed are calculated if the reprojection error from the PnP estimate using all keypoints is above a threshold. The permutation with the lowest error is used as the final estimate.

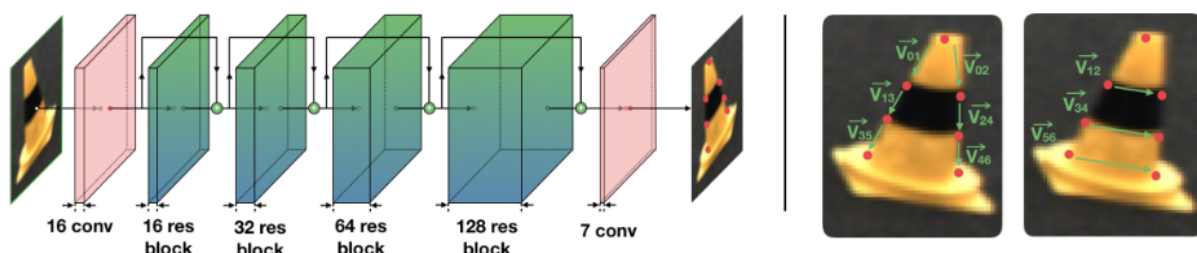


Figure 4.16: ResNet architecture (left). Vectors used in geometric loss function (right)

Two important modifications were made to the network:

First, the fully connected output layer was replaced with a convolutional layer, which predicts a probability heatmap over the image for each keypoint, and the expected value over the heatmap [11] is used as the keypoint location. The use of a fully convolutional network not only reduces the number of network parameters for faster convergence, but is also a better choice given the benefits of convolutional layers for tasks of predicting features that are spatially interrelated.

The second modification is an additional term in the loss function to leverage the geometric relationship between points. Since the key points on the sides of a cone are collinear, the dot products of the unit vectors between points on these lines should be one. One minus the values of these dot products are used directly in the loss function. The same is done for the three horizontal vectors across the cone. An illustration of these vectors is shown in Figure 4.16. Because the keypoint locations now need to be back propagated, the differentiable expected value function [11] is used to extract coordinates from heatmaps. The final loss function is as follows:

$$L_{total} = L_{mse} + \gamma_{horz}(2 - V_{12} \cdot V_{34} - V_{34} \cdot V_{56}) + \gamma_{vert}(4 - V_{01} \cdot V_{13} - V_{13} \cdot V_{35} - V_{02} \cdot V_{24} - V_{24} \cdot V_{46})$$

Using a Bayesian optimization framework like the one previously described, the values were determined to be :

$$\gamma_{vert} = 0.038 \text{ and } \gamma_{horz} = 0.055.$$

3) Training the dataset

The DNN was trained with 3.2k images with 18k cone patches like in Figure 4.12 (right) to run to detect seven key points on each cone patch detection for use in a Perspective-n-Point (PnP) algorithm(next module). Cone patches were extracted from full images and manually hand-labeled. The dataset was further augmented by transforming the image with 20 random transforms consisting of rotation, scaling and translation. The data is split as 16,000 cone patches for training and 2,000 cone patches for testing. During the training procedure, the data is further augmented on the fly in the form of contrast, saturation and brightness. Stochastic Gradient Descent (SGD) was used for optimization, with a learning rate, $l_r = 0.0001$ and momentum = 0.9 and a batch size of 128. The learning rate is scaled by 0.1 after the first 75 and 100 epochs. The network is trained for 250 epochs.

4) 3D Localization of cones - Pnp algorithm

The “Keypoint Extraction Network” provides accurate locations of very specific features, the key points. Since, there is a **priori** information available about the shape, size, appearance and 3D geometry of the object, the cone in this case, 2D-3D correspondences can be matched. With access to a **calibrated** camera and 2D-3D correspondences, it is possible to estimate the pose of the object in question from a **single** image.

We define the camera frame as F_c and the world frame as F_w . F_w can be chosen arbitrarily, as long as it is used consistently. In this case, we choose the world frame, F_w to be at the base of the cone, for ease of measurement of the 3D location of the keypoints (with respect to F_w) and convenience of calculation, as will become apparent. We use Perspective n-Point or PnP to estimate the pose of every detected cone. This works by estimating the transform C_t between the camera coordinate system and the world coordinate system. Since, the world coordinate system is located at the base of the cone, lying at an arbitrary location (that we want to estimate) in R^3 This transform is exactly the pose we are looking for.

A pose consists of a translation and a rotation. The fact that the cone is symmetric along the axis through its apex and center of the base simplifies the situation. As we are concerned only with the translation between F_c and F_w , which is exactly the position that we care to estimate, we can discard the orientation due to the cone’s symmetric geometry.

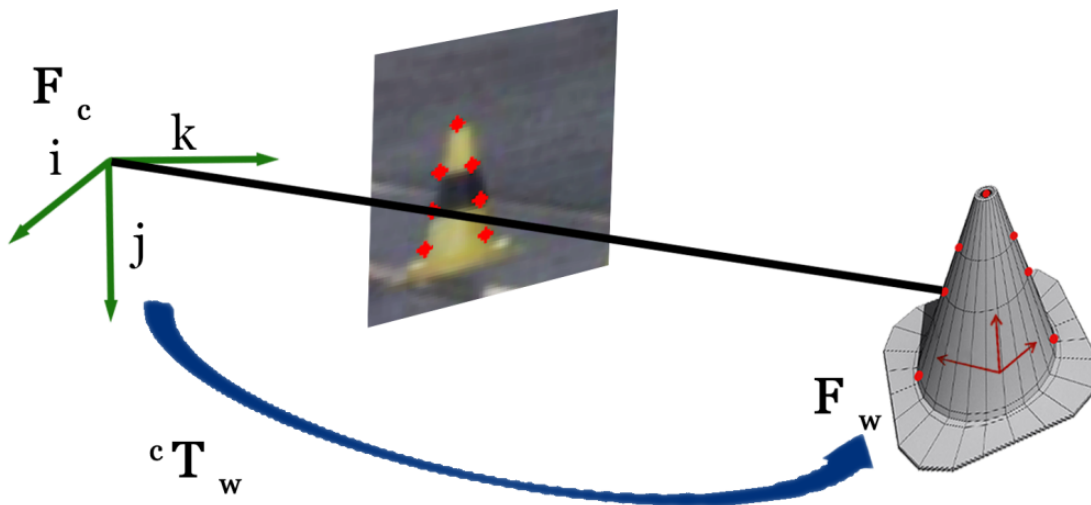


Figure 4.17 Schematic illustrating matching of 2D-3D correspondence and estimation of transformation between the camera frame and the world frame.

To estimate the position of the cone accurately, we use the non-linear version of the PnP implemented in the OpenCV library [12] that uses Levenberg-Marquardt to obtain the transformation. In addition, RANSAC PnP is used instead of vanilla PnP, tackling noise correspondences. RANSAC PnP can be done for every cone detected, that is extract the 7 features by passing the patch through the “keypoint regressor” and use the pre-computed 3D correspondences to estimate the transform, allowing to estimate pose of multiple objects from a single image using a priori knowledge about the object of interest.

4.4.3. Design Constraints

In computer vision one of the most important steps in most of the computer vision algorithms is the feature extraction module because it is used to detect the changes in intensity or in the shape or to detect corners and edges and the key points which are special for this object so these things can be helpful in object classification, structure from motion, place recognition, locating “points of interest” that are unique and distinguishable and has always been a fundamental technique for decades. Most of the feature extraction algorithms are generic and work for extracting fine details in the images. There has been a large collection of work that focuses on finding better, faster, more efficient, robust feature extraction techniques. Most of these are very generic and can be used in arbitrary applications. A desirable property that many of these possess is invariance to transformations such as scale, rotation and illumination. Such work includes Harris corners [13], renowned SIFT[14], SURF[15], efficient features with binary descriptors: BRISK[16] and BRIEF[17].

The intrinsic parameters of the monocular camera are known as a result of the calibration using a large checker-board to a distance up to 15 meters. One would be able to estimate an object’s 3D pose, if there is a 2D-3D correspondence between the 3D object and the 2D image, additionally the calibration parameters of the camera.

The problem with the generic feature extraction technique is that it detects any changes or fine changes with the image and if the resolution of the image is low it will not detect the all points that we need , so in our case we need exactly 7 points on a specific locations on the cone patch as discussed in the last section. For instance, a Harris corner does not distinguish whether it lies on a cone or on a crack on the asphalt. This makes it hard to draw the relevant correspondences and match them to their 3D counterparts. In our case the mapping accuracy and the 3D positions of the cones in the 3D map will depend on these 7 key points. To this end, we implemented a feature extraction scheme that is inspired by classical computer vision but has a flavor of learning from data via machine learning and deep learning algorithms.

4.5 Stereo camera pipeline

4.5.1 Functional description

In order to achieve the redundant perception concept which we talked about in the system architecture, using the monocular camera in short range and the stereo camera in the long range (up to 20m), we developed a stereo pipeline which uses the left frame and the depth map of the stereo camera and produce the 3D cone positions in each frame. In this module we will describe the implemented stereo camera pipeline which is used to detect and localize the track cones in the 3D space up to 20m using the stereo camera.

Using the first module (2D space localization and cone detection) to detect the cone batches (bounding boxes) in each frame and estimate the 2D position of each cone in the 2D space (image space), then using the intrinsic camera parameters of the stereo camera and the depth map we localize each cone in the 3D world.

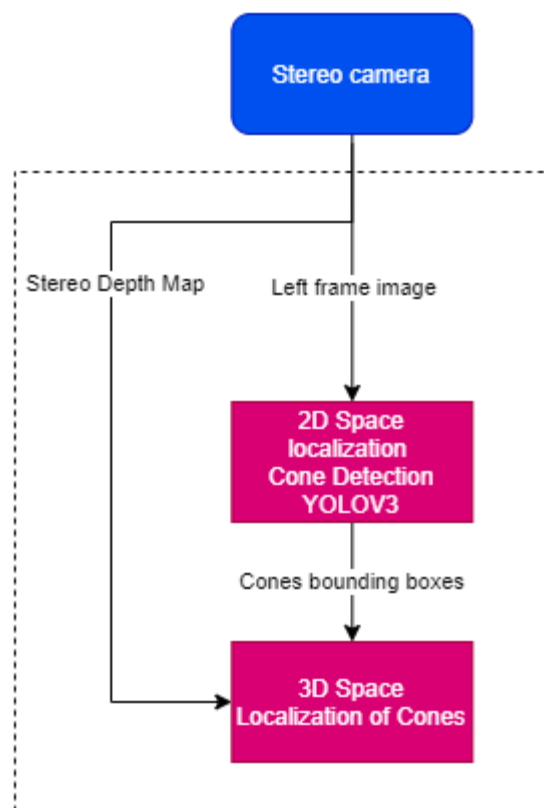


Figure 4.18 Stereo camera pipeline's block diagram

4.5.2 Modular decomposition

In this section the stereo pipeline will be described using the ZED stereo camera.

The used sensor message from the stereo camera are:

- left/image_rect_color: Left camera color rectified image
- depth/depth_registered: Depth map image registered on left image (32-bit float in meters by default)

In order to get the 3D position of each cone in the frame by using these sensor data from the stereo camera the following steps are applied on these data:

1- Object detection on the rectified left image

Using the left frame rectified image frame as input from the stereo camera we pass it to the object detection module to extract the cone boundary boxes which exists in the rectified image, so we can know the 2D position of each cone in the rectified image

Image rectification is a transformation process used to project images onto a common image plane. This process has several degrees of freedom and there are many strategies for transforming images to the common plane. It is used in computer stereo vision to simplify the problem of finding matching points between images (i.e. the correspondence problem).

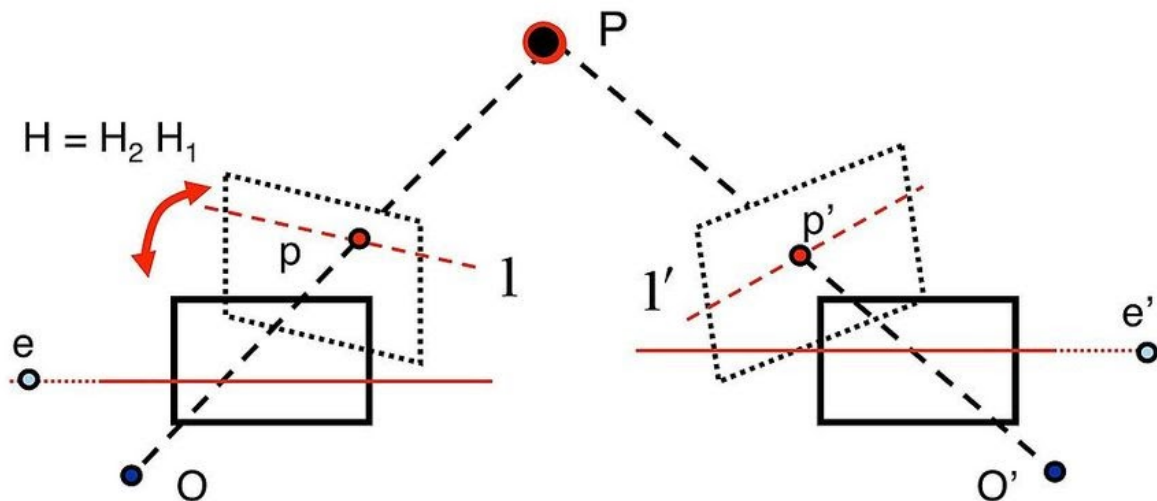


Figure 4.19: Image rectification example

Till now we have the bounding boxes of the cones (2D position of each cone) on the left rectified image frame in form if $[xmin,ymin,xmax,ymax]$ as shown in Figure 4.20, so the next step is to calculate the position of the center of each cone bounding box in the 2D left rectified image frame.

Calculations:

- for each detected cone (boundary box) $[x_{min}, y_{min}, x_{max}, y_{max}]$ are given from the detection pipeline as shown in Figure 4.20 (right)
- $box_width = X_{max} - X_{min}$
- $box_height = Y_{max} - Y_{min}$
- $center_x = X_{min} + box_width / 2$
- $center_y = Y_{min} + box_height / 2$

The center point $[center_x, center_y]$ will be used in the next step to identify the position of the cone in the depth map in the next step.



Figure 4.20: 2D Cone bounding box detection(left) boundary box points (right)

2- Depth map

The ZED stereo camera reproduces the way human binocular vision works. Human eyes are horizontally separated by about 65 mm on average. Thus, each eye has a slightly different view of the world around. By comparing these two views, our brain can infer not only depth but also 3D motion in space.

Likewise, Stereolabs stereo cameras have two eyes separated by 6 to 12 cm which allow them to capture high-resolution 3D video of the scene and estimate depth and motion by comparing the displacement of pixels between the left and right images.

Depth maps captured by the ZED stereo camera store a distance value (Z) for each pixel (X, Y) in the image. The distance is expressed in metric units (meters for example) and calculated from the back of the left eye of the camera to the scene object.

Depth maps cannot be displayed directly as they are encoded on 32 bits. To display the depth map, a monochrome (grayscale) 8-bit representation is necessary with

values between [0, 255], where 255 represents the closest possible depth value and 0 the most distant possible depth value.

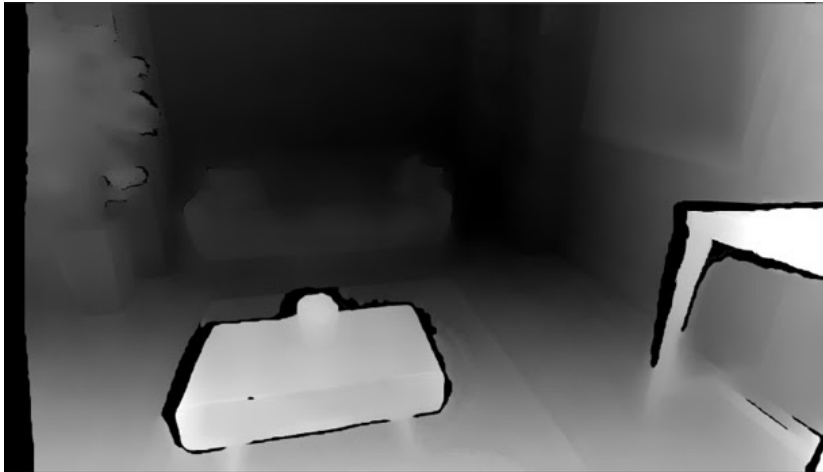


Figure 4.21: Stereo camera depth map 2D image

Using the center point [center_x,center_y] which is produced from the previous step we can find the cone depth in the depth map and using the intrinsic camera parameters (cx,cy,fx,fy) to convert from pixel coordinate to world coordinate we can calculate the 3D position of the cone in the world coordinates by the following calculations:

- By taking the depth from a square area around the center of the cone for better results We can calculate Z by:
 $Z = \text{lastFrame.depth}(\text{center})$ taking into consideration the square area and calculating the mean.
- $y_{\text{world}} = (\text{round}(y+h/2) - cy) / fy * z$
- $x_{\text{world}} = (\text{round}(x+w/2) - cx) / fx * z$
-

So now we have the XYZ (3D position) of each cone in the stereo camera frame. These cone positions will be fused with the Monocular pipeline using extended kalman filter (EKF) to ensure the redundant perception in the simultaneous localization and mapping (SLAM) algorithm which will be described in the next module in detail and how the mapping algorithms works and fuses all of these sensors' data.

4.5.3 Design constraints

While designing the Stereo camera pipeline and choosing the stereo camera specifications, we take into consideration the perception design concept which is the redundant perception Monocular camera for the short range and the stereo camera

for the long range up to 20m, so the specifications and limitations of the chosen algorithm and hardware works with the following constraints:

- Depth range corresponds to the minimum and maximum distance at which the depth of an object can be estimated, so the depth range of the stereo pipeline is
min=0.3m, and max=20m but taking into account the depth accuracy Stereo vision uses triangulation to estimate depth from a disparity image, with the following formula describing how depth resolution changes over the range of a stereo camera:

$Dr = Z^2 \cdot \alpha$, where Dr is depth resolution, Z the distance and α a constant.

Depth accuracy decreases quadratically over the z-distance, with a stereo depth accuracy of 1% of the distance in the near range to 9% in the far range. Depth accuracy can also be affected by outliers' measurements on homogenous and textureless surfaces such as white walls, green screens and specular areas. These surfaces usually generate temporal instability in depth measurements.

So the chosen range for the stereo pipeline is up to 20m to estimate the cone positions in an accurate way for the long range.

- Depth FOV: The chosen field of view for the stereo pipeline to match with the design concept of mapping taking into account the track width is 110° (H) x 70° (V) x 120° (D) max.
- Depth FPS: In order to achieve the max performance of the perception pipeline of the high performance autonomous vehicle the chosen frame per seconds of the stereo camera is up to 100Hz

4.6 EKF Robot Localization and Mapping

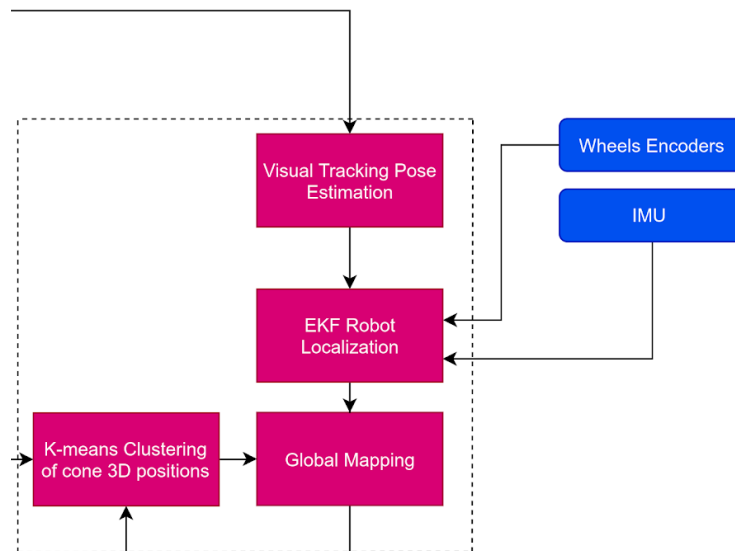


Figure 4.22: EKF Robot localization and mapping block diagram

4.6.1 Overview and Assumptions

In this section we will describe the implementation of three modules, EKF robot localization module, Visual odometry module received from the stereo camera, and k-means-based global mapping module. These modules are responsible to output two main things; a real time map of the environment and the vehicle pose in this environment in order for the path planning algorithm to use these information and output the trajectory of the vehicle to the MPC module in which it will use it for sending actuating commands to the motors controllers.

EKF Robot Localization

We implemented the module within a node in the Robotics Operating Systems ROS (see section 4.9) the module is an implementation of an extended Kalman filter. It uses an omnidirectional motion model to project the state forward in time, and corrects that projected estimate using perceived sensor data. Using a 15-dimensional state of the vehicle

$$(X, Y, Z, pitch, yaw, roll, x', y', z', pitch', yaw', roll', x'', y'', z'')$$

We fuse the IMU, wheel encoders, and the visual odometry in order to get an accurate vehicle pose estimation. We will discuss the parameters selection in the next subsection.

Visual Odometry

We are using the ZED Stereo Camera developed by StereoLabs which is a camera system based on the concept of human stereo vision. In addition to viewing RGB, stereovision also allows the perception of depth. Advanced computer vision and geometric techniques can use depth perception to accurately estimate the 6DoF pose $(x,y,z,roll,pitch,yaw)$ of the camera and therefore also the pose of the system it is mounted on. Visual Odometry is the process of estimating the motion of a camera in real-time using successive images. We used a visual odometry as an input to the ekf robot localization module alongside the IMU which gives

$(pitch', yaw', roll', x'', y'', z'')$ and wheel speed encoders that give indication about the speed and orientation.

K-means-based Global Mapping

We will be using a database of cones that includes the cones positions, colors and the mean and covariance that define the uncertainties of the cones detections. In this sense, we are using the sampled data to increase and decrease the uncertainty according to the number of hits and the difference of position estimates for existing cones and the added cones to the database. These detections are first translated from the vehicle frame to the global frame according to the pose estimation from the EKF localization module as shown in the next figure.

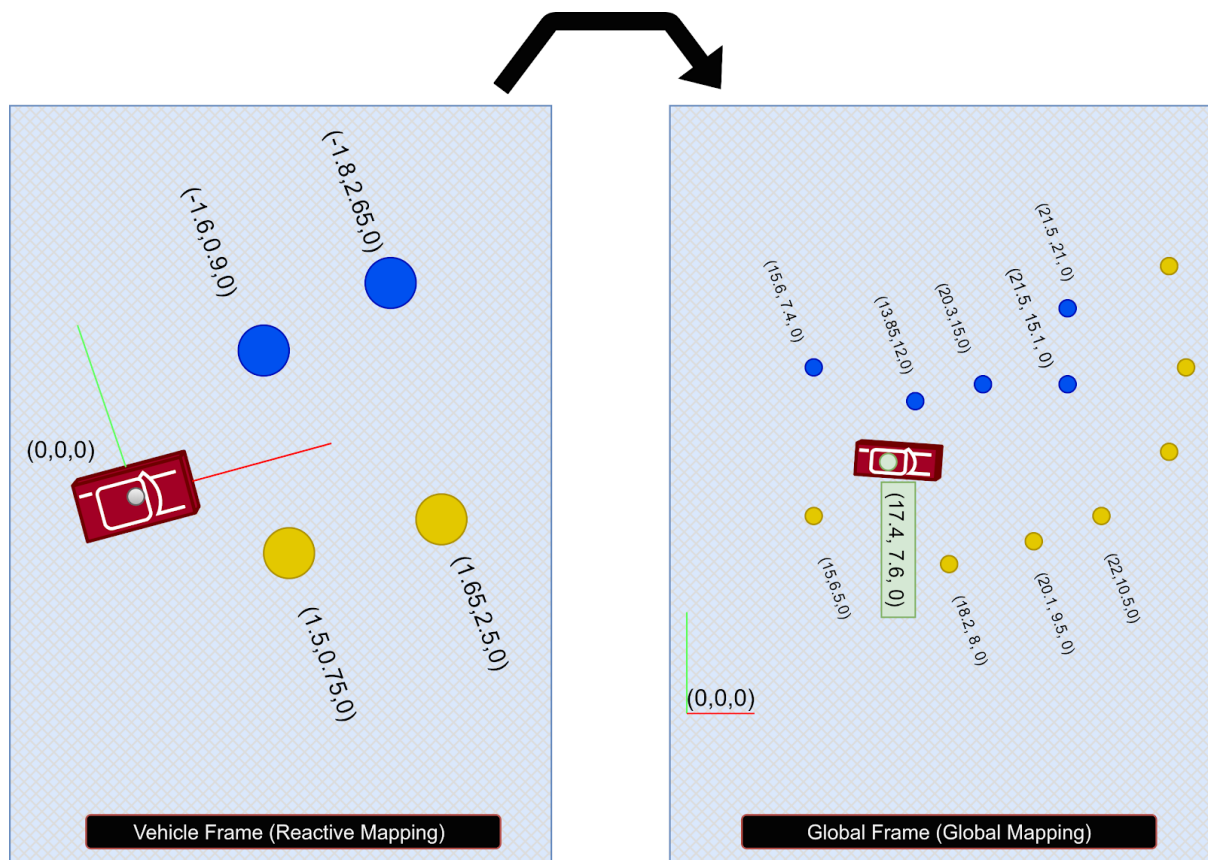


Figure 4.23: Vehicle Frame to Global Frame Transformation

This is done by multiplying the received cone positions (X_w, Y_w, Z_w) by the transformation matrix containing the translation vector and the rotation matrix of the vehicle to get the global position of the cones (X_c, Y_c, Z_c) in order to update the map with the new samples.

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

4.6.2 EKF Robot localization Design Parameters

Each sensor reading updates some or all of the filter's state. These options give greater control over which values from each measurement are fed to the filter. The order of the values is

$(x, y, z, roll, pitch, yaw, vx, vy, vz, vroll, vpitch, vyaw, ax, ay, az)$.

When measuring one pose variable with two sensors, a situation can arise in which both sensors under-report their covariances. This can lead to the filter rapidly jumping back and forth between each measurement as they arrive. In these cases, it often makes sense to (a) correct the measurement covariances, or (b) if velocity is also measured by one of the sensors, let one sensor measure pose, and the other velocity. However, doing (a) or (b) isn't always feasible, and so we expose the differential parameter. When differential mode is enabled, all absolute pose data is converted to velocity data by differentiating the absolute pose measurements. These velocities are then integrated as usual.

EKF Localization uses a 3D omnidirectional motion model. The robot's position in the vehicle frame will drift over time, but is accurate in the short term and should be continuous; this short term accuracy is enough to estimate the track trajectory and boundaries. The vehicle frame is therefore the best frame for executing local motion plans. The global frame, unlike the vehicle frame, is a world-fixed coordinate frame, and while it contains the most globally accurate position estimate for our vehicle, it is subject to discrete jumps, e.g., due to the fusion of IMU data or a correction from a global-based localization node.

All of that was implemented in the ROS node receiving data from IMU, wheel encoders, and visual odometry and publishes the pose of the vehicle to the Global Mapping Module .

4.6.3 Visual Odometry Modular Decomposition

Stereo visual odometry consists of five steps. Firstly, the stereo image pair is rectified, which undistorts and projects the images onto a common plane. Feature detection extracts local features from the two images of the stereo pair. Then, Stereo Matching tries to find feature correspondences between the two image feature sets. Since the images are rectified, the search is done only on the same image row. Usually the search is further restricted to a range of pixels on the same line. There is also an extra step of feature matching, but this time between two successive frames in time. Finally, an algorithm such as RANSAC is used for every stereo pair to incrementally estimate the camera pose. This is done by using the features that were tracked in the previous step and by rejecting outlier feature matches.

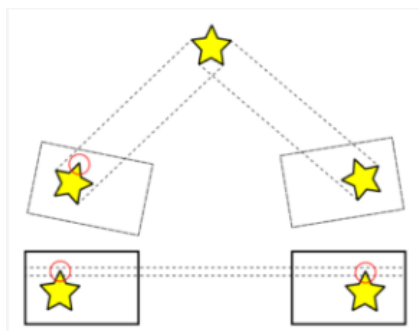


Figure 4.24: Stereo Rectification

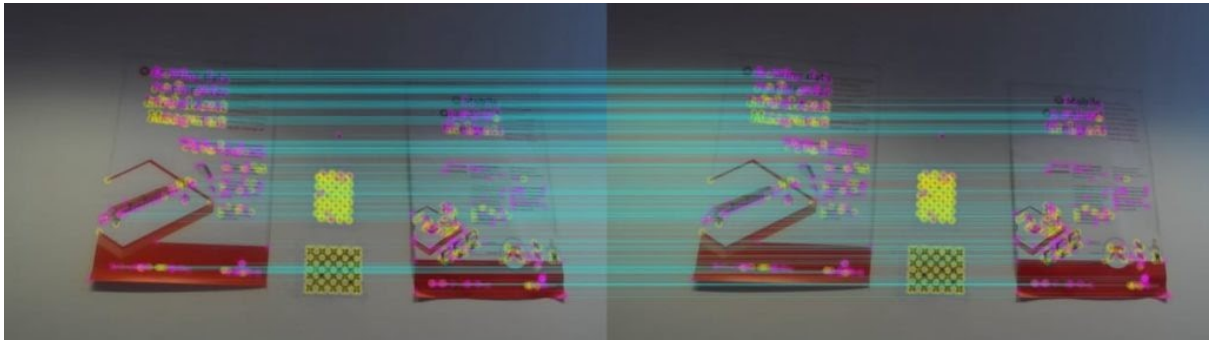


Figure 4.25: Stereo Feature Matching

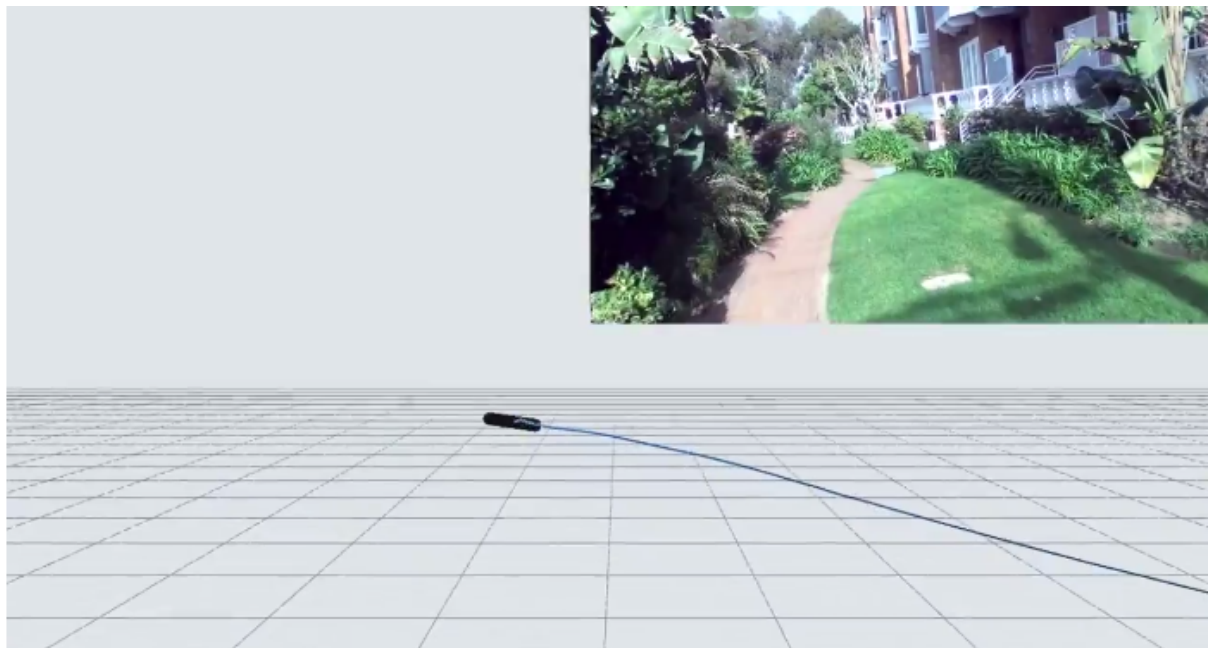


Figure 4.26: Stereo Camera Visual Odometry Pose Estimation

Due to the incremental nature of this particular type of pose estimation, error accumulation is inevitable. The longer the system operates, the bigger the error accumulation will be. Therefore, we improve the pose estimation algorithm by fusing the visual odometry with the IMU and the wheel encoders to provide a more robust pose estimate.

4.6.4 K-means-based Global Mapping Modular Decomposition

As mentioned before, we are using a database containing the positions of the cones (landmarks) in the track, their colors, and the covariance R that indicates the amount of uncertainty about this information that is updated or augmented using the samples received. Each sample here is a global cone position (after being transformed from the vehicle frame to the global frame as illustrated above) and the cone color. The sample received is either for a previously received cone information thus it will change the covariance R (changing the uncertainty based on the euclidean distance between the new sample and the mean of previous samples for the same cone) OR the sample is for a new cone that was previously unseen and this is again determined by using thresholding on the euclidean distance between the sample and the previous samples, that is if the new sample is away from previous samples then it might be a new cone information so it is added to the database with the maximum covariance R until receiving more samples at the same location or close to it in order to decrease this covariance. In addition, if the cone estimate in the database remains with high covariance between samples until it is out of the field of view (FOV) of the vehicle, it is then removed from the database as it will be considered as a random noise. The next figure illustrates the process of adding cones and updating them in the database.

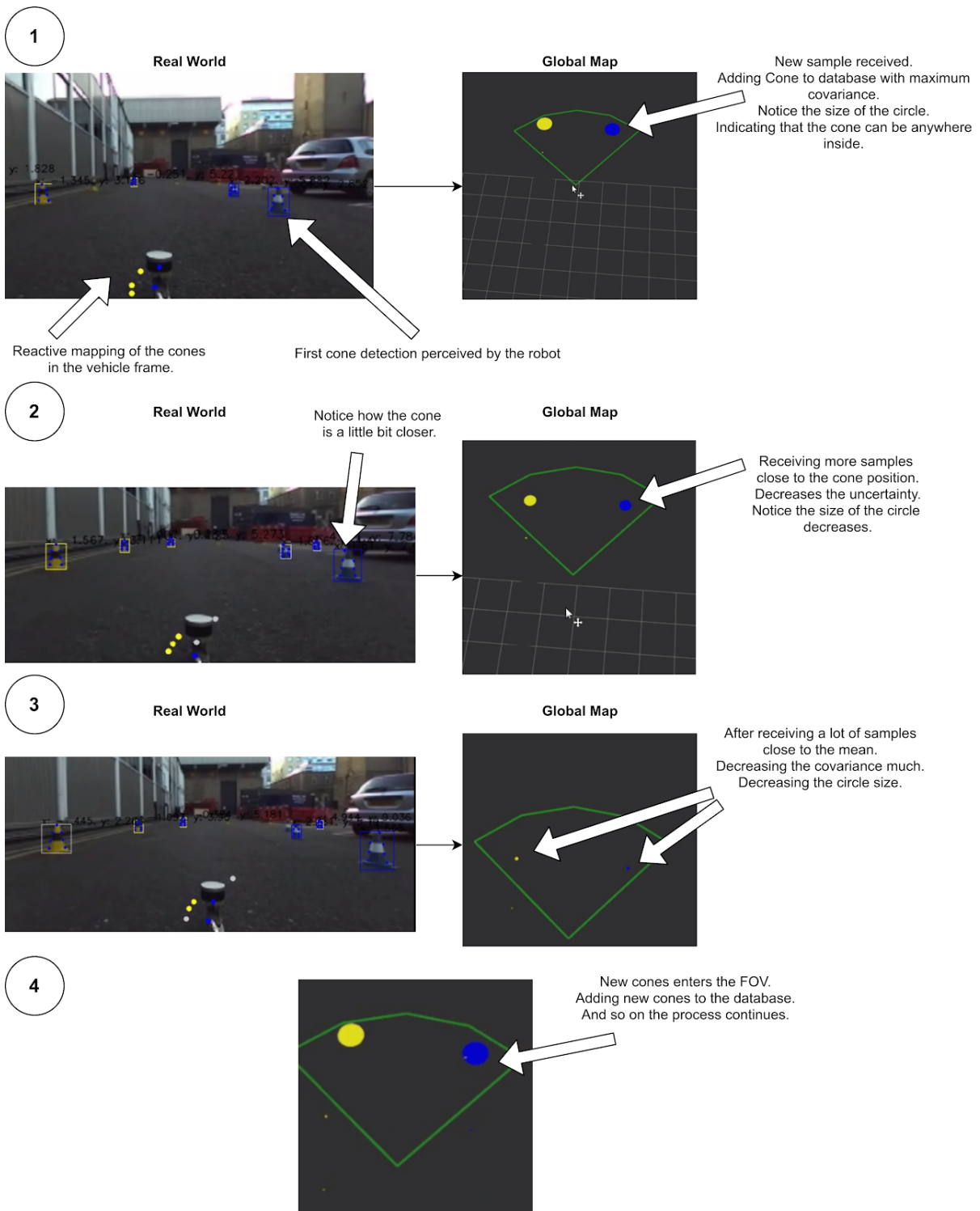


Figure 4.27: The process of adding and updating cone estimates in the global map.

4.6.5 K-means-based Global Mapping Design Constraints

Thinking about the constraints that lead to this implementation can be illustrated with this tradeoff; between the accuracy with which we need to map the track, and the time we need to finish the SLAM round for perceiving the track layout and pass it to the motion control algorithm. This is simply thought of as when the vehicle is moving slowly while receiving samples then the map will receive more samples to build a more accurate track, while when moving fast the amount of samples received for the same landmarks decreases and thus the accuracy of building the map decreases. In addition, controlling the threshold of the FOV prune the uncertain cone estimates received and depend more on the close cone estimates which the vehicle can be more confident about. Hence, our objective was more to build an accurate map of the track and then race with maximum performance in this track. In addition, we proposed modes of operation in order for the vehicle to work with, these modes of operation will decide which parameters it will use while mapping the environment and will range between being inaccurately fast and being accurately slow in mapping. Here, we must assure that this is just for mapping where after the SLAM round is finished the vehicle can race the track with its maximum performance.

The modes of operation will change the configuration parameters which we will discuss now.

Maximum Depth FOV, the maximum depth of a cone estimate to be considered as a received sample.

Minimum Depth FOV, the minimum depth of a cone estimate to be considered as a received sample, as some vehicles may have some difficulties sensing close cones because of the positioning of cameras on the vehicle.

Angle FOV, the angle in front of the vehicle by which the cone estimates inside this angle of view is considered as a cone estimate.

Maximum Hits, the number of samples received for the same cone (close to the mean of samples) after which we are extremely confident about its position and we don't need to consider any samples for it again, because if there was a drifting error in the sensors which accumulate with time adding more samples would influence the cluster with wrong estimates.

Minimum Hits, the number of samples received away from the mean of the samples, in which if the cone estimate reached and the mean became out of the FOV then the cone estimate will be removed from the database. Note: the hits number is increased or decreased according to the received samples, that is if the samples are close to the mean, then the hits number is increased, while if they are away from it, it will be decreased to finally reach the maximum hits or the minimum hits or somewhere in between.

Minimum Covariance, the number in which if the covariance of a cone estimate becomes below even if it exists in the FOV it will be removed from the database directly because it represents a random error.

Maximum Covariance, the number in which if the covariance of a cone estimate reaches, it will never be considered for removal from the database.

Hit-sample Covariance, the number added to the covariance when a hit happens, that is receiving a sample close to the mean of the cluster of samples.

Failure-sample Covariance, the number subtracted from the covariance when a failure happens, that is receiving a sample for the cone estimate away from the mean of the clusters of samples.

The **Covariance** of a cone estimate is updated as follows,

$$\Delta_{cov} = Max_{cov} \frac{Max_{hits} - 1}{Max_{hits}^2}$$
$$Cone_{cov} = Max_{cov} - \Delta_{cov} \times Cone_{hits}$$

where, Max_{cov} is the maximum covariance, Max_{hits} is the maximum hits, $Cone_{cov}$ is what we are updating, and $Cone_{hits}$ is the number of hits the cone estimate has received till now.

4.7 Model Predictive Control

4.7.1 Overview and Assumptions

In this section we will talk about motion control in full details. This chapter represents the body of our project so we should answer some questions like “what has been done?”, and “how it has been done?”. We also will discuss and clarify our scientific approaches and methodologies.

After the first successfully finished lap, the complete track is mapped using the SLAM algorithm. After this step the vehicle now has the knowledge of the track layout and has the ability to localize itself within the environment. By having this capability, we can race the car around a known track. Which brings us to our motion planning problem where the goal is to drive around the track as fast as possible and avoid oversteering in corners and hard braking. For this purpose we tried to solve this problem by using nonlinear Model Predictive Control which aims to maximize the progress along a given reference path (in our case the center line) while respecting the vehicle model and track constraints. The two main advantages of our Model Predictive Control is the direct consideration of the vehicle limits when computing the

command and that the algorithm does not need any pre-determined logic, only the track layout, and the vehicle model.

4.7.2 MPC Design parameters

Before talking about how an MPC controller works and showing its architecture we first have to talk about its parameters.

Choosing proper values for these parameters is important as they affect not only the controller performance but also the computational complexity of the MPC algorithm that solves an online optimization problem at each time step. We will also talk about how to choose these parameters.

These parameters are:

- Sample time
- Prediction Horizon
- Control Horizon
- Constraints
- Weights

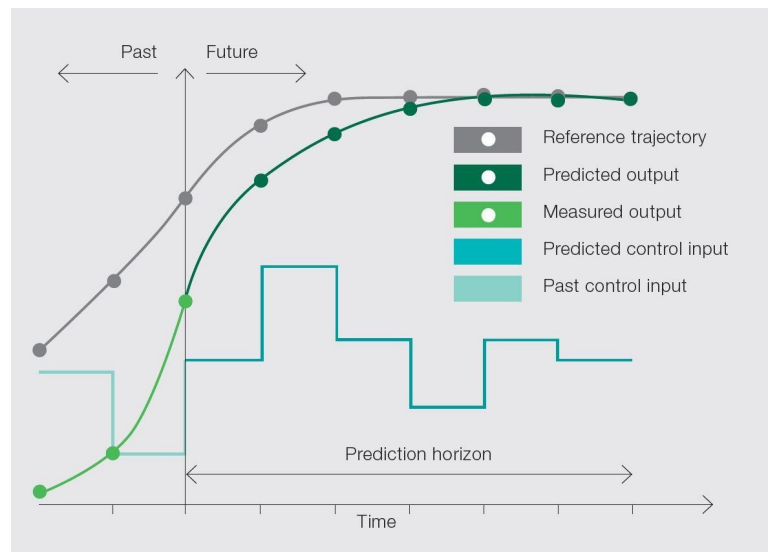


Figure 4.28: parameters of MPC

By choosing the sample time we determine the rate at which the controller executes the control algorithm if it is too big, when a disturbance comes in, the controller will not be able to react to the disturbance fast enough. On the contrary, if the sample time is too small, the controller can react much faster to disturbances and setpoint changes but this causes an excessive computational load. To find the right balance between performance and computational effort the recommendation is to fit 10 to 20 samples within the rise time of the open-loop system response.

As we will discuss, at each time step, the MPC controller makes predictions about the future plant output and the optimizer finds the optimal sequence of control inputs that drives the predicted plant output as close to the setpoint as possible. The number of the predicted future time steps is called the prediction horizon and shows how far the controller predicts the future. What happens if it is too short? Think of this example, while going at 50 mph you know that it will take your car 5 seconds to stop if you press on the brake pedal. If your prediction horizon is 2 seconds, by the time you see the traffic lights, it will be too late to apply the brakes. The car will only be able to stop after passing the traffic lights. So we have to choose the prediction

horizon that will cover the significant dynamics of the system, but why do not we select a much longer prediction horizon? The answer is because of computational loads. Another example is when replacing traffic lights with corners this may lead to death if we drive a formula car with maximum speed and we did not see the corner on time we will make an accident. Another design parameter is the control horizon which is shown in the above figure. Each control move in the control horizon can be thought of as a free variable that can be computed by the optimizer. So the smaller the control horizon the fewer the computations. If we make it only one step it may not give us the best maneuver. And by increasing the control horizon we can get better predictions but at the cost of increasing the complexity and usually the first couple of moves have a significant effect on the predicted output behavior while the remaining moves have only minor effects. So choosing a large control horizon only increases computational complexity. The best way to choose a control horizon is to set it to 10% or 20% of the prediction horizon. MPC can incorporate constraints on inputs, rate of change of inputs and the output. These can be hard or soft constraints. Hard constraints can not be violated while soft constraints can be violated. The recommendation is to set output constraints as soft and avoid having hard constraints on both inputs and rate of change of inputs. MPC has multiple goals, we want outputs to track as close as possible their setpoints but at the same time we want to have smooth control moves to avoid aggressive control maneuvers. The way to achieve a balanced performance between these competing goals is to weigh the input rates and outputs relative to each other. We also adjust relative weights within the groups as well. For example if in a 2x2 system it is more critical to perform reference tracking of the first output than the second output so we assign a larger weight to the first output.

4.7.3 Block Diagram

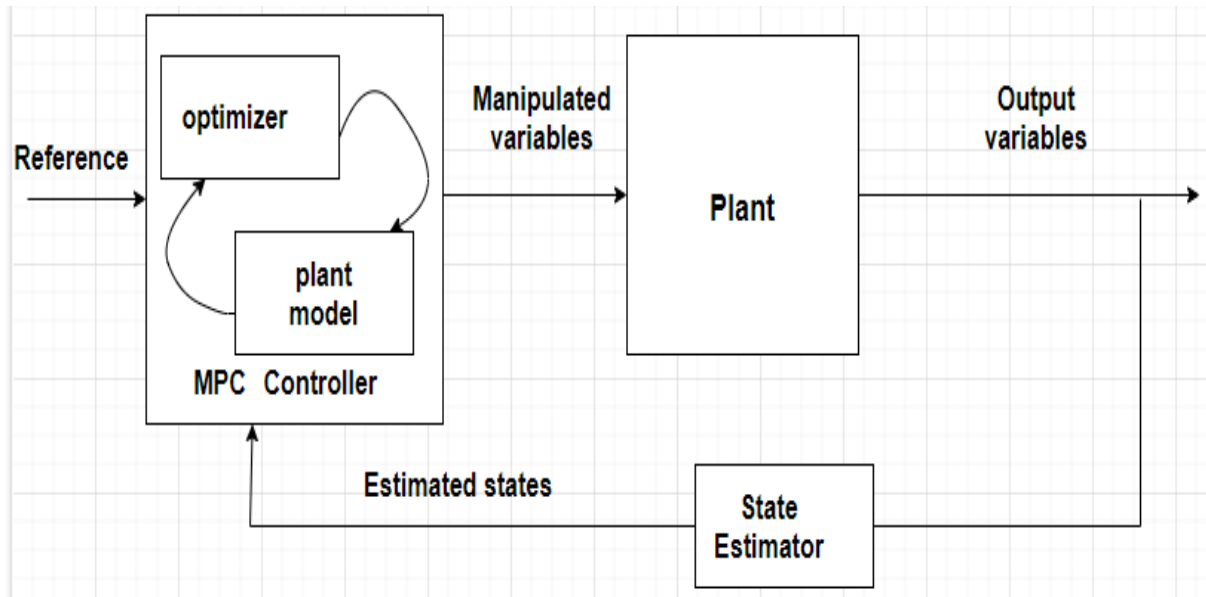
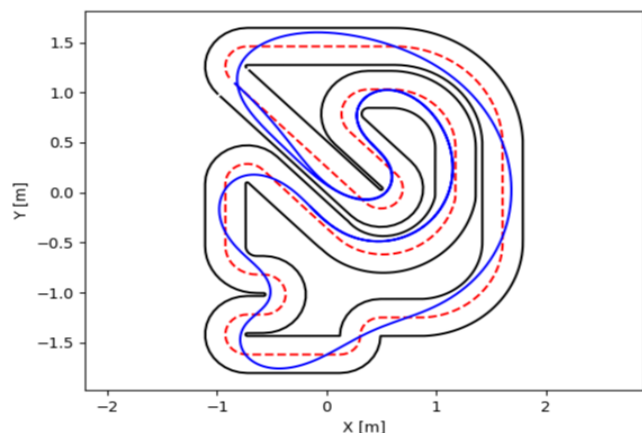


Figure 4.29: Architecture of implemented MPC

4.7.4 Functional Description

In this part we will talk about and describe functionality of our controller which is Model Predictive Control (MPC) controller to control motion of our vehicle from the complete track that is mapped using simultaneous localization and mapping algorithm which is mapped successfully in the first lap. MPC utilizes the model of the system to predict its future behavior and it solves an online optimization problem to select the best control action that drives the predicted output to the reference. In the control problem the goal of the controller is to calculate the input to the plant such that the plant output follows the desired reference. In model predictive controller's strategy to compute this input is to predict the future, it sounds like fortune-telling but we will discuss how it works. MPC uses the model of the plant to make predictions about the future plant output behavior. It also uses an optimizer which ensures that the predicted future control output tracks the desired reference. In our design let us say that we control our vehicle by model predictive control controller to keep it in the middle of the lane for simplicity we assume that our speed is constant and our control input is just the steering angle which is used to control our car.



First we will talk about how the controller uses the car model then later we will discuss the optimizer. As we said before that we consider the middle of the lane to be our reference. At the current time the MPC controller uses the car model to simulate the vehicle's path in the p (where p is the prediction horizon) next time steps if the steering wheel would be turned as in the below figure where reference path is in green and path controlled by steering angle is in purple color. P is the measure of how far ahead MPC looks into the future and is referred to as the prediction horizon. It is often represented by the length of time into the future or the number of future time steps.

Figure 4.30: Vehicle's path in blue car vs reference path in red color

The MPC controller needs to find the best predicted path that is the closest to the reference. So it simulates multiple future scenarios. These scenarios are simulated in a systematic way not in a random order and this is why the optimizer comes into the picture. By solving the optimization problem the MPC controller tries to minimize the error between the reference and predicted path of the car. It also tries to minimize the change in the steering

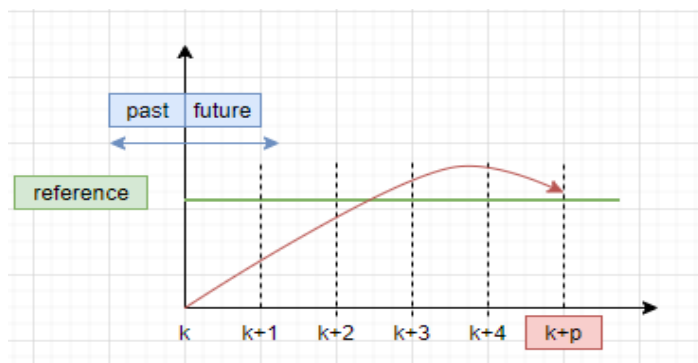


Figure 4.31: best predicted path close to reference taken over p time steps

wheel angle from one time step to the next because if the steering wheel is turned sharply, the ride may become uncomfortable for the passengers. The cost function J of this optimization problem is used to choose the best path. While minimizing this cost function MPC also makes sure that the steering angle and vehicle's position stay within prescribed limits. These are referred to as constraints. There is a limit on how far the steering wheel can be turned as we know.

At the current time step the MPC controller is solving the optimization problem over the prediction horizon while satisfying the constraints. The predicted path with the smallest or minimum J gives the optimal solution, and therefore determines the optimal steering wheel angle and other control input sequences that will get the car as close as possible to the reference path.

At the current time, MPC applies only the first step of this optimal sequence to the car and disregards the rest, based on the applied steering wheel angle the car travels some distance. At the next time step the controller gets a new measurement of the car position and it might be slightly different than what MPC controller has predicted before. This could be because of some unmeasured disturbances acting on the vehicle it might be the wind or slippery road surface. Now the prediction horizon shifts forward one time step and the controller repeats the same cycle of calculations to compute the optimal steering wheel angle for the next time.

In the above figure manipulated variables are the inputs to the plant which is our vehicle that is used to control the vehicle these inputs are steering angle and torque request or in other word manipulated variables are signals computed by the controller and sent to the plant. While output variables are variables that we try our best to make it the same as our reference that is mapped using the SLAM algorithm in the first lap.

In a feedback diagram there is a state estimator that is used if we could not directly measure the states so these states can be estimated using this state estimator and fed back to the model predictive control controller.

4.8 Communication and Simulation module

4.8.1 Functional Description

In order to ensure the reliability of our modules a high performance framework is chosen to build a communication environment which enable us to visualize and see the interaction between the pipeline's modules, and by implementing our pipeline from perception to mapping and localization of the vehicle it is found that most of pipeline modules needs to interact and send data to each other starting from camera sensing ending with producing the torque request and steering angle, so the pipeline's modules are run as nodes using Robot Operating System or ROS as the framework that eases handling of communication and data messages across multiple systems as well as different nodes. Different modules communicate via messages, they receive data and output processed information. Another important aspect is that ROS is open-source and provides tools for visualization, monitoring and simulation, making it easy to integrate, test, diagnose and develop the complete software system. In this module, a detailed description of the deployment of the pipeline modules and transform them to ROS nodes. Each node in the ROS framework represents a module or submodule in the pipeline and each node takes its input data by subscribing to a topic which another node produced. This messaging system which based on publish and subscribe methodology is used to manage the communication between modules in an effective way and used to

reduce the time to make a low latency and efficient pipeline. Also, ROS framework is used to visualize and simulate the pipeline using rViz by visualizing the detected cones and the produced map of the track as shown in [figure](#). and these simulations and visualization are used in testing and validation of the pipeline as will be described in chapter 5.

ROS

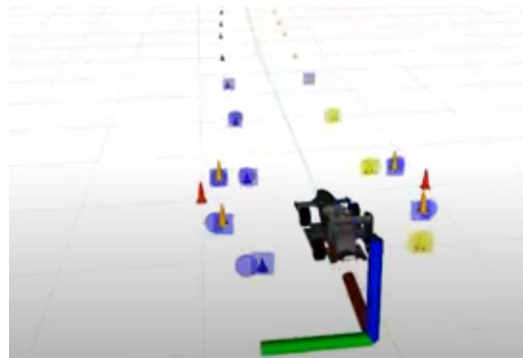


Figure 4.32 Robot operating system environment

4.8.3 Modular Decomposition

The object detection node and its topics in ROS:

The object detection and color recognition are the first **two nodes** which exists in the ROS framework and they take the input data from the sensor (mono camera) directly as explained in the block diagram and take the input data from the camera as a set of frames and the object detection module works on each frame (image) in the input set of frames, so this node takes the input from the Mono camera and produces the output as a set of detected cone batches for each frame (image) this output is published on the **bbox_det** topic as in the following Figure 4.33. This set of batches which are published on the **bbox_det** topic are used in the color detection node, then the color detection node publishes the boundary boxes with its colors on the **Frame_BBox_Color** topic which is used as an input to the next module (key points extraction ResNet)



Figure 4.33: ROS nodes and published topics of the object detection module

The 7 key points extraction node and its topics in ROS:

The “keypoint extraction DNN” is implemented and used via ROS’s rospy interface and the node is created on ROS to take the cone patches from the object detection module and for each patch it detects the 7 key points in the extract landmarks node and publish the 7 key points for each patch on the perception cones topic as shown in the Figure 4.34, and then the perception cone module integrates each cone in the frame with its extracted landmarks and apply PNP algorithm on it to extract its 3D position and then publishes the localized cones to the reactive mapping node to integrate the perception pipeline with the localization and mapping algorithm.

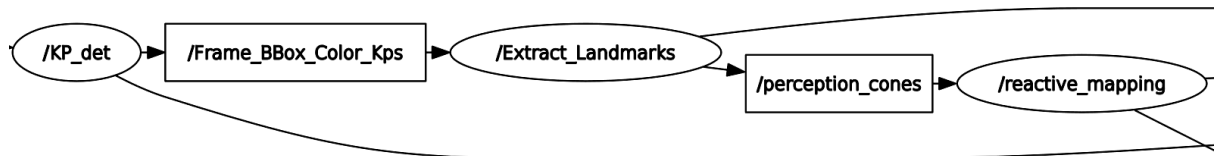
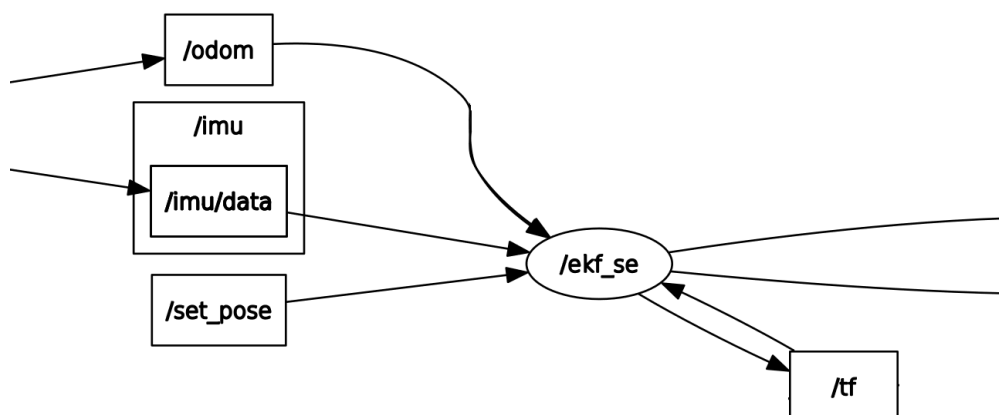


Figure 4.34: ROS nodes and published topics of the Key points extraction

Mapping and Localization to visualization:

For the localization and mapping nodes there is a node for the extended kalman filter which takes the IMU and wheel encoders sensors readings as shown in Figure 4.35 and by applying the EKF algorithm on it it publishes the filtered odometry to the odometry node which integrates the localization of the vehicle’s location with the mapping algorithm, the mapping algorithm subscribing two nodes which are the reactive mapping which have the cone position in the corresponding frame, then the global mapping algorithm subscribes to the odometry filtered node which publishes the odometry of the vehicle, the global mapping node publishes the landmarks locations and the vehicle location in the global map to the rViz node to visualize the map and landmarks in real time as shown in Figure 4.35



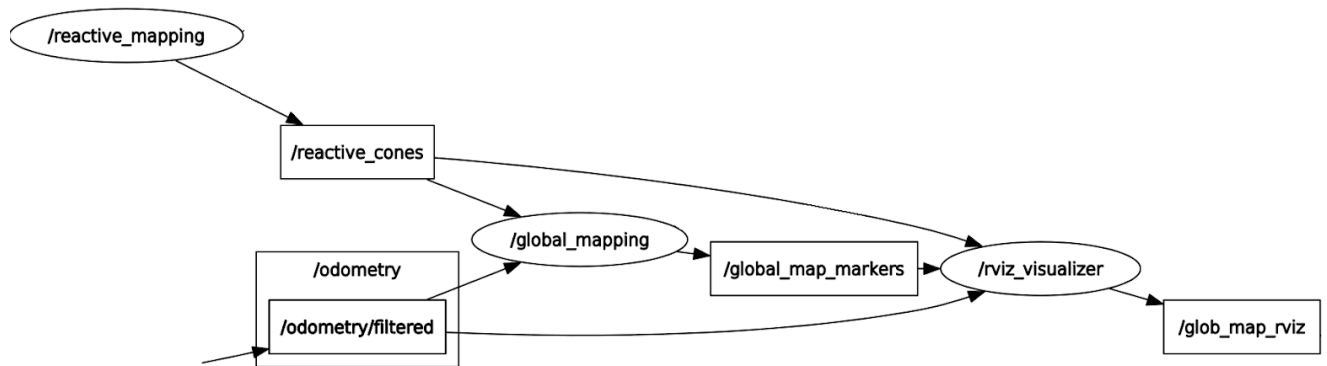
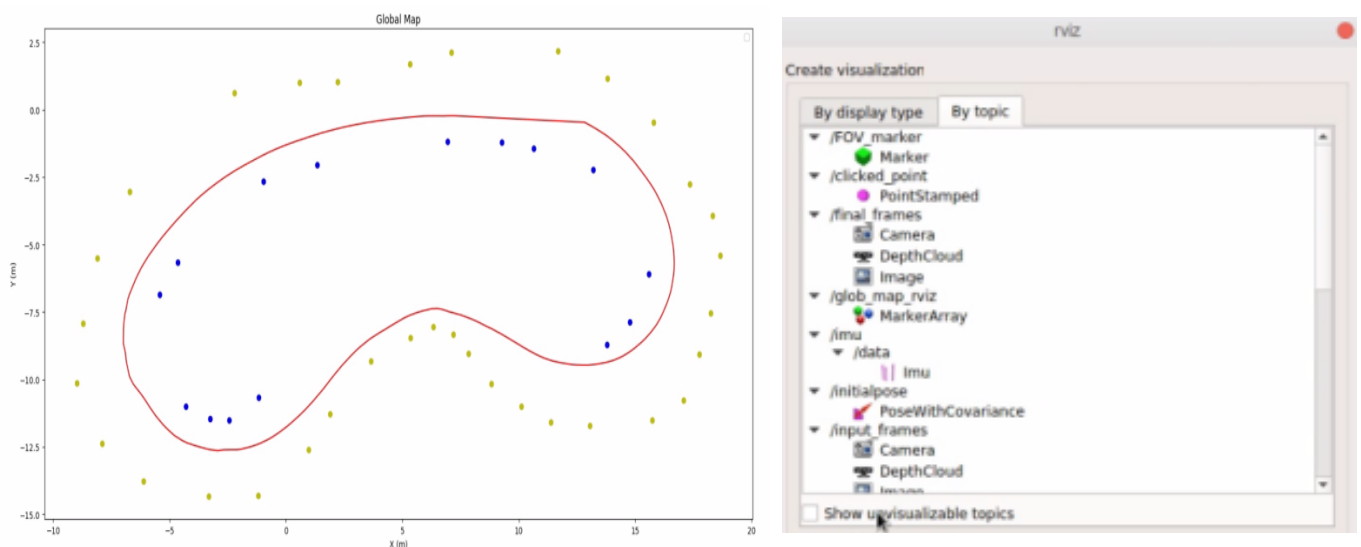


Figure 4.35: Rqt Graph of the ROS nodes and topics in mapping and localization algorithm connected to the node of ROS visualization rViz

All of these nodes and topics can be visualized on the ROS using rViz which can subscribe to any topic and shows its data in form of camera, image, pointcloud, and/or map, rViz is used to validate and test the pipeline outputs with the help of rosbags which described in chapter 5 in details. The following figure shows an example of the rViz tool.



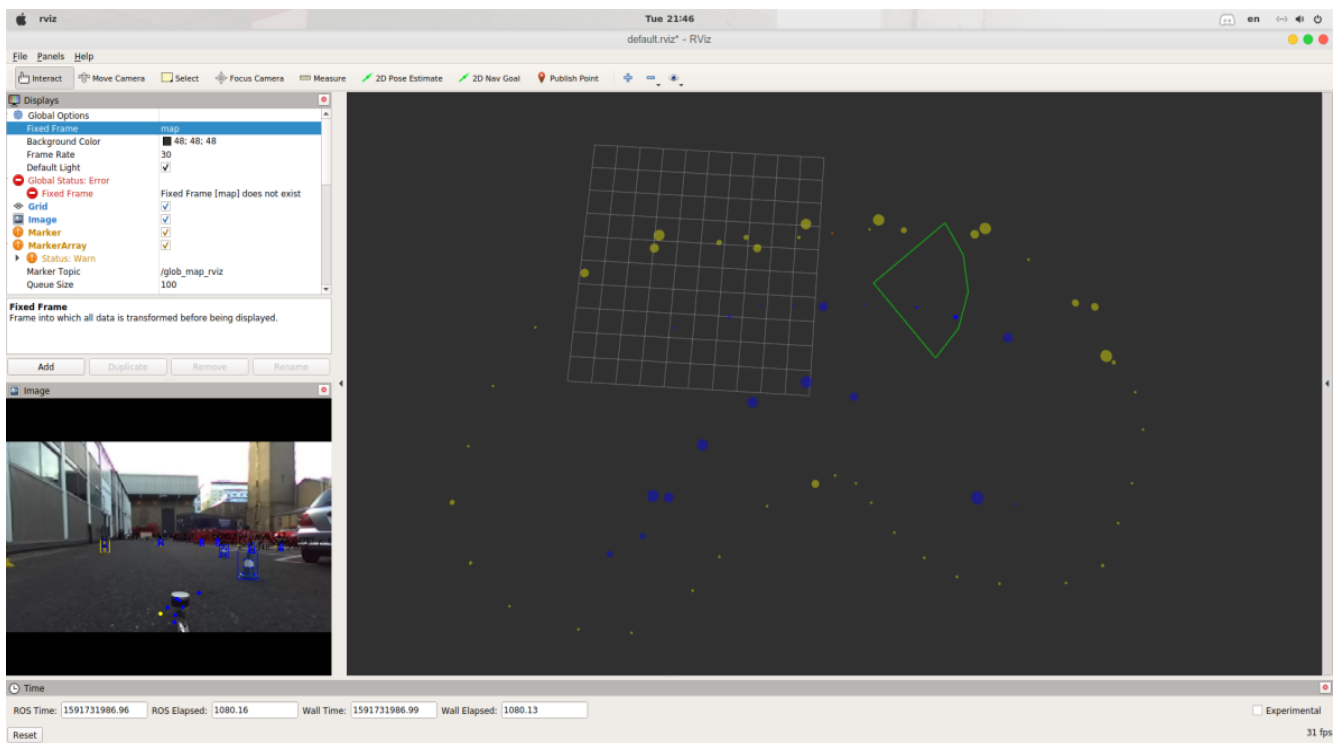


Figure 4.36 shows a sample from the rviz tool window

Chapter 5: System Testing and Verification

In order to test and verify our pipeline, all of the vehicle's modules were integrated on the ROS environment see chapter 4. The ROS environment helped us to visualize each module as a node with its inputs and outputs represented as ROS topics and messages, and using rviz the topics and messages of each module can be subscribed and visualized in the shape of 3D map, pointcloud, and/or graphs. The readings of each sensor can be visualized on rviz to help us to verify the pipeline by comparing the results of many sensors. Another shape of testing that will be shown in this chapter is the model testing using the testing and validation datasets and this type of testing exists in two modules, object detection and keypoints extraction modules because that each of these two modules have a deep learning model that's needed to be trained and tested using appropriate datasets. In this chapter we will describe in full details the testing setup, strategy and environment of our project. The Figure 5.2 shows an overview of the testing procedure and the track that we went with during the testing phase. Going from the perception pipeline which detects the position and color of each cone to mapping and localization of each cone on the track and building the global map of the track, ending with the control commands that controls the motion of the car in the given track path. Each of these modules was

tested individually and tested after integration with another module and so on, as shown in Figure 5.2 which summarizes the testing phase procedure. Also, this chapter aims to verify that our project reached its main goal as shown in Figure 5.1 by taking the testing procedure that is shown in the following page.

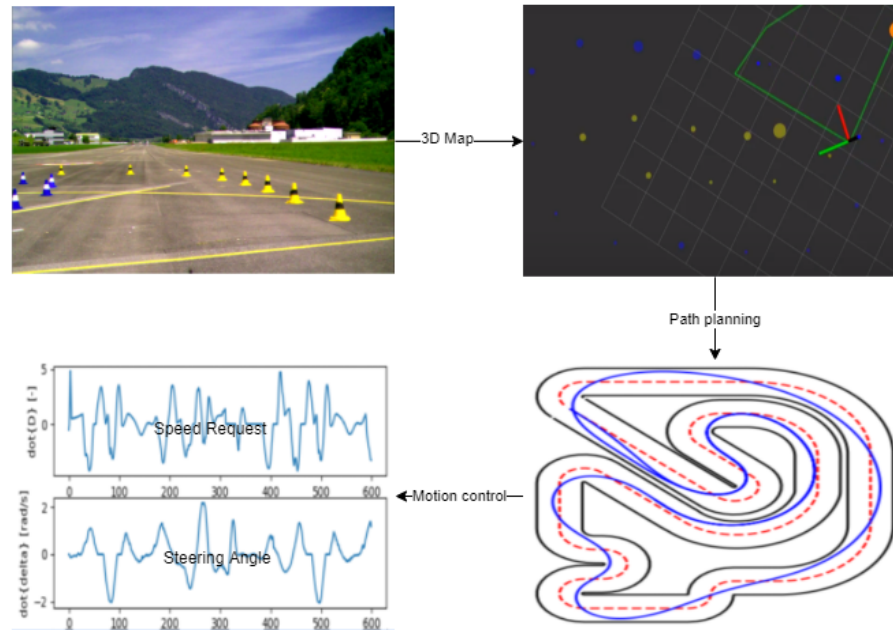


Figure 5.1: The Main Goal of the project

5.1. Testing Setup

First of all our plan was to deploy these modules and this pipeline on Cairo University Racing team (CURT) Electric Vehicle (EV) to test the pipeline in the real environment and on a real vehicle that already joined Formula Student UK competition 2019 as an Electric Vehicle. Our aim was to transform it into a Driverless Racing Vehicle (DV). But unfortunately due to the Covid-19 epidemic that we are facing in the last 5 months, our plan was totally changed from deploying these modules on our real vehicle to test our software modules by simulating these modules on Robot Operating System (ROS).

With the help of Formula Student Artificial Intelligence (FSAI) Competition, which provided us with some real time data in the form of **rosbags** which includes some of real time sensors' readings like Monocular camera frames, stereo camera (left and right) frames, LIDAR, wheel odometry, and IMU data(as shown in the following figure). We used these data in our testing phase by taking the required inputs to the modules and testing the behaviour of the pipeline according to these real time data. All of these works were done on the ROS environment and using the ROS tools to

Table 5.1: Rosbags topics and input sensors' data

| Topic | Type | Freq | Source | Notes |
|------------------------------------|-----------------------------|------|-----------------|--|
| /imu/data | sensor_msgs/Imu | 10Hz | X-Sens MTi-G | 6 DoF IMU Data from X-Sens MTi-G INS with embedded Kalman Filter. Includes orientation data. |
| /imu/mag | sensor_msgs/MagneticField | 10Hz | X-Sens MTi-G | |
| /fix | sensor_msgs/NavSatFix | 10Hz | X-Sens MTi-G | Unreliable as far as our tests go. |
| /velodyne_points | sensor_msgs/PointCloud2 | 10Hz | Velodyne VLP-16 | Full 360 degree data. Testbed must be filtered out. |
| /left/image_rect_color/compressed | sensor_msgs/CompressedImage | 30Hz | ZED Camera | 672 x 376 resolution |
| /right/image_rect_color/compressed | sensor_msgs/CompressedImage | 30Hz | ZED Camera | 672 x 376 resolution |
| /odom | nav_msgs/Odometry | 30Hz | ZED Camera | Visual odometry. Very good!! |
| voxel_frid/output | sensor_msgs/PointCloud2 | 30Hz | ZED Camera | Slightly filtered 3D pointcloud from the stereo camera. |
| /tf | tf2_msgs/TFMessage | 30Hz | - | Contains odom->base_link published by the ZED camera |
| /tf_static | tf2_msgs/TFMessage | once | - | Transforms for all sensors on the car |

visualize the inputs and outputs of the pipeline and modules the most important ROS tool is **rViz** which we used to visualize, simulate, and test our work using it.

We have collected more than 10GB of real time sensors' readings of a real formula student Driverless Vehicle (DV) on a track which is shown in the above figure, and we use these data to test and visualize the behaviour of our modules and validate it in the real environment. All of these datasets come in a rosbag format which can be used directly in ROS. Another strategy of testing that we used in our project in the module testing is the testing datasets which exist in the modules which are based on deep learning models or machine learning algorithms In the following part we will talk in full details about the testing of each module according to its type and the integration testing which based on the ROS environment.

5.2. Testing Plan and Strategy

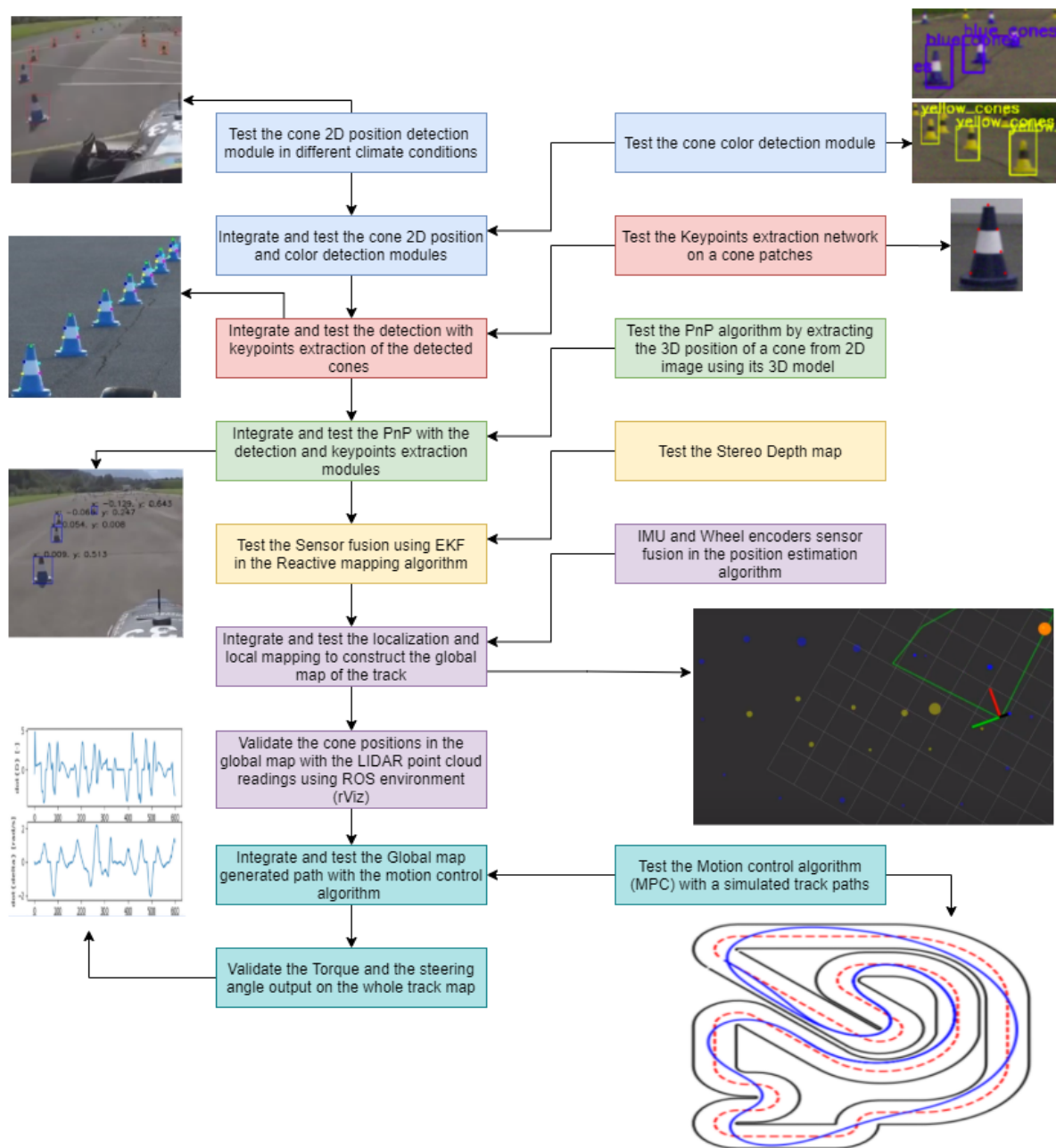


Figure 5.2: Testing Setup and Procedure overview

As explained in the testing setup Figure 5.2 the testing process which is applied on the pipeline is for each module and integrated with the other modules step by step to the end of the pipeline as shown in the testing procedure in Figure 5.2 in the following subsections each block in the figure above will be tested and integrated with the corresponding blocks.

5.2.1 Testing the cone 2D localization module -Customized YOLOv3

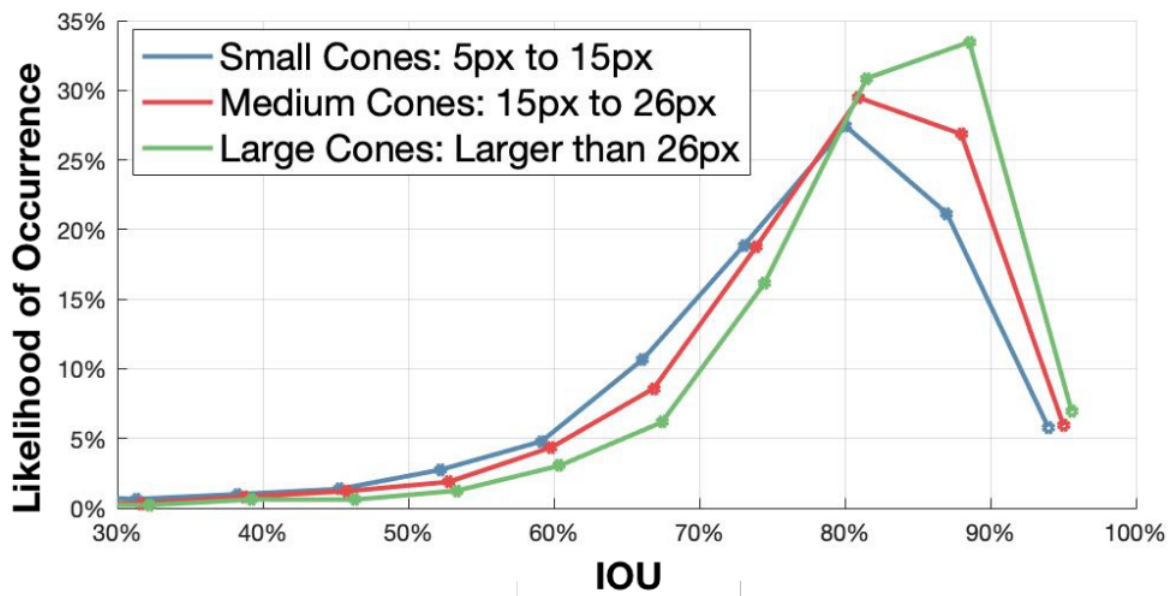


Figure 5.3: Intersection-over-union - examined detection accuracy across three landmark sizes

To characterize accuracy of the localization phase of the pipeline, we examined detection accuracy across three landmark sizes. The results are shown in Table 5.2. We use a widely-used metric, intersection-over-union (IoU), that measures alignment of our bounding box with ground truth [30]. We examined IoU across >24,000 landmarks. We achieve a median IoU of 88% for large cones, and 83-84% for smaller cones. This bounding box tightness enables whole system localization accuracy described next.

Our final accuracy metrics for detecting traffic cones on the racing track:

Table 5.2: The results of detection accuracy

| mAP | Recall | Precision |
|------------|---------------|------------------|
| 89.35% | 92.77% | 86.94% |

Testing the detection module in different climate conditions:

1) Sunny



Boundry box detection



Detected Cone batches



Figure 5.4: Testing the detection module in sunny climate condition

2) High light



Cone boundary boxes



Detected cones batches



Figure 5.5: Testing the detection module in high light climate condition

3) Cloudy



Figure 5.6: Testing the detection module in cloudy climate condition

4) Low light

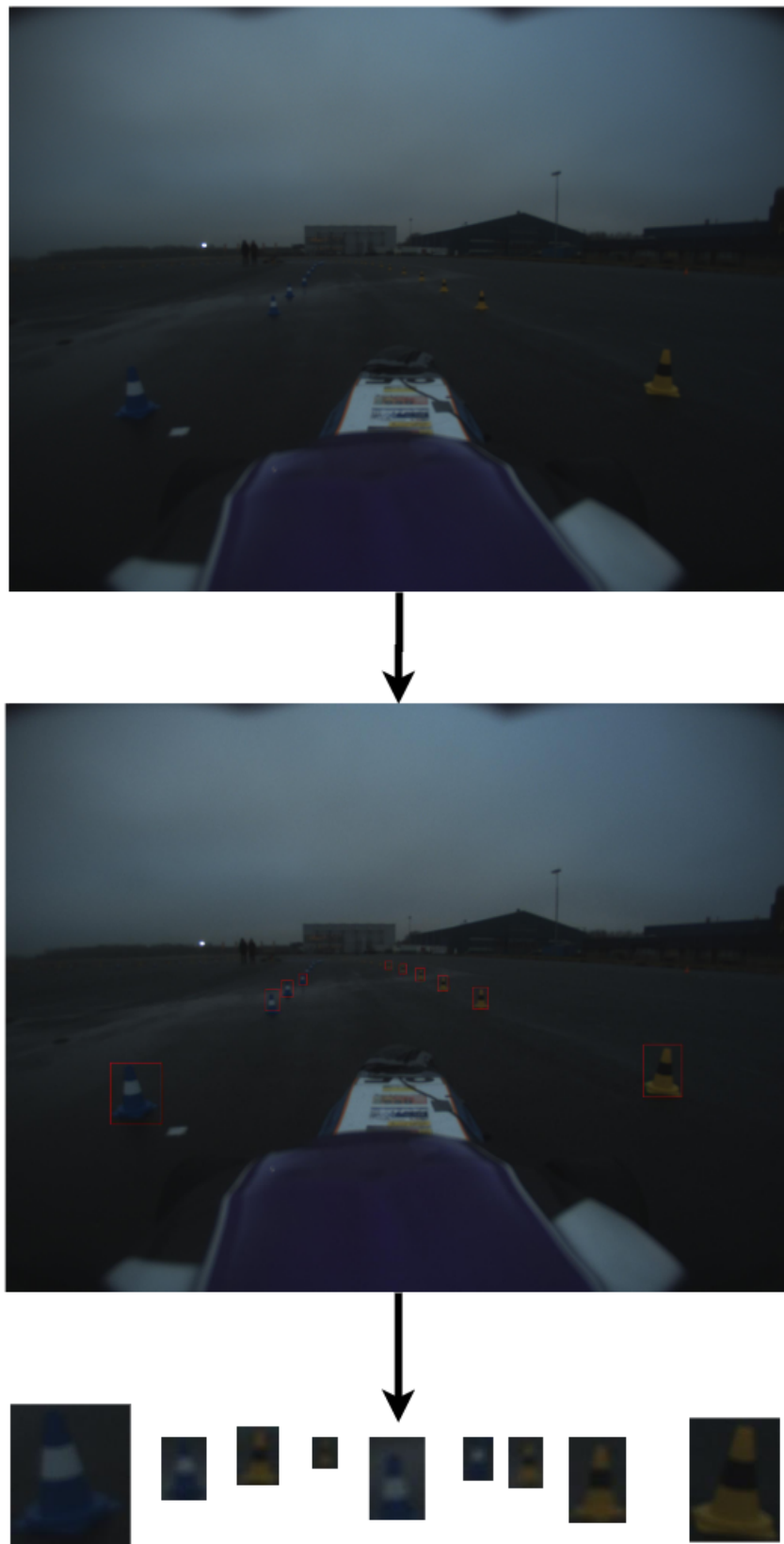


Figure 5.7: Testing the detection module in low light climate condition

5) Rainy



Figure 5.8: Testing the detection module in rainy climate condition

5.2.1 Testing the Cone Color Detection module

The following figure shows the testing metrics for multiple saturated epochs compared to the ground truth labels.

| | | | | | |
|--------|----------|-------|-------|---------|----------|
| GIoU | obj | cls | total | targets | img_size |
| 1.53 | 0.357 | 0.288 | 2.18 | 20 | 608: |
| Images | Targets | P | R | mAP@0.5 | F1: |
| 637 | 5.94e+03 | 0.647 | 0.81 | 0.776 | 0.717 |
| GIoU | obj | cls | total | targets | img_size |
| 1.48 | 0.355 | 0.293 | 2.13 | 15 | 576: |
| Images | Targets | P | R | mAP@0.5 | F1: |
| 637 | 5.94e+03 | 0.647 | 0.824 | 0.794 | 0.724 |
| GIoU | obj | cls | total | targets | img_size |
| 1.5 | 0.344 | 0.281 | 2.12 | 41 | 320: |
| Images | Targets | P | R | mAP@0.5 | F1: |
| 637 | 5.94e+03 | 0.613 | 0.849 | 0.805 | 0.711 |
| GIoU | obj | cls | total | targets | img_size |
| 1.46 | 0.348 | 0.279 | 2.09 | 19 | 608: |
| Images | Targets | P | R | mAP@0.5 | F1: |
| 637 | 5.94e+03 | 0.682 | 0.826 | 0.804 | 0.746 |

Figure 5.9: Color detection module testing metrics on the saturated epochs of training.

The mean average precision mAP of color recognition for confidence which is more than 50% is about 80.

Testing the color detection with object detection:

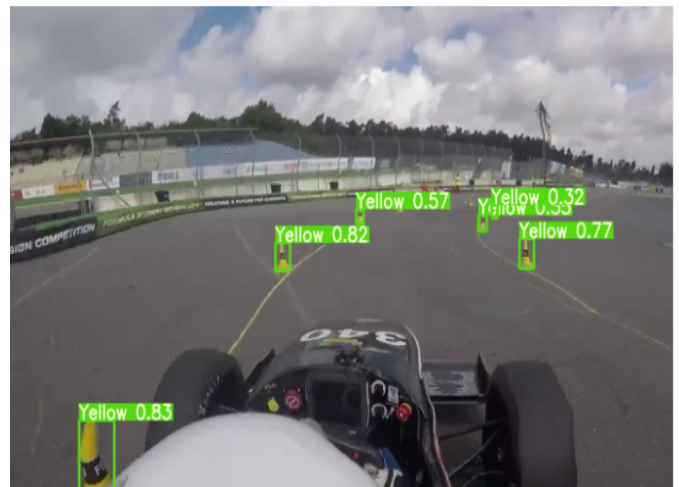


Figure 5.10: Testing the color detection with object detection

5.2.3 7 Key Points Extraction Network

1) validation test

The feature extraction network was trained with 3.2k images 2720 for training and about 480 for validation as shown in the figure:

training image number: 2718
validation image number: 479
Including geometric loss: True
Loss type: l2_softargmax



Figure 5.11: Key points extraction network dataset

By training on this dataset the best loss, loss: location/geometric/total which is loss: 0.011191/0.0/0.011191 was found in EPOCH 48 as shown in the following figure.

```
EPOCH 47
  Training: MSE/Geometric/Total Loss: 0.0211922256/0.0/0.0211922256
  Starting validation...
  Validation: MSE/Geometric/Total Loss: 0.018063846/0.0/0.018063846
  Saving model to outputs/april-2020-experiments/testTwo/47_loss_0.02.pt
EPOCH 48
  Training: MSE/Geometric/Total Loss: 0.0170994289/0.0/0.0170994289
  Starting validation...
  Validation: MSE/Geometric/Total Loss: 0.0111918962/0.0/0.0111918962
  Saving ONNX model to outputs/april-2020-experiments/testTwo/best_keypoints_8080.onnx
EPOCH 49
  Training: MSE/Geometric/Total Loss: 0.0156765056/0.0/0.0156765056
  Starting validation...
  Validation: MSE/Geometric/Total Loss: 0.0131025789/0.0/0.0131025789
EPOCH 50
  Training: MSE/Geometric/Total Loss: 0.0152619437/0.0/0.0152619437
  Starting validation...
  Validation: MSE/Geometric/Total Loss: 0.0132306492/0.0/0.0132306492
EPOCH 51
  Training: MSE/Geometric/Total Loss: 0.0149801664/0.0/0.0149801664
  Starting validation...
  Validation: MSE/Geometric/Total Loss: 0.0115105509/0.0/0.0115105509
  Saving model to outputs/april-2020-experiments/testTwo/51_loss_0.01.pt
EPOCH 52
  Training: MSE/Geometric/Total Loss: 0.01526335/0.0/0.01526335
  Starting validation...
  Validation: MSE/Geometric/Total Loss: 0.0127581186/0.0/0.0127581186
EPOCH 53
  Training: MSE/Geometric/Total Loss: 0.0146441231/0.0/0.0146441231
  Starting validation...
  Validation: MSE/Geometric/Total Loss: 0.0123967328/0.0/0.0123967328
EPOCH 54
  Training: MSE/Geometric/Total Loss: 0.0137630213/0.0/0.0137630213
  Starting validation...
  Validation: MSE/Geometric/Total Loss: 0.0126291188/0.0/0.0126291188
EPOCH 55
  Training: MSE/Geometric/Total Loss: 0.0138158908/0.0/0.0138158908
  Starting validation...
  Validation: MSE/Geometric/Total Loss: 0.0123110787/0.0/0.0123110787
  Saving model to outputs/april-2020-experiments/testTwo/55_loss_0.01.pt
EPOCH 56
  Training: MSE/Geometric/Total Loss: 0.0134418112/0.0/0.0134418112
  Starting validation...
  Validation: MSE/Geometric/Total Loss: 0.0131435319/0.0/0.0131435319
  Training is stopped due; loss no longer decreases. Epoch 48 is has the best validation loss.
```

Epoch with lower loss

End of training

Figure 5.12: Key points extraction network's best loss result

2) Visual test

Samples for the key points extraction on different cone batches and the heat map which is the output of the last layer (7 Conv) which gives 7 heat maps one for each point:

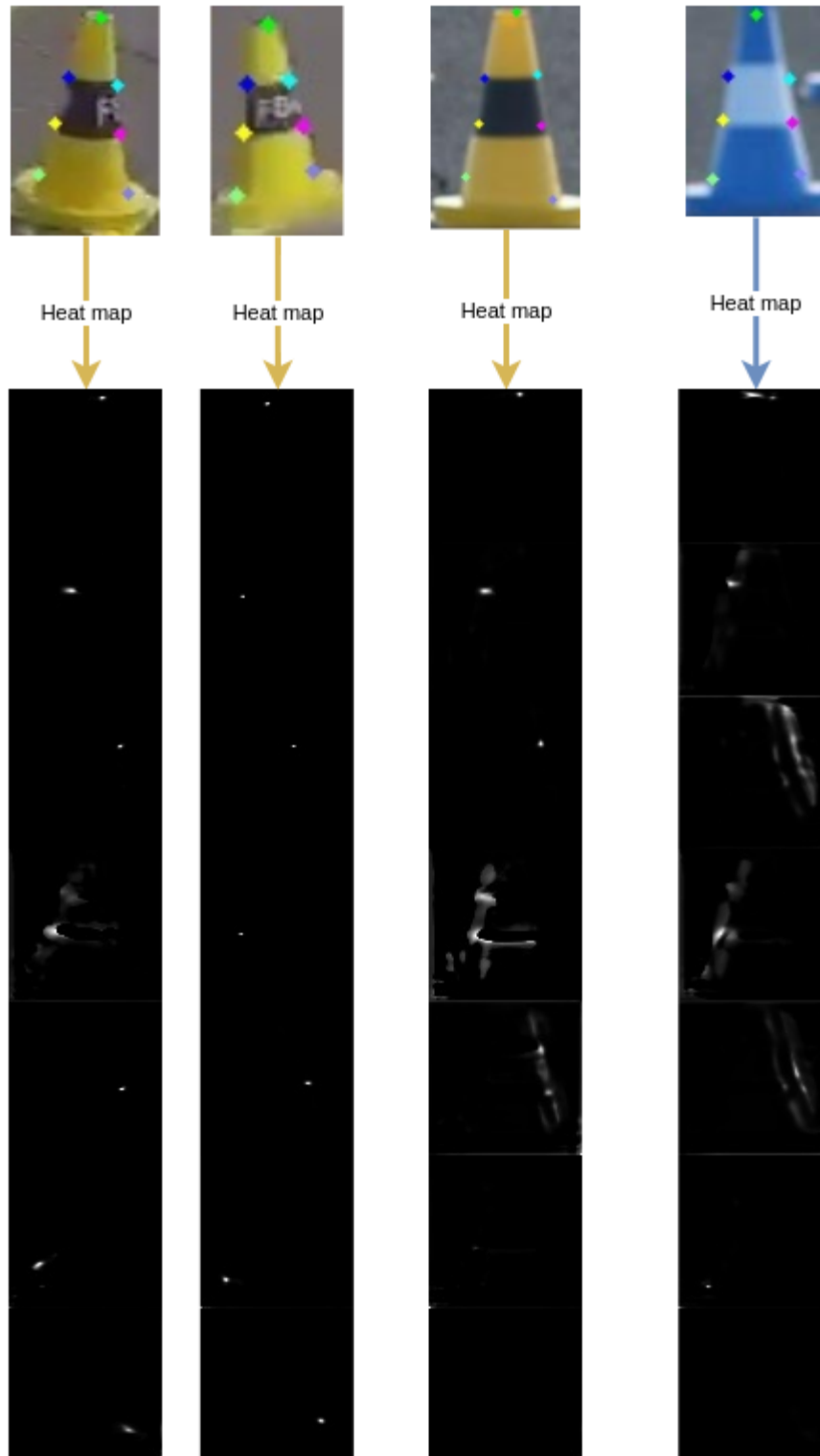


Figure 5.13: Key points extraction network's results and points' heatmap

5.2.4 Testing 3D space localization of cone:

The recorded data is from Rosbag with the given information about the camera parameters which is given in the Camera_Info topic, by using these camera parameters with the prior knowledge of the 3D cone model as shown in the following Figure 5.14, we tested our PnP algorithm to estimate the 3D cone position with respect to the vehicle's center. All of these simulations and data from Rosbags are tested and visualized on the rViz tool, on the Robot Operating System (ROS) environment as shown in Figure 5.14.

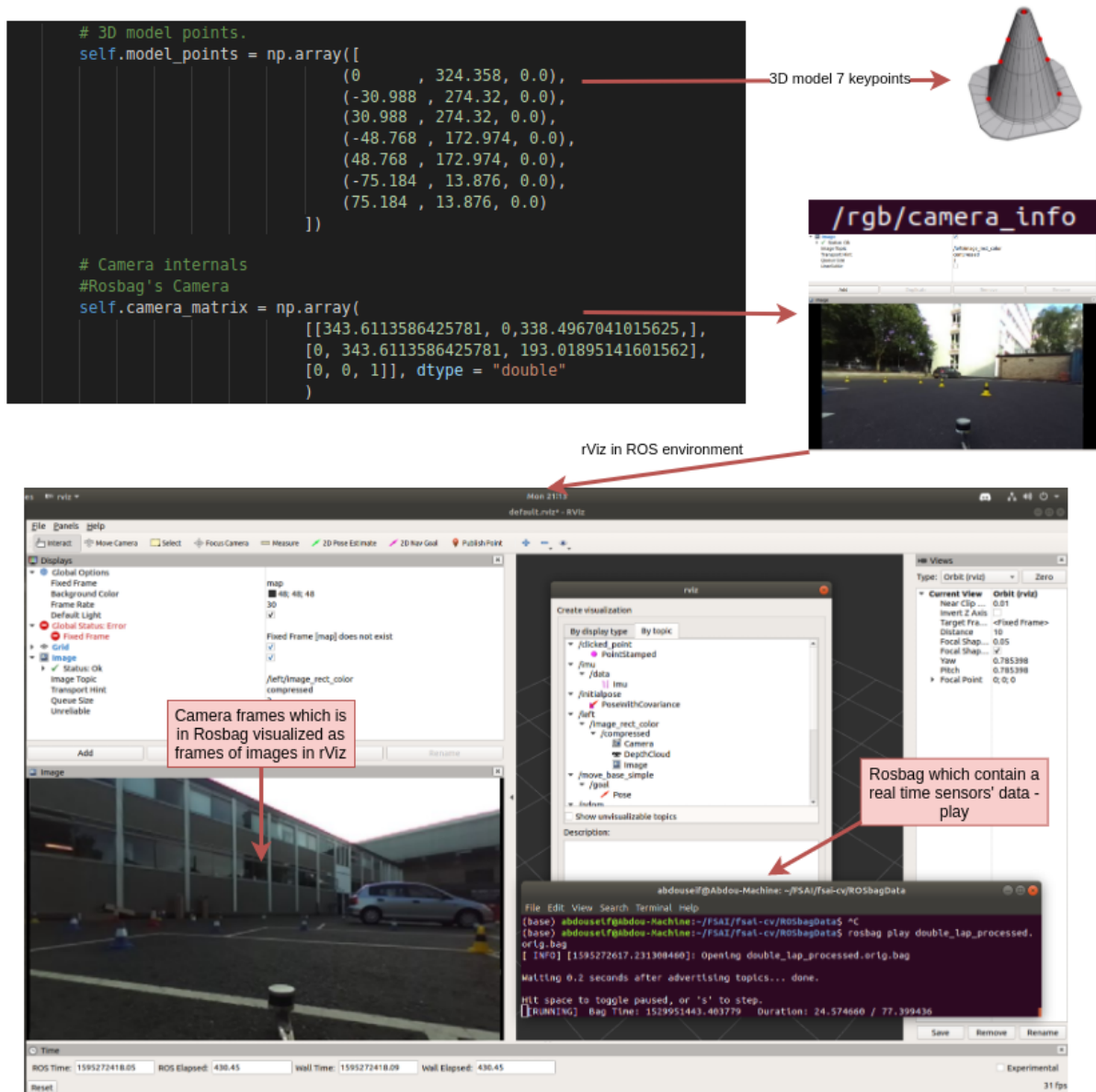


Figure 5.14: 3D Cone points, camera information, visualization of the rosbag in rViz

To test this module it must be tested with the existence of the detection and keypoints extraction, so the PnP algorithm will be applied on the extracted 7 key points (7 features) and then it estimates the cone 3D position in the world coordinates as shown in the following figures the 3D positions of each detected cone exists on the top of the batch square with respect to the car center.

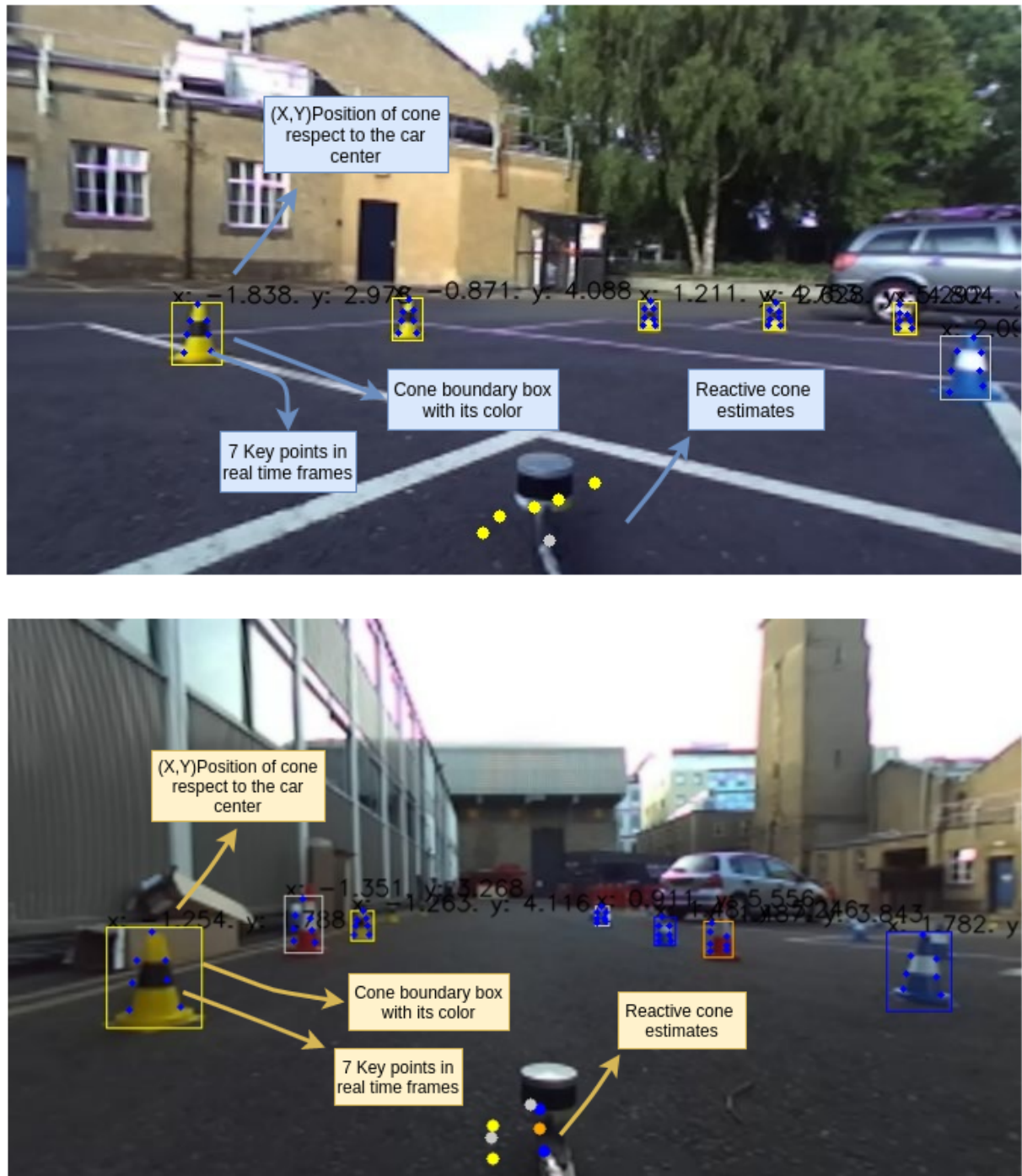


Figure 5.15: Testing the 3D localization PnP algorithm with the detection and key points extraction modules

5.2.5 Testing EKF Robot Localization and Mapping Modules

□ Testing EKF Robot Localization module

In order to test our EKF localization module we needed to get a superior source of vehicle locations, so we used the data found by the team to include GPS vehicle locations to indicate a performance measure to our module, then we tested the EKF Robot Localization by computing the mean average euclidean distance between vehicle pose estimations and the GPS location computed. The redline shows the vehicle trajectory of pose estimated through visual odometry, IMU, and wheel encoders, while the green line shows the gps locations.

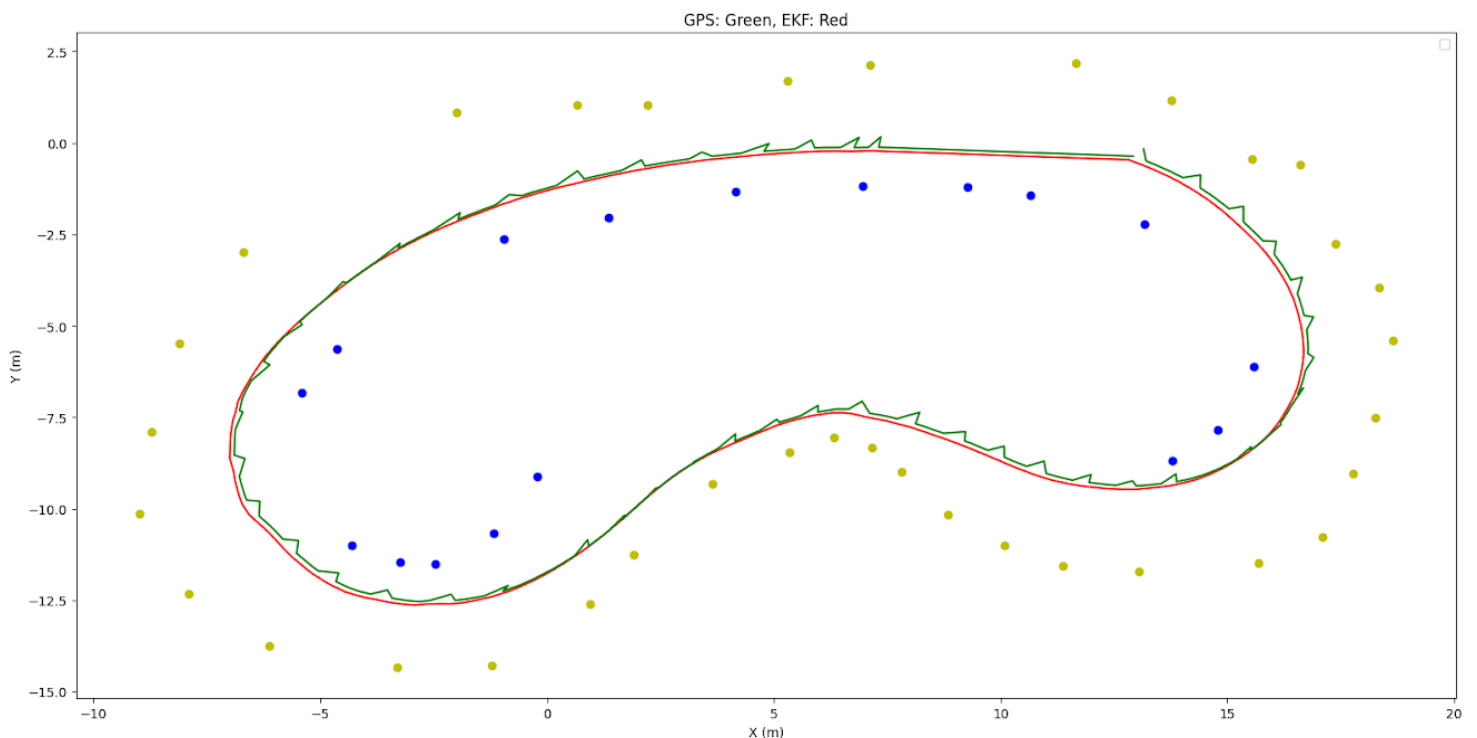


Figure 5.16: Difference between GPS positioning (green) and EKF pose estimation position (red).

It's clear that GPS has this spiky noise behaviour because of its high latency in receiving vehicle positions, but the average of these locations will serve as a measure for our results. Our results seem more smooth and robust, with low mean average difference with the GPS locations, as you can see from the next table.

Table 5.3: X, Y locations received from both EKF Localization module and GPS positions

| EKF Estimates | | GPS | | EKF Estimates | | GPS | | EKF Estimates | | GPS | |
|---------------|------------|--------------|--------------|---------------|------------|--------------|--------------|---------------|------------|--------------|--------------|
| X | Y | X | Y | X | Y | X | Y | X | Y | X | Y |
| 12.751588 | -0.4392151 | 13.055252725 | -0.046222706 | 16.65253 | -5.2076044 | 16.75253067 | -5.107604408 | 14.166345 | -9.258035 | 14.266344643 | -9.158034706 |
| 13.077087 | -0.5831773 | 13.177087402 | -0.483177328 | 16.673521 | -5.4896235 | 16.773521042 | -5.389623547 | 14.010949 | -9.306141 | 14.110949135 | -9.2061409 |
| 13.325321 | -0.7083315 | 13.425321198 | -0.608331525 | 16.680456 | -5.701807 | 16.780456161 | -5.601807022 | 13.780231 | -9.363036 | 13.880231476 | -9.263036156 |
| 13.697774 | -0.9150184 | 13.797773933 | -0.81501838 | 16.675627 | -5.8866587 | 16.775626755 | -5.786658669 | 13.653424 | -9.386725 | 13.910167525 | -9.176981299 |
| 13.875209 | -1.0268552 | 13.975208855 | -0.92685523 | 16.649448 | -6.1487155 | 16.967140161 | -5.891711962 | 13.424148 | -9.422302 | 13.524147606 | -9.322302246 |
| 14.142914 | -1.2121458 | 14.517450152 | -0.886918553 | 16.608603 | -6.3709664 | 16.708602524 | -6.270966434 | 13.290972 | -9.438931 | 13.390971756 | -9.338931465 |
| 14.309564 | -1.3428967 | 14.409563637 | -1.2428967 | 16.577728 | -6.5394216 | 16.677728271 | -6.439421558 | 13.003729 | -9.46968 | 13.103728867 | -9.369679832 |
| 14.577298 | -1.5613768 | 14.677298164 | -1.46137681 | 16.527979 | -6.7417746 | 16.627978897 | -6.641774559 | 12.903445 | -9.471548 | 13.003445244 | -9.37154808 |
| 14.876035 | -1.8427087 | 14.976034737 | -1.742708707 | 16.44989 | -6.966886 | 16.549890137 | -6.866886044 | 12.717529 | -9.470927 | 13.086887761 | -9.131411971 |
| 14.911039 | -1.8811067 | 15.011039352 | -1.781106734 | 16.400328 | -7.0913506 | 16.603525064 | -6.865885592 | 12.429993 | -9.460288 | 12.529992676 | -9.360288048 |
| 15.028656 | -2.007513 | 15.347923211 | -1.718022494 | 16.299997 | -7.3114476 | 16.39999733 | -7.21144762 | 12.277549 | -9.444866 | 12.37754879 | -9.34486618 |
| 15.273092 | -2.2815325 | 15.37309227 | -2.181532526 | 16.246761 | -7.4301147 | 16.346761322 | -7.330114746 | 12.004944 | -9.4121895 | 12.104943848 | -9.312189484 |
| 15.479443 | -2.5255563 | 15.579442596 | -2.425556326 | 16.116135 | -7.670271 | 16.216134644 | -7.57027092 | 11.869444 | -9.392286 | 11.969443893 | -9.292286301 |
| 15.56878 | -2.6360118 | 15.668779945 | -2.536011839 | 16.06361 | -7.7544 | 16.163610077 | -7.654399776 | 11.62757 | -9.348621 | 11.886860932 | -9.872340025 |
| 15.69319 | -2.7904599 | 15.793189621 | -2.690459871 | 15.929611 | -7.9518595 | 16.172662072 | -7.566824719 | 11.506348 | -9.326769 | 11.606347656 | -9.226768875 |
| 15.889088 | -3.0680988 | 16.202625166 | -2.78847146 | 15.847659 | -8.056123 | 15.947659111 | -7.95612278 | 11.208726 | -9.245402 | 11.308725929 | -9.145402336 |
| 15.969961 | -3.1964126 | 16.069961166 | -3.096412563 | 15.668716 | -8.256372 | 15.768716431 | -8.156372452 | 11.070266 | -9.20077 | 11.17026577 | -9.100770378 |
| 16.122475 | -3.4726918 | 16.22247467 | -3.372691774 | 15.573828 | -8.351734 | 15.673827744 | -8.251734161 | 10.825365 | -9.108476 | 10.925365067 | -9.008475685 |
| 16.165636 | -3.5649748 | 16.265636063 | -3.464974785 | 15.395264 | -8.521927 | 15.495263672 | -8.42192688 | 10.7027855 | -9.0539665 | 11.100188287 | -8.817639315 |
| 16.315325 | -3.879207 | 16.415324783 | -3.779206896 | 15.302329 | -8.599032 | 15.509555232 | -8.306503256 | 10.473968 | -8.946174 | 10.573967552 | -8.846173668 |
| 16.365702 | -4.0090537 | 16.738784074 | -3.761343935 | 15.071908 | -8.772785 | 15.171907997 | -8.672785187 | 10.333625 | -8.876646 | 10.43362484 | -8.776646042 |
| 16.442463 | -4.223333 | 16.542462921 | -4.123332882 | 14.966802 | -8.848421 | 15.066801643 | -8.748421097 | 10.136366 | -8.77742 | 10.236365891 | -8.677420044 |
| 16.488127 | -4.3801494 | 16.588126755 | -4.280149364 | 14.773539 | -8.971962 | 14.873538589 | -8.871961975 | 9.962409 | -8.681781 | 10.062409019 | -8.581780815 |
| 16.536757 | -4.5645895 | 16.636756516 | -4.4645895 | 14.675985 | -9.026952 | 14.775985336 | -8.92695179 | 9.763115 | -8.582067 | 10.075053975 | -8.350312716 |
| 16.587595 | -4.8226833 | 16.687594986 | -4.722683334 | 14.468397 | -9.13629 | 14.71123663 | -8.881472683 | 9.601928 | -8.496126 | 9.7019277573 | -8.396126175 |
| 16.62362 | -4.989852 | 16.850801112 | -4.757699141 | 14.33721 | -9.193175 | 14.437209702 | -9.093175316 | 9.365766 | -8.375156 | 9.4657655716 | -8.275156403 |

The mean average euclidean distance is evaluated to be about 0.13669 m. This difference is acceptable as it does not exceed 10% of the average width of a vehicle, and 3% of an average track width.

❑ K-means-based global mapping module

In order to test our complete mapping system we have used another piece of data sent in the rosbags, the velodyne points of the cones (velodyne is a 3D laser scanner that scans the surrounds of the vehicle and map it directly in reference to the vehicle position) we will compare the cone locations in the global map from the 3D laser scanner (Velodyne) and our simple monocular camera pipeline, this will show the performance of our simple approach and the laser scanner taking into consideration the cost of each system and the logistic difficulties of the velodyne.

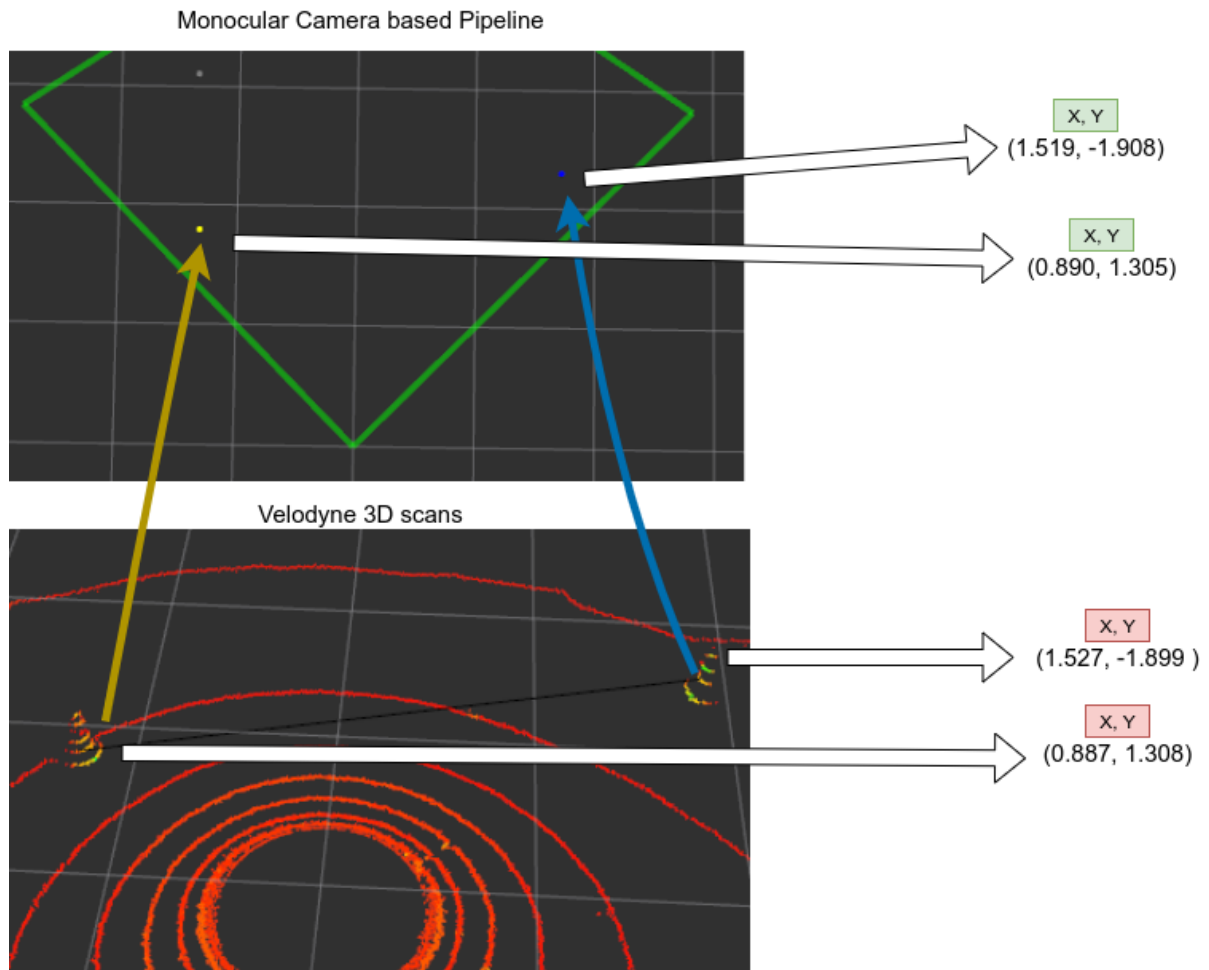


Figure 5.17: Visualizing the difference between Monocular Mapping pipeline and velodyne 3D scans. |

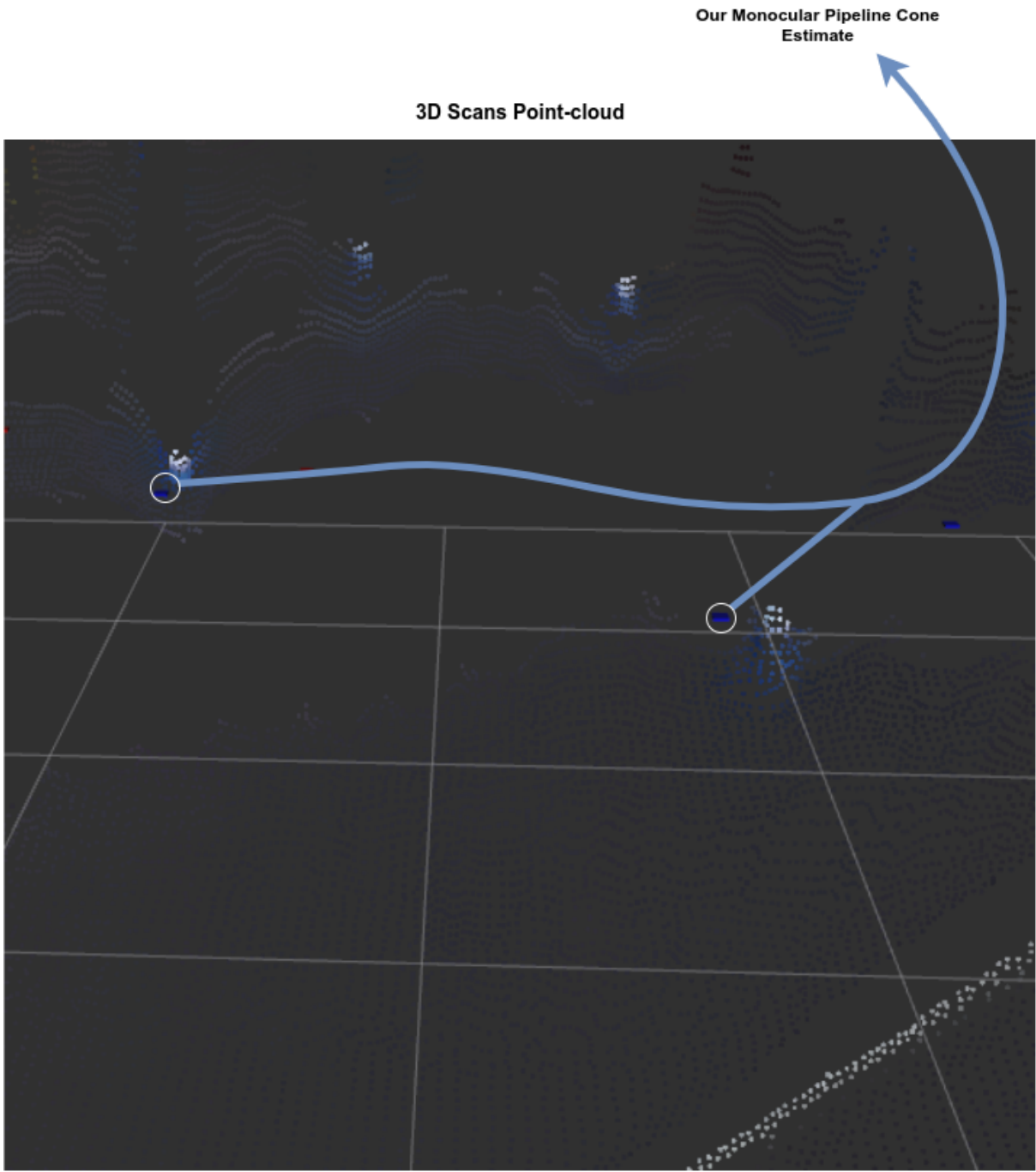


Figure 5.18: Visualizing the difference between our pipeline and the 3D scanned point-cloud. ||

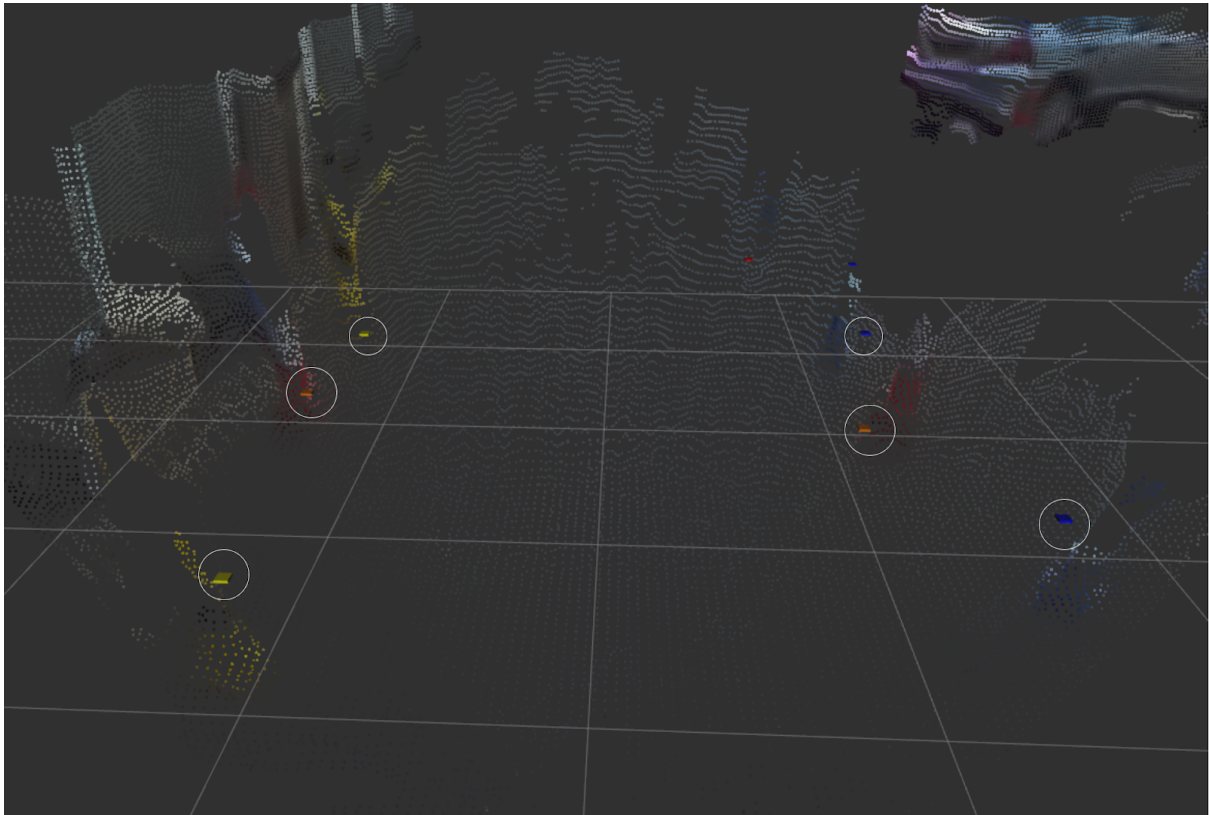


Figure 5.19: Visualizing the difference between our pipeline and the 3D scanned point-cloud. |||

The difference is averaged between many samples and it seemed it doesn't exceed 0.15m inside the FOV which is very impressive results.

5.2.5 Testing Model Predictive Controller

Now we have the map of the track from the localization and mapping algorithm and tested on a real time sensor data from the ROSbag and we built the track and validated it, so it's the time for motion control testing on this given track map as shown in the following figure the chosen path for the path planning algorithm is the simplest path at the center of the estimated track map as shown in the figure below.

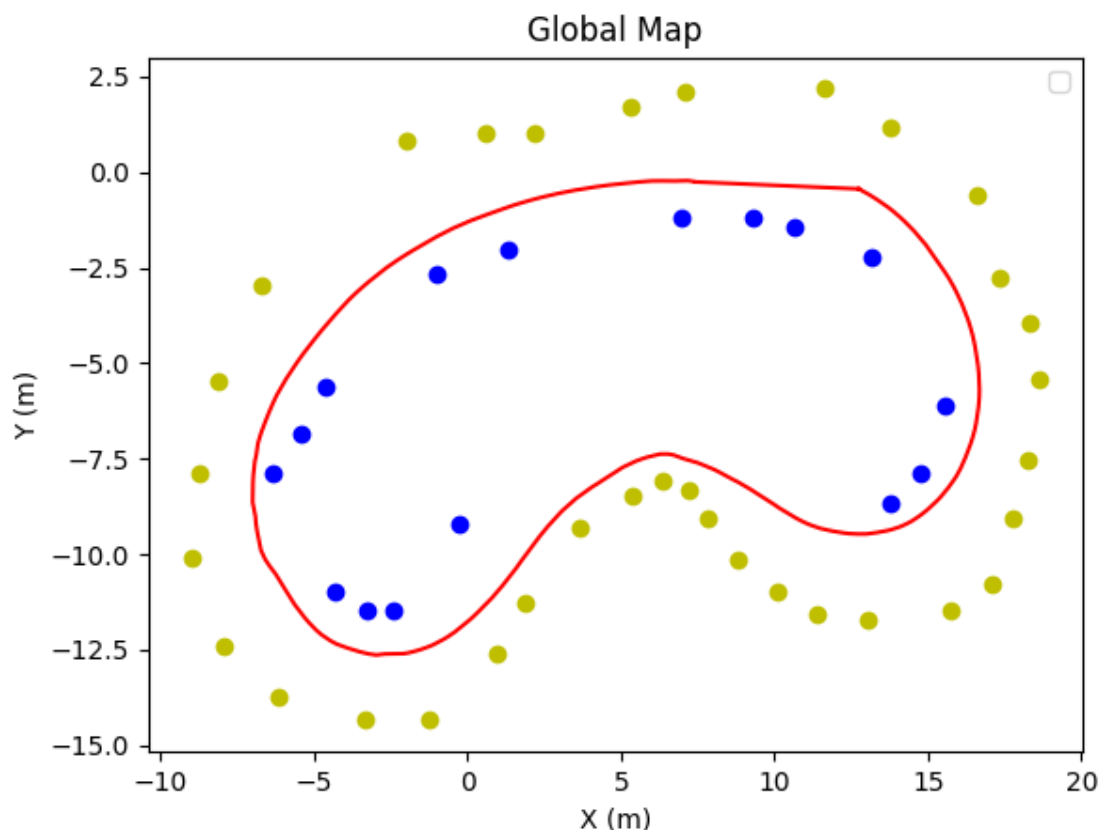


Figure 5.20: Simplest path at the center of the estimated track map from the mapping algorithm

1) Path tracking

The motion control algorithm will be tested by sending this path to it and evaluating how much the generated path from the motion control algorithm will match the given track, based on the vehicle model and track width. The following figure shows how much the MPC is close to the generated path from the mapping and path planning algorithm.

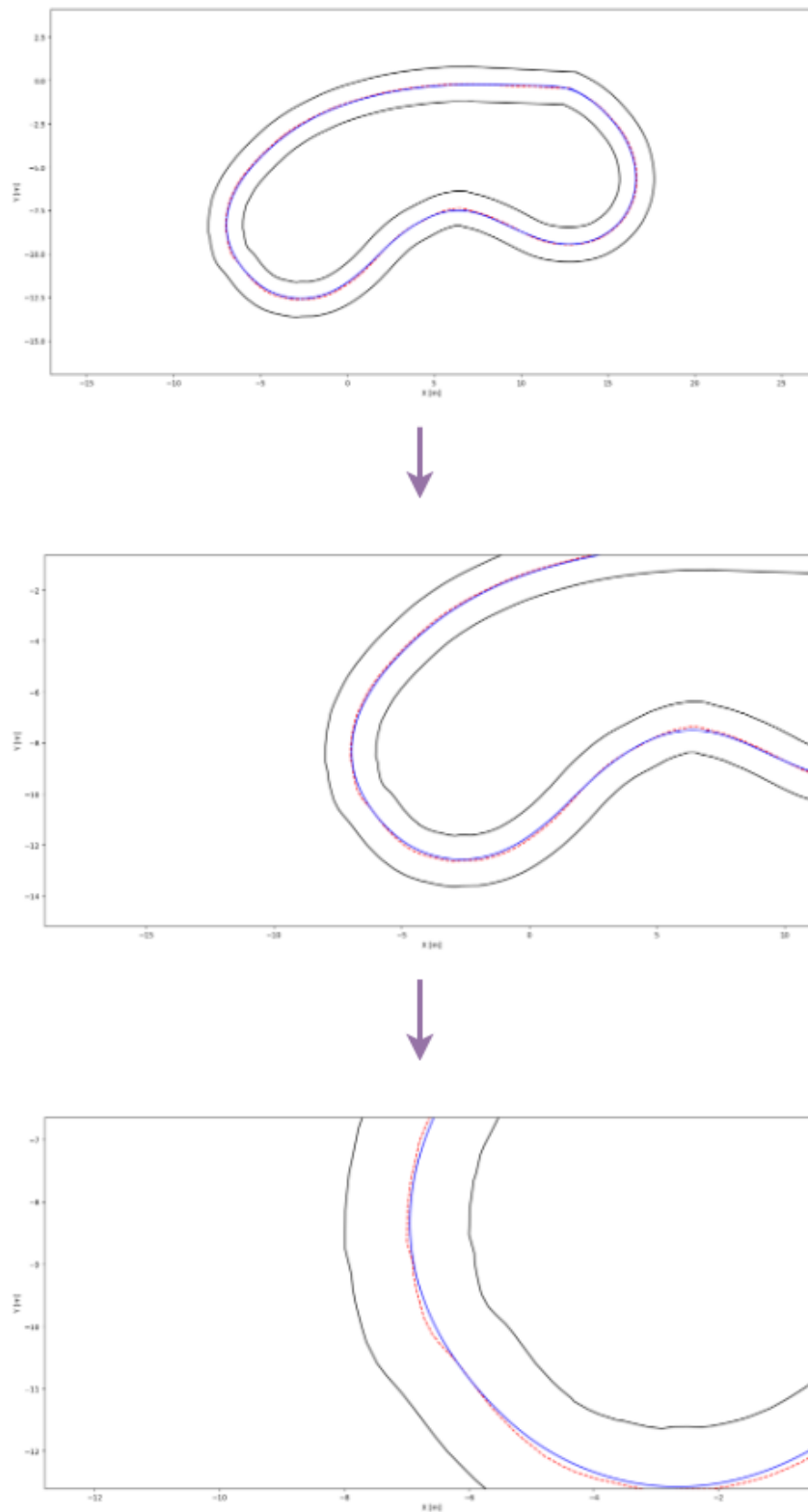


Figure 5.21: The actual track which produced by the MPC algorithm with the estimated map track

2) MPC control outputs

After receiving the vehicle trajectory the MPC outputs the control parameters estimated to race the vehicle along the track, these estimated parameters are estimated by optimizing a cost function over the motion model as illustrated in section 4.7. The following is an example of the manipulated and state variables that resulted in our simulations. Further testing will be needed in reality, but the simulated output is as expected to build a competent racing autonomous vehicle.

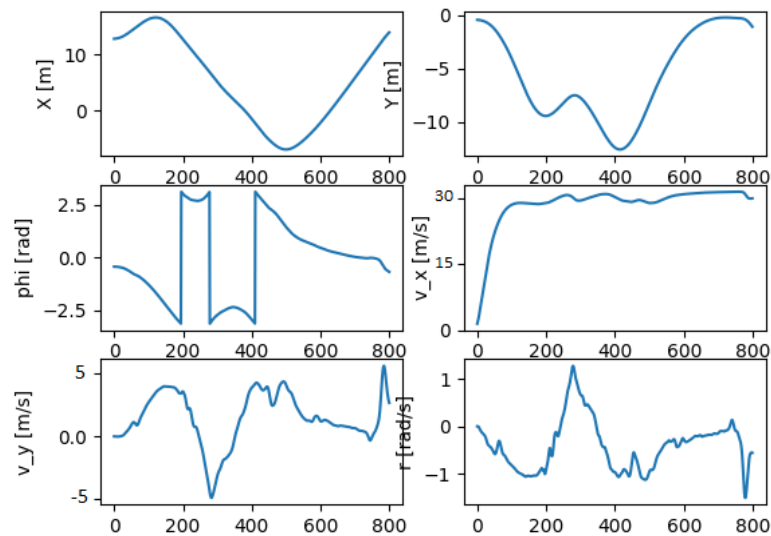


Figure 5.22: State variables during the testing simulation of the MPC module.

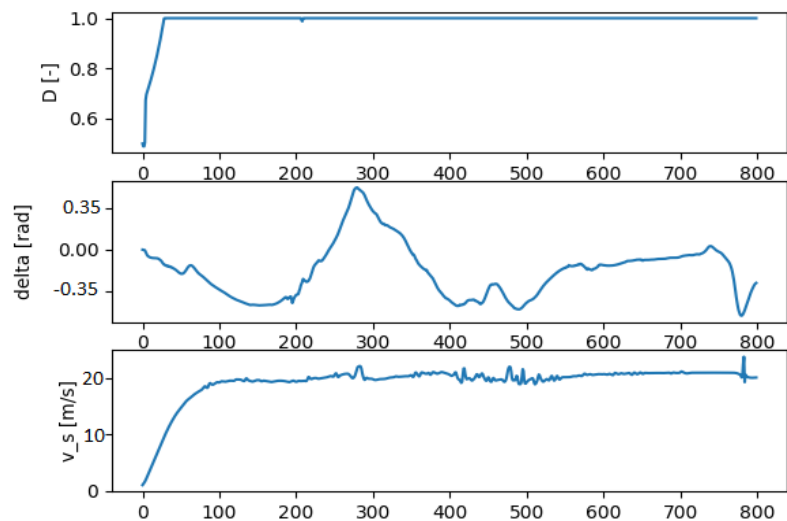


Figure 5.23: Control variables during testing simulation of the MPC module.

5.4. Comparative Results to Previous Work

The basic computer vision architecture is by using a stereo camera pipeline, using the data collected in the rosbags the 3D mapping error based on the monocular camera and the stereo camera pipeline is given with reference to the distance. It's clear how important is the perception redundancy, as the error of the monocular camera in the short range is way better than the stereo camera while the stereo camera exceeds the performance of the monocular camera in the mid and long range, this is according to the setup of the camera as illustrated in the system architecture. The following figure shows the results of the comparative study.

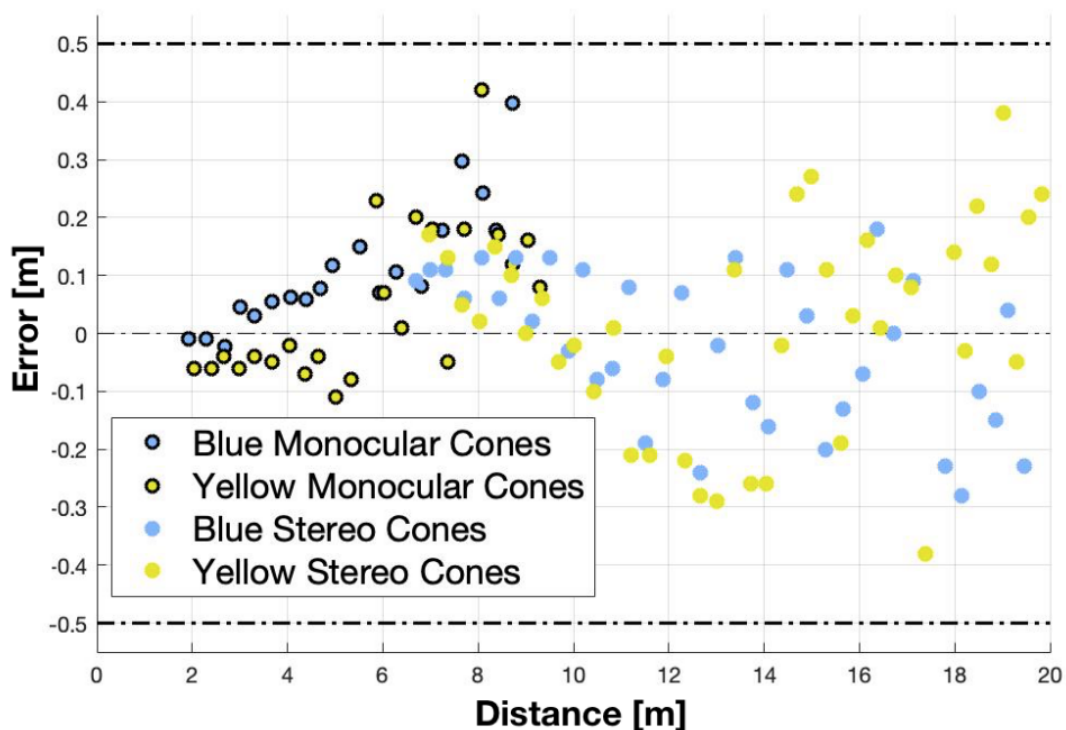


Figure 5.24 Distance errors over various distances of Monocular and Stereo Camera

Distance errors over various distances are shown in [Figure](#). Mean errors for the 20 seconds of recorded data were below 0.5m for both pipelines, and standard deviations were below 5cm for the monocular pipeline and 10cm for the stereo pipeline. Ground truth values for Euclidean distances from the cameras were measured using a Leica Disto D1. The experiment was done statically to remove the dependency on the vehicle's state estimation system. The results reveal a maximum effective disparity offset of 0.15 pixels in the stereovision pipeline achieved through the clustering algorithm, which is 40% lower than the reported value on the datasheet.

Chapter 6: Conclusions and Future Work

Autonomous driving is a fast growing field and a very challenging one, combining computer vision algorithms, robotics, deep learning and creative engineering. Developing software and testing it on hardware for **autonomous racing** is even more challenging, but it massively quickens the creation of reliable and robust autonomous urban vehicles that are ready to be deployed. With this project, we have achieved very convincing results of using a single monocular camera to guide a vehicle into unknown territories within a reliable redundant perception pipeline, a localization & mapping module that effectively maps the track being explored by the vehicle, and finally, a path planner & motion control module that drives the vehicle to its fullest autonomously. This paves the way for testing this software on real hardware, and builds the base for research in the field.

6.1. Faced Challenges

Every member in this project believed in it, and wanted to achieve the greatest of results possible with whatever resources available. There were a lot of challenges and hardships that we faced during our journey with this project, but with enough dedication, out of the box thinking, and support from our supervisor, we were able to get through them and accomplish a goal that we aspired for so long.

Building a workstation: In our project, we train highly complicated deep learning models, develop complex modules and algorithms, and use ROS to integrate and communicate between them, this requires a PC with a decent GPU and CPU. At the start of the project, we didn't have such a PC, during the research and literature review phase, we were able to save enough money and get our hands on a decent PC that will satisfy the project needs.

Collecting cones dataset: For the perception pipeline, the object detector functionality is to detect and classify blue and yellow cones, we weren't able to find any open source cones images dataset. Later on, we found a dataset by Formula Student Driverless teams that required a contribution of at least 600 cones images to be downloadable for other teams, we were able to get our hands on cones, and obtain 600 images and download a 8000 images dataset containing more than 30,000 cones.

Labelling cones dataset: The cones dataset was a very good dataset, containing images of cones in very random and challenging conditions, however, it wasn't labelled by cones color. Labelling such a huge dataset was such a big challenge, but we were able to develop a **manual annotation tool** using Python, and spent three full days, working about 16 hours per each day, labelling the dataset.

Recorded vehicle data: In order to be able to test the developed software for the vehicle, and then deploy such software on real hardware, we needed recorded data of a vehicle moving, such data shall contain various sensor readings and recorded frames through a camera on the vehicle. In our initial plan, we were willing to use the Cairo Uni Racing Team 2019 electric vehicle, but due the COVID-19 issues that was not possible anymore. We were then able to reach out to other formula student teams, and get our hands on some of their recorded data, we fused a lot of sent data together and were able to create our own test cases and scenarios that are going to match with the hardware we will be using in the future.

6.2. Gained Experience

This project was a great journey overall, throughout our work we learnt and gained hand-on experience in lots of fields.

First things first, in the beginning of this project, we began a research phase that lasted for about 3-4 months. Throughout this phase, we finished two Udacity Nanodegrees; Deep Learning Nanodegree, and Self-Driving Car Engineer Nanodegree. In these two nanodegrees, we gained extensive academic and practical knowledge in; **neural networks, CNNs, recurrent neural networks, sensor fusion, robot localization, path planning, and control.**

Additionally, we completely studied more than 15 research papers; in pose estimation, robot localization and mapping, object detection, autonomous racing vehicles, PNP algorithm, depth estimation and many more. All that added to our academic knowledge and allowed us to gain extensive experience in many fields and apply gained knowledge and experience in our project.

Secondly, we gained a broad experience in PyTorch, and implementing deep learning models completely using the framework. PyTorch is a very important and reputable framework in deep learning, we gained experience in debugging model training and inference, in data parallelism, and dynamic computational graphs.

Thirdly, we gained respectable experience in deep learning models training and optimization of training. During the literature survey, we trained multiple object detectors to select the most suitable approach. That enhanced our understanding of terms like precision, recall and mAP. Additionally, learning about selecting the best training hyperparameters, and optimization algorithms.

Last, but not least, we gained great experience and became very familiar in dealing with Robot Operating System (ROS). We developed a skeleton for the entire project, using ROS. Additionally, integrated all the modules together, defining the communication between all these modules.

6.3. Conclusions

Through hard work and dedication, time management, and support from our supervisor, we are able to develop a complete, reliable and robust software of an autonomous racing vehicle.

The software consists of a perception pipeline; that consists of two main embedded pipelines, a monocular camera and a stereo camera pipelines, that work together parrelly. The monocular pipeline effectively detects cones in a track, in a real-time manner, and then classifies them according to their color, such detected cones are then localized within the local map, their depth is estimated through the application of key-points extraction and PNP algorithm. These cones locations estimates are then fused with estimates from the stereo vision pipeline, which provides more accuracy, and reliability through redundant perception. We believe that with more time and research put into this pipeline, we can achieve results that, in terms of accuracy, can be very close to results using a Lidar based sensing. However, the vision range of the Lidar is much better than any vision sensors.

A localization and mapping module then localizes the vehicle in the global map, while updating the map the cones position, additionally, the team developed additional correctional algorithms, making the global mapping more intelligent and error-prune.

The global map is then fed to a path planning and control module, that generates the low level commands for the movement of the vehicle across the track, with the target of achieving maximum performance and driving the vehicle to its absolute limits.

6.4. Future Work

1. Create a Gazebo simulation, that is close as possible to the environment the vehicle will deal with. The simulated vehicle model shall contain all the sensing technologies that are on the vehicle. This simulation shall be used to obtain more test data for software testing.
2. Obtain the actual hardware components; monocular & stereo camera, IMU sensor, wheel odometry sensors, and vehicle embedded computer. Such hardware shall be installed into the vehicle, and be used to record data of the vehicle moving.
3. Test the developed software on Cairo Uni Racing Team 2019 electric vehicle, after installing the hardware and software, and taking enough precautions.
4. Update the Model Predictive Controller to be a controller that utilizes Reinforcement learning. A controller that gains experience within the operation of the vehicle, and applies that experience in improving the vehicle performance along the track.

References

- [1] Formula student germany rules, <https://www.formulastudent.de/fsg/rules/>
- [2] Satyarth Praveen, "Efficient Depth Estimation Using Sparse Stereo-Vision with Other Perception Techniques", Published: May 24th 2019
- [3] ROS: Robot Operating System. <https://ros.org/>
- [4] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. ArXiv, abs/1804.02767, 2018.
- [5] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.
- [6] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. CoRR,abs/1406.2283, 2014.
- [7] Ankit Dhall. Real-time 3d pose estimation with a monocular camera using deep learning and object priors on an autonomous racecar. arXiv preprint arXiv:1809.10548, 2018.
- [8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.
- [9] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. CoRR, abs/1505.04597, 2015.
- [10] Cross ratio. <http://robotics.stanford.edu/~birch/projective/node10.html>. Accessed: 2018-09-11.
- [11] Aiden Nibali, Zhen He, Stuart Morgan, and Luke Prendergast. Numerical coordinate regression with convolutional neural networks. ArXiv,abs/1801.07372, 2018.
- [12] OpenCV calibration and 3d reconstruction. https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html#solvepnp. Accessed: 2018-09-11.
- [13] Chris Harris and Mike Stephens. A combined corner and edge detector. Citeseer, 1988.
- [14] David G Lowe. Distinctive image features from scale-invariant keypoints. International journal of computer vision, 60(2):91–110, 2004.
- [15] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In European conference on computer vision, pages 404–417. Springer, 2006.
- [16] Stefan Leutenegger, Margarita Chli, and Roland Y Siegwart. Brisk: Binary robust invariant scalable keypoints. In Computer Vision (ICCV), 2011 IEEE International Conference on, pages 2548–2555. IEEE, 2011.
- [17] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In European conference on computer vision, pages 778–792. Springer, 2010.
- [18] All graphs and diagrams made with <https://app.diagrams.net/>
- [19] Thrun, S., Burgard, W., Fox, D. (2005). *Probabilistic robotics*. Cambridge, Mass.: MIT Press. ISBN: 0262201623 9780262201629

Appendix A: Development Platforms and Tools

This section explains used tools, platforms, and hardware kits. Any ready-made module used to build the AutoRace system. At first we will look for the hardware used to develop this autonomous kit and then the software tools which are used to construct and develop the full pipeline from the perception, localization, mapping, to the motion control system.

A.1. Hardware Platforms

The following figure describes the Autonomous kit sensors and actuators which are needed to convert Cairo University racing team's electric vehicles into autonomous vehicles and to match the system architecture which is mentioned in chapter 4.

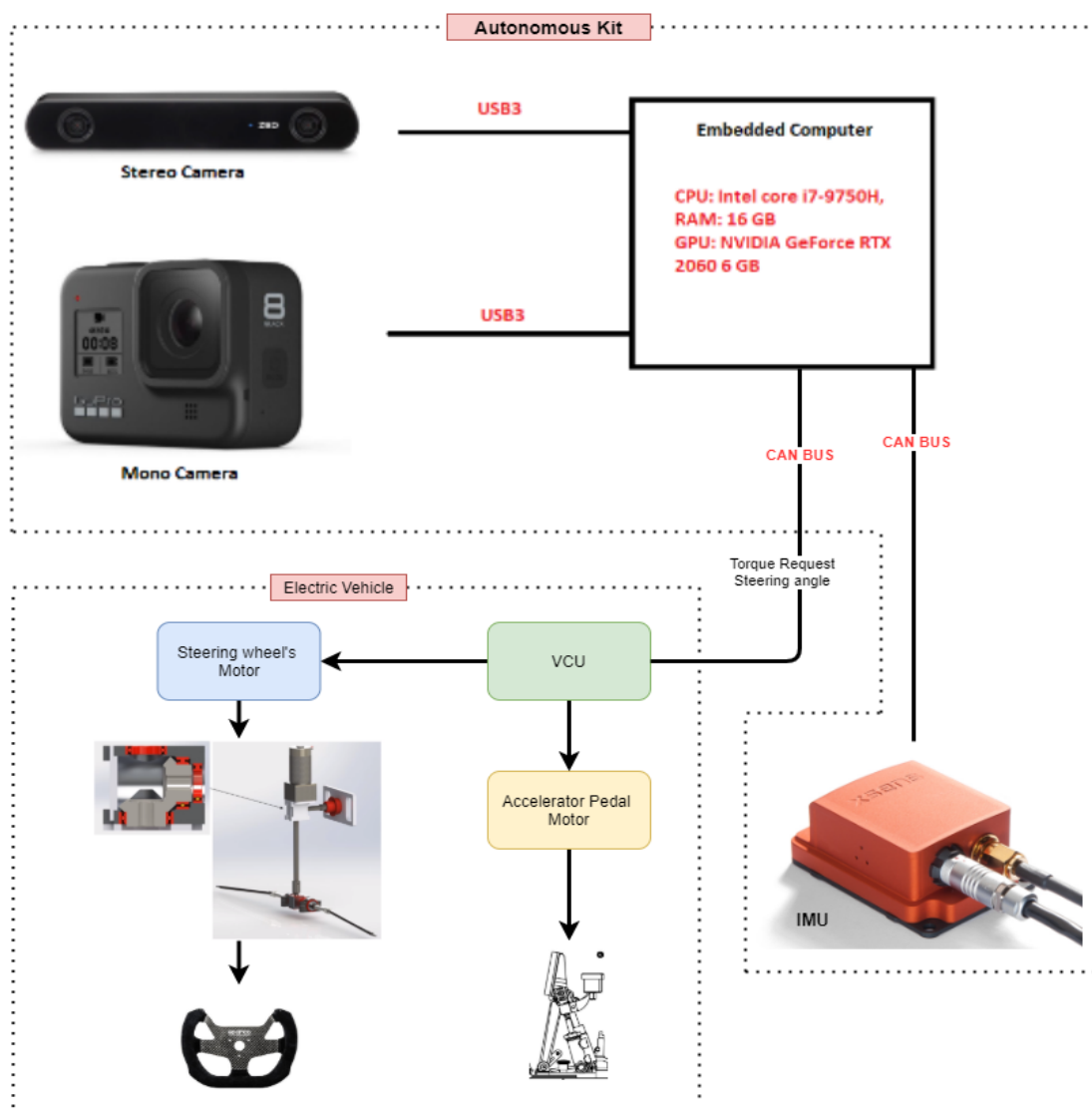


Figure A.1 Overview of the hardware sensors and actuators of the autonomous kit

A.2. Software Tools

AutoRace was developed with python and C++, where python is used in the high level tasks, and the C++ was used in the motion control when executed commands by sending them to the VCU. Some packages are used in order to build the full software stack.

A.2.1 NumPy

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

A.2.2 OpenCV

OpenCV is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez (which was later acquired by Intel). The library is cross-platform and free for use under the open-source BSD license.

A.2.3 TensorFlow

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It is used for both research and production at Google.

A.2.4 PyTorch

PyTorch is an open source machine learning library based on the Torch library, used for applications such as computer vision and natural language processing, primarily developed by Facebook's AI Research lab (FAIR). It is free and open-source software released under the Modified BSD license.

A.2.5 ROS

Robot Operating System (ROS or ros) is robotics middleware (i.e. collection of software frameworks for robot software development). Although ROS is not an operating system, it provides services designed for a heterogeneous computer cluster such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. Running sets of ROS-based processes are represented in a graph architecture where processing takes place in nodes that may receive, post and multiplex sensor data, control, state, planning, actuator, and other messages.

Despite the importance of reactivity and low latency in robot control, ROS itself is *not* a real-time OS (RTOS). It is possible, however, to integrate ROS with real-time code. The lack of support for real-time systems has been addressed in the creation of ROS 2.0, a major revision of the ROS API which will take advantage of modern libraries and technologies for core ROS functionality and add support for real-time code and embedded hardware.

A.2.6 Basic Detection Network YOLO

YOLO, is a new approach to object detection. Prior work on object detection repurposes classifiers to perform detection. Instead, it frames object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it is optimized end-to-end directly on detection performance. We have used the basic idea of the network to customize for our 2D object detection module.

Appendix B: Use Cases

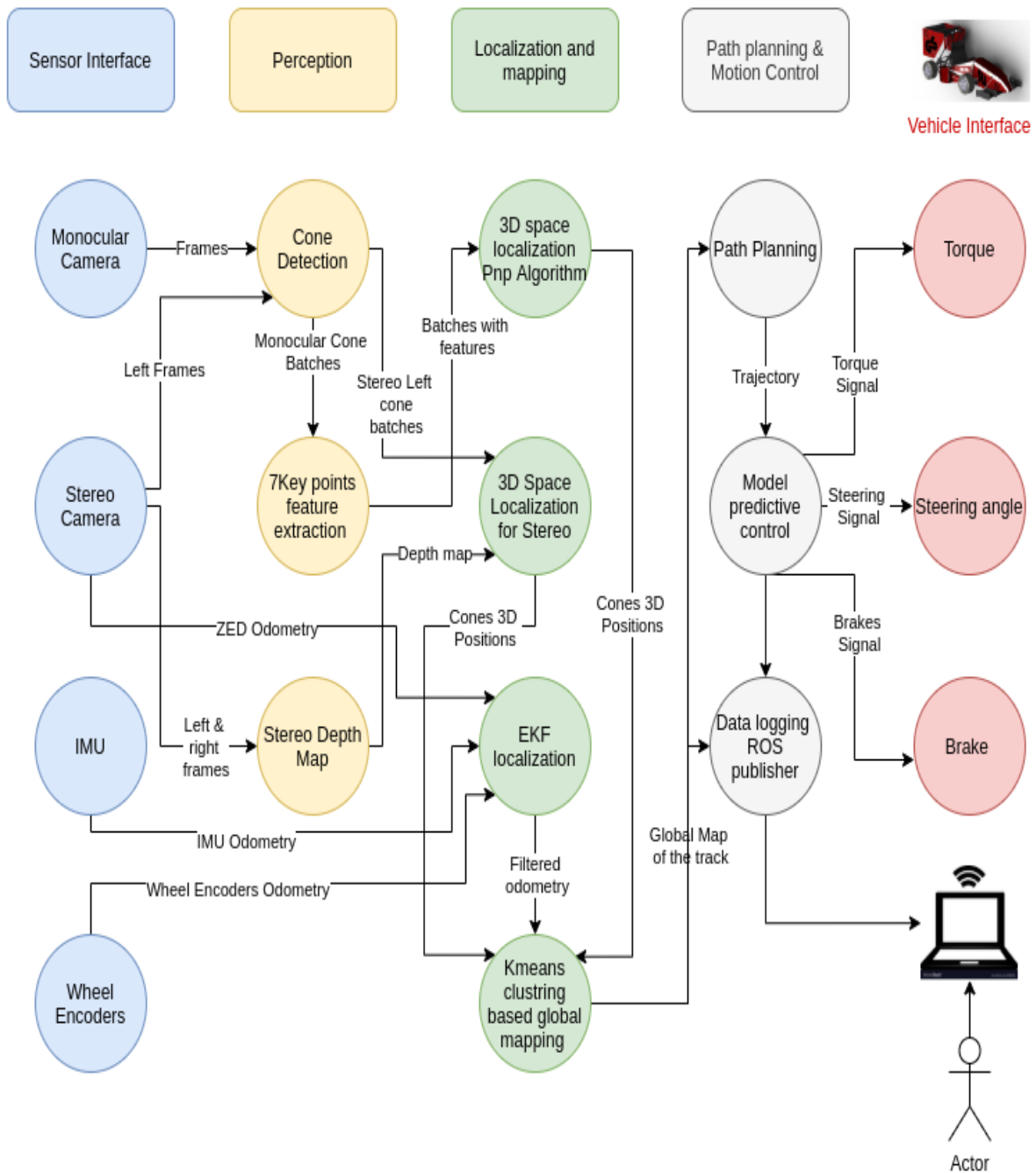


Figure A.2 Use case of the project

Appendix C: User Guide

1) Hardware Platform

NVIDIA RTX 1060 Super or higher (with CUDA capabilities)
AMD Ryzen 5 3600xt or higher
DDR4 RAM - 16GB or higher

2) Software Platform

Install Linux-Ubuntu 18.04 LTS : <https://releases.ubuntu.com/18.04/>

Install ROS Melodic : <http://wiki.ros.org/melodic/Installation>

Install CONDA Environment With Python 3.7 :

<https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html>

3) Software Packages Requirements

- CUDA>=10.1
- python==3.6
- numpy==1.16.4
- matplotlib==3.1.0
- torchvision==0.3.0
- opencv_python==4.1.0.25
- torch==1.1.0
- requests==2.20.0
- pandas==0.24.2
- imgaug==0.3.0
- onnx==1.6.0
- optuna==0.19.0
- Pillow==6.2.1
- protobuf==3.11.0
- pymysql==0.9.3
- retrying==1.3.3
- tensorboardX==1.9
- tqdm==4.39.0

4) Training and Inference The detection network

- Get the cones dataset and extract it.
- Get the labels of the dataset as a CSV file.
- Training command:
`python3 train.py --model_cfg=model_cfg/yolo_baseline.cfg
--weights_path=dataset/sample-yolov3.weights`
- Once you've finished training, you can access the weights file in `./outputs/`
- Inference command:
`python3 detect.py --model=<path to .pt weights file> --img=<path to an image>`
- Once you've finished inference, you can access the result in `./outputs/visualization/`

5) Training and Inference The feature extraction network

- Get the cones dataset and extract it.
- Get the labels of the dataset as a CSV file.
- Training command:
`python3 train_eval.py --study_name=<name for this experiment>`
- Once you've finished training, you can access the weights file in `./outputs/`
- Inference command:
`python3 detect.py --model=<path to .pt weights file> --img=<path to an image>`
- Once you've finished inference, you can access the result in `./outputs/visualization/`

6) Launching the whole program on ROS

- To simulate and visualize the whole project get the rosbag and go to in the terminal to its directory and play it by this command:
`-rosbag play "Name of the rosbag"`
- Open rViz to visualize and simulate the project by this command:
`-ros run rViz rViz`

Appendix E: Feasibility Study

E.1 Economic Feasibility

In this section, the initial costs needed for the development and completion of the project are stated. Additionally, the cost estimation for future improvements of the project, and testing the developed software on real hardware is also added.

E.1.1 Initial Costs

Table E.1 Initial costs of the project

| Item | Cost | Description | Fulfilled |
|------------------------------|------------|--|-----------|
| NVIDIA RTX 2060 Super | 8300 EGP | Graphical processing unit of workstation to develop the project on. It will be mainly used to train deep learning models and test the project modules. | YES |
| AMD Ryzen 5 3600xt | 5000 EGP | CPU of workstation. It was found that this processor will give us enough computing power with its multithreading properties and fairly fast clock speed. | YES |
| DDR4 RAM - 16GB | 1800 EGP | Memory of the workstation. | YES |
| 1TB HDD + 250GB SSD | 2200 EGP | HDD drive is for larger local files, SSD is mainly for workstations operating system to allow for faster operation. | |
| Other workstation components | 2700 EGP | Casing - power supply - motherboard - Ethernet module. | YES |
| Total: | 20,000 EGP | | |

E.1.2 Cost Estimation for Future Improvements

Table E.2 Initial costs of the project

| Item | Cost | Description | Fulfilled |
|---------------------------------------|-------------|---|------------------|
| Embedded Computer - ASUS g731gv | 31,000 EGP | This will act as the brain of the vehicle. The embedded computer will run and process the entire code, and run the operating systems (ROS). It must be compatible with TensorRT technology to allow for real time capabilities. | NO |
| Monocular Camera - GoPro HERO 8 Black | 8200 EGP | The monocular camera that will input frames into the perception pipeline. It must be an action camera to ensure high quality frames at a high FPS. | NO |
| Stereo Camera - ZED 2 | 10800 EGP | The stereo camera for the stereo vision pipeline. | NO |
| Total: | 50,000 EGP | | |

E.2 Technical Feasibility

In this section, the risk assessment and solutions we anticipated will be stated.

Tables E.3, E.4, and E.5 show the criterias we used to evaluate a potential risk.

| Likelihood | |
|------------|----------------|
| 1 = | Very unlikely |
| 2 = | Unlikely |
| 3 = | Likely |
| 4 = | Very Likely |
| 5 = | Almost certain |

| Potential Severity | |
|--------------------|--|
| 1 = | Minimal Delay of 1-2 day(s) |
| 2 = | Medium-range delay of 3-6 days. |
| 3 = | High delay of 1 week. |
| 4 = | More than 1 week delay. |
| 5 = | A delay that will directly delay a goal from being achieved. |

| Risk Rating | |
|-------------|---------------------------|
| 1-4 | Low |
| 5-8 | Medium |
| 9-25 | High/ intolerable risk |

Table E.6 shows our risk assessment and solutions.

| Risk | Existing controls | Likelihood (L) | Severity (S) | Risk Rating (L X S) | Additional controls |
|---|---|----------------|--------------|---------------------|--|
| Economical inability to buy a workstation. | Use online resources like Google Colab or AWS. | 2 | 5 | 10 | Additional controls can be using CUFEE existing servers or GPUs with coordination with our supervisor. Additionally, there are a lot of internet cafes with decent workstations, we can get a deal with one of them and use with their machines regularly. |
| Research phase fails to prepare members for the next phases of the project. | The team members already have fairly good knowledge in the field, the courses that were selected have high ratings and excellent reviews, and the papers used have excellent results, were published in reputable articles. | 1 | 5 | 5 | The timeline schedule gives the team the opportunity to extend the research phase, if needed, and still be able to finish the design within the deadline. |
| Selecting false approaches, or big errors in the designs. | Research phase should have given the members the enough knowledge to be able to select the proper approaches and designs. | 1 | 3 | 3 | The timeline schedule shall give the team enough time to redesign or reselect different approaches and still be able to finish the project within the deadline. |
| Team members failing to implement selected approaches. | The research phase should have given the teams the enough knowledge. Additionally, spending enough time in the modeling phase should give the team members the ability to select the right approaches, that are feasible. | 2 | 5 | 10 | The timeline schedule gives the team members a fairly big period of time, that shall be more than enough to finish all the project modules, test them and finally integrate them. |

E.3 Schedule Feasibility

In order to finish this project efficiently and manage to stay within the time limits, we managed to plan a **timeline** for the entire project. Figure E.1 shows a Gantt chart that shows how we divided the project into four main phases.

The research phase (09/01/19 - 01/01/20); is the learning period, throughout this phase the team members should study two main courses and start literature review, gathering enough knowledge and background information that shall give the team enough push to start designing and implementing the project.

The modeling phase (01/10/20 - 02/10/20); this is mainly the design phase, in this phase an initial design shall be made for all the modules of the project, modeling these designs in the form of block diagrams, selecting software and simulation tools and preparing an integration plan.

The prototyping phase (02/10/20 - 06/10/20); this is the phase where the designs shall be implemented, programming all the main modules, doing system testing, integrating all the modules and then testing the entire system.

The documentation phase (04/10/20 - 07/10/20); in this phase, the entire project shall be documented, focusing on documenting the approaches & the implementations used, the testing procedures, and the outputs of the project.

Table E.7 shows the Gantt chart of the prototyping (implementation phase).

| WBS | Task | Start | End | Progress |
|----------|---|-------------|-------------|----------|
| 1 | Research Phase | Sun 9/01/19 | Wed 1/01/20 | |
| 1.1 | Machine Learning Nanodegree | | | 100% |
| 1.2 | Self Driving Car Nanodegree | | | 100% |
| 1.3 | Research Paper (Literature Review) | | | 100% |
| 1.4 | Hardware Component Selection | | | 100% |
| 1.5 | YOLOv3 High fps Cone Detection and Classifier using OpenCV and CUDA | | | 100% |
| 2 | Modeling Phase | Wed 1/01/20 | Mon 2/10/20 | 100% |
| 2.1 | Modeling Perception Pipeline Software Architecture | | | 100% |
| 2.2 | Modeling Localization & Mapping Architecture | | | 100% |
| 2.3 | Modeling Trajectory planning Architecture | | | 100% |
| 2.4 | Design of a Complete Circuit Schematic | | | 100% |
| 3 | Prototyping Phase | Mon 2/10/20 | Wed 6/10/20 | 100% |
| 3.1 | Implementing Designs | | | 100% |
| 3.2 | Unit Testing | | | 100% |
| 3.3 | Integrating All Systems | | | 100% |
| 3.4 | System Testing | | | 100% |
| 4 | Documentation phase | Fri 4/10/20 | Wed 7/22/20 | 100% |
| 4.1 | GP Document and Presentation | | | 100% |
| 4.2 | Research Reporting | | | 100% |

Due to the complexity and difficulty of the prototyping phase, we created a dedicated timeline for that phase. Table E.7 and E.8 shows the Gantt chart of the prototyping (implementation phase).

Table E.8 shows the Gantt chart of the prototyping (implementation phase).

| WBS | Task | Start | End | Progress % |
|----------|--|-------------|-------------|-------------|
| 1 | Monocular Camera Perception Module | Mon 2/10/20 | Fri 4/10/20 | 100% |
| 1.1 | Training object detection module with 50,000 cone images dataset. | | | 100% |
| 1.2 | Implement key points extraction deep neural network (ResNet). | | | 100% |
| 1.3 | Training ResNet module with 200,000 cone batches dataset. | | | 100% |
| 1.4 | Integrate object detection module with the ResNet. | | | 100% |
| 1.5 | Test with a video input and output cone batches with 7 key points. | | | 100% |
| 1.6 | Discover different motion control algorithms to drive the vehicle through the planned path | | | 100% |
| 2 | Monocular Camera Localization Module | Fri 4/10/20 | Fri 5/01/20 | 100% |
| 2.1 | Implement PNP algorithm to localize 3D position of cones. | | | 100% |
| 2.2 | Integrate the PNP algorithm with the two modules. | | | 100% |
| 2.3 | Test PNP algorithm on single frames (set of batches). | | | 100% |
| 2.4 | Integrate the three modules to work on video (set of frames). | | | 100% |
| 2.5 | Implement a Simulink MPC model and run simulations | | | 100% |
| 3 | Mapping and visualization | Fri 5/01/20 | Wed 6/10/20 | 100% |
| 3.1 | Implement the self localization algorithm from estimated cone positions. | | | 100% |
| 3.2 | Construct a 3D map of complete track from input video. | | | 100% |
| 3.3 | Constructed a skeleton to integrate the vehicle modules using a Robotics Operating Systems (ROS) | | | 100% |
| 3.4 | Implemented a 3D visualization map using the robotics operating system (rViz) | | | 100% |
| 3.5 | Imported the vehicle and track cone model into the visualization for simulation and testing | | | 100% |
| 3.6 | Implement a MPC C++ node to be deployed in the vehicle skeleton | | | 100% |

WEE Water Engineering and Environment

STE Structural Engineering

PPC Petro Chemical Engineering

MDE Mechanical Design Engineering

CEM Construction Engineering and Management

CCE Communication and Computer Engineering

AET Architectural Engineering and Technology

Sponsor

**FORMULA
STUDENT**

Institution of
**MECHANICAL
ENGINEERS**

