FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Autonomous Vehicle Racing - Perception

Gonçalo Faria Ferreira Pimentel da Mota

Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Fernando Arménio da Costa Castro e Fontes

October 31, 2024

Resumo

Este projeto explora o desenvolvimento de um sistema para corridas autónomas ('autonomous racing'), onde o objetivo é que veículos em escala reduzida naveguem pela pista de forma totalmente autónoma, sem intervenção humana em tempo real. Embora os veículos utilizados sejam semelhantes a carros controlados remotamente (RC) em termos de escala, o controlo é realizado de forma autónoma através de algoritmos avançados de perceção, estimação de estado, e controlo de trajetória. O projeto visa replicar os desafios de condução autónoma em ambiente de competição, utilizando um sistema que permite ao veículo seguir uma trajetória ideal de forma precisa e eficiente. Os principais componentes utilizados são carros na escala 1/43 e uma câmara.

No contexto da condução autónoma, a capacidade de seguir uma trajetória precisa é de importância fundamental para garantir uma navegação segura e eficiente. É fundamental alcançar uma elevada precisão na pose e no movimento do veículo. Consequentemente, é incorporada uma câmara na cadeia de localização, aumentando assim a robustez e a fiabilidade do sistema. Os dados visuais obtidos a partir da câmara fornecem ao veículo autónomo uma compreensão abrangente do seu ambiente circundante, permitindo a execução precisa e meticulosa do caminho pretendido.

Para complementar os dados visuais, é utilizado um filtro de Kalman estendido (EKF) para melhorar a precisão da estimativa da posição e orientação do veículo em tempo real. Ao corrigir continuamente a trajetória estimada do veículo com base nas medições dos sensores, o EKF minimiza os erros de seguimento e garante que o automóvel segue a trajetória pretendida com maior precisão.

Além disso, este projeto está alinhado com vários Objectivos de Desenvolvimento Sustentável (ODS) das Nações Unidas. A incorporação de tecnologias respeitadoras do ambiente serve para fazer avançar os objectivos de desenvolvimento urbano sustentável e de infra-estruturas resilientes. Para além disso, a implantação destas tecnologias sofisticadas neste projeto tem o potencial de melhorar a segurança rodoviária e a mobilidade urbana, contribuindo assim para a criação de cidades inclusivas, seguras e sustentáveis. O projeto também apoia a educação STEM e a aprendizagem ao longo da vida, proporcionando uma aplicação prática para estudantes e investigadores, promovendo a inovação e a adaptação contínua no domínio da condução autónoma.

Em suma, esta dissertação apresenta uma abordagem abrangente para melhorar o desempenho e as capacidades dos veículos de corrida autónomos através de técnicas avançadas de perceção, controlo e aprendizagem, com implicações mais vastas para o desenvolvimento urbano, a segurança e a educação.

Palavras-chave: Carros RC, câmara, pista de corridas, filtro de Kalman estendido (EKF), modelo cinemático da bicicleta, otimização da trajetória, localização visual, calibração da câmara.

Abstract

This project examines the development of an autonomous racing system, whereby the objective is for small-scale vehicles to navigate the track without human intervention in real-time. The system is designed to enable autonomous navigation of the track by the cars, which are of a small scale. Although the vehicles employed are analogous to remote-controlled (RC) cars in scale, control is conducted autonomously through sophisticated algorithms for perception, state estimation and trajectory control. The project aims to reproduce the difficulties inherent to autonomous driving in a racing context, utilising a system that enables the vehicle to adhere to an optimal trajectory with precision and efficiency. The principal components employed are 1/43 scale cars and a camera.

In the context of autonomous driving, tracking a precise trajectory is vital for ensuring safe and efficient navigation. It is paramount to achieve high accuracy in the vehicle's pose and motion. Consequently, a camera is incorporated into the localisation pipeline, thus enhancing the system's robustness and reliability. The visual data obtained from the camera provides the autonomous vehicle with a comprehensive understanding of its surrounding environment, enabling the precise and meticulous execution of its intended path.

To supplement the visual data, an Extended Kalman Filter (EKF) is employed to improve the estimation accuracy of the vehicle's position and orientation in real-time. By continuously correcting the vehicle's estimated trajectory based on sensor measurements, the EKF minimises tracking errors and ensures that the car follows the intended path with higher precision.

Furthermore, this project is aligned with several United Nations Sustainable Development Goals (SDGs). The incorporation of environmentally friendly technologies serves to advance the objectives of sustainable urban development and resilient infrastructure. Moreover, the deployment of these sophisticated technologies in this project has the potential to enhance road safety and urban mobility, thereby contributing to the creation of inclusive, safe, and sustainable cities. The project also supports STEM education and lifelong learning by providing a practical application for students and researchers, fostering innovation and continuous adaptation in the field of autonomous driving.

In summary, this dissertation presents a comprehensive approach to enhancing the performance and capabilities of autonomous racing vehicles through advanced perception, control, and learning techniques, with broader implications for urban development, safety, and education.

Keywords: RC cars, camera, racetrack, Extended Kalman Filter (EKF), Bicycle kinematic model, trajectory optimisation, visual localisation, camera calibration.

Sustainable Development Goals (SDGs)

The intersection of autonomous racing and the United Nations Sustainable Development Goals (SDGs) 12, 11, 9, and 4 presents several intriguing possibilities for urban development, infrastructure, education, and technological innovation.

Table 1: Contributions to SDGs and Performance Indicators.

SDG	Target	Contribution	Performance Indicators and Metrics
12	12.1	Environmental Sustainability	Incorporation of environmentally friendly technologies
11	11.2	Sustainable Urban Development Technological Innovation for Safety	Create inclusive, safe, and sustainable cities; enhance overall urban mobility Innovations can be applied to general road safety
9	9.1 9.2	Resilient Infrastructure Sustainable Industrialisation	Require the construction of advanced infrastructure Promote inclusive and sustainable industrialisation
4	4.3 4.4	Inclusive and Equitable Education Promotion of Lifelong Learning	Provide all people with learning opportunities The need for continuous learning and adaptation

SDG 12

'*Ensure sustainable consumption and production patterns*' [26]

Environmental Sustainability

- The development and deployment of autonomous racing vehicles frequently incorporate environmentally friendly technologies, such as electric propulsion, which contribute to sustainable industrialisation, urban development, and overall environmental responsibility.

SDG 11

'*Make cities and human settlements inclusive, safe, resilient and sustainable*' [26]

Sustainable Urban Development

- The application of autonomous racing technology has the potential to influence urban planning and transportation systems, thereby contributing to the creation of inclusive, safe, and sustainable cities.
- The implementation of efficient autonomous transportation has the potential to reduce traffic congestion, lower emissions, and enhance overall urban mobility.

Technological Innovation for Safety

- The advancement of autonomous racing technology has the potential to drive innovation in many areas, including vehicle autonomy and safety features. These innovations can be applied to general road safety, thereby contributing to the creation of safer urban environments.

SDG 9

'*Build resilient infrastructure, promote inclusive and sustainable industrialisation and foster innovation*' [26]

Resilient Infrastructure and Sustainable Industrialisation

- The development of autonomous racing requires the construction of advanced infrastructure, including intelligent roadways, communication networks, and energy-efficient facilities.
- The investment in such infrastructure can positively affect the broader transportation systems and promote sustainable industrialisation.

SDG 4

'*Ensure inclusive and equitable quality education and promote lifelong learning opportunities for all*' [26]

Inclusive and Equitable Education

- The development and integration of autonomous racing technologies present an opportunity for students' education and skill development in science, technology, engineering, and mathematics (STEM) fields. These technologies provide students with learning opportunities that can extend beyond the classroom and promote lifelong learning.

Promotion of Lifelong Learning

- The field of autonomous racing is undergoing rapid evolution, necessitating continuous learning and adaptation. This aligns with the goal of promoting lifelong learning opportunities for all.

The advancements in technology, safety, and sustainability associated with autonomous racing have the potential to positively impact various aspects of urban living and education, creating synergies with the UN's Sustainable Development Goals.

Agradecimentos

Antes de mais, gostaria de expressar a minha profunda gratidão às pessoas que contribuíram de maneira significativa para o meu percurso pessoal e académico, nomeadamente aos meus pais, irmão, família e a todos os meus professores,

em particular destacaria

os meus pais, pelo amor e pela confiança que sempre depositaram em mim, e o meu irmão, pela paciência e compreensão nos momentos mais desafiadores. Sem o vosso apoio, esta conquista não teria sido possível;

toda a restante família, avós, tios e primos, pelo apoio incondicional e encorajamento constante ao longo de todo este meu percurso;

o meu professor e orientador Fernando Fontes ao qual estou profundamente grato e cuja a orientação e o conhecimento foram fundamentais para o desenvolvimento deste trabalho. Agradeço pela paciência, pelas críticas construtivas e pelos conselhos valiosos que o enriqueceram. A sua dedicação e entusiasmo pela pesquisa foram uma fonte de inspiração contínua para mim;

a equipa do Laboratório I202 - Otimização e Estimativa do Controlo de Sistemas, o meu sincero agradecimento pelo suporte técnico e pela colaboração imprescindível. Agradeço a todos pela assistência constante, pela partilha de conhecimentos e pelas discussões produtivas que ampliaram os meus horizontes. O trabalho em equipa e o ambiente acolhedor foram essenciais para a realização das atividades práticas deste projeto;

a unidade de investigação SYSTEC-ARISE Associated Laboratory, o meu profundo agradecimento por me ter financiado, com fundos de projetos FCT, os componentes necessários para a realização da Tese de mestrado;

todos os amigos pelo suporte, incentivo e amizade que sempre me disponibilizaram, entre os quais saliento o João Rodrigues e o João Sampaio.

A todos, meu muito obrigado.

Gonçalo Mota

"All things are difficult before they are easy."

Dr. Thomas Fuller

Contents

List of Figures

List of Tables

Abbreviations and Symbols

EKF	Extended Kalman filter
UKF	Unscented Kalman Filter
PCB	Printed Circuit Board
IMU	Inertial measurement unit
CNN	Convolutional neural networks
RC	Remote-controlled
ROI	Region of interest
FOV	Field-of-View
PID	Proportional-Integral-Derivative
RL	Reinforcement Learning
PPO	Proximal Policy Optimisation
CTE	Cross-Track error
LED	Light Emitting Diode
ETH Zurich	Eidgenössische Technische Hochschule Zürich (Swiss Federal Institute of Technology in Zurich)
px	pixel
cv2	OpenCV (Open Source Computer Vision Library)
f_x	Focal Length in the x-direction (pixels)
f_y	Focal Length in the y-direction (pixels)
K	Intrinsic Parameters of the Camera
P	Camera matrix
R	Rotation matrix
t	Translation matrix
r	Radius in Distortion Equations
k_1, k_2, k_3	Radial Distortion Coefficients
p_1, p_2	Tangential Distortion Coefficients
$\mathbf{X}$	State Vector
$f(X, u)$	State Transition Function
u	Control Inputs
F_x, F_w	Jacobian matrices
$\mathbf{P}$	Covariance matrix
$\mathbf{z}$	Actual Sensor Measurement vector
$\mathbf{Q}$	Process Noise Covariance
$\mathbf{R}$	Measurement Noise Covariance
$\mathbf{h}(\mathbf{x})$	Predicted Measurement
$\mathbf{H_x}$	Jacobian of the Measurement function
$\mathbf{y}$	Innovation (measurement residual)
$\mathbf{S}$	Innovation Covariance

$\mathbf{K}$	Kalman Gain
$\mathbf{I}$	Identity matrix
b	Wheelbase of the vehicle
δ	Steering angle
θ	Heading angle
α	Slip angle
a	Distance from the centre of mass to the rear axle
v	Vehicle's velocity

Chapter 1

Introduction

1.1 Context

Motorsports like Formula 1, NASCAR, and WRC are centred around intense competition. Still, the technological advancements that result from the pursuit of performance and safety have a much wider impact on the broader automotive industry. For instance, safety improvements like crumple zones, advanced braking systems, tyre technology, and driver assistance systems are all examples of advancements that originated in the high-stakes world of motorsports and have since been adopted by mainstream automobile manufacturers.

One of the primary goals of these racing series is to achieve the fastest lap times possible, and this requires a significant investment in new technologies to push the car to its absolute limits. Having the best car technology is certainly important, but a driver's raw skill, precise control, quick reflexes, and ability to push the car to the edge while maintaining control are equally crucial factors in determining success on the racetrack.

The emerging field of autonomous vehicle racing operates on a similar premise, with one key difference - rather than a human driver, the vehicle is controlled entirely by advanced software and sensor systems. This software must be able to detect the track, identify and avoid obstacles, determine the vehicle's precise location relative to other cars, and make the necessary real-time corrections to keep the autonomous racer on the optimal racing line.

The complex challenge of estimating vehicle location has been the focus of extensive research, with the Kalman Filter method and its variants being one of the leading approaches. "The Kalman filter is a powerful mathematical tool used for sensor fusion and state estimation in a variety of applications, including autonomous vehicles" [28]. Since the autonomous racing environment is inherently non-linear, an Extended Kalman Filter (EKF) is typically implemented instead of a standard Kalman Filter. However, before this sophisticated estimation and control system can be deployed, the camera system used to visualise the race circuit and detect the vehicle's position must first be carefully calibrated. This calibration process is absolutely essential to ensure accurate perception and decision-making by the autonomous racing platform.

The overall autonomous racing project comprises three interconnected dissertation projects: Perception, Control, and Digital Twin. These three components work together to create a fully autonomous racing system capable of navigating a race circuit.

The Perception component is concerned with the development of advanced computer vision and sensor fusion techniques with the objective of accurately detecting and localising the vehicle within the race environment. This encompasses the calibration of camera systems to enable the precise mapping of the track layout, the identification of the vehicle's position relative to the racing line, and the detection of obstacles or other vehicles on the circuit. Sophisticated algorithms, such as the Extended Kalman Filter, are employed to fuse data from multiple sensors, including cameras, inertial measurement units (IMUs), light detection and ranging (LIDAR), and global positioning systems (GPS), in order to maintain a robust and reliable perception of the vehicle's surroundings.

The Control component translates the perception data into the necessary steering, throttle, and braking inputs to keep the autonomous vehicle on the optimal racing line. This requires the implementation of complex control algorithms that can rapidly process sensor data, predict the vehicle's future state, and make split-second adjustments to the control inputs. The objective is to push the autonomous car to its absolute limits, just as a human driver would while maintaining stability and control.

Finally, the Digital Twin component generates a comprehensive digital representation of the autonomous racing system, including the vehicle dynamics, sensor suite, and control algorithms. This digital replica enables extensive simulation, testing, and optimisation of the system before deployment on the physical racetrack. The Digital Twin facilitates exploring a wide range of scenarios, refining algorithms, and validating system performance in a safe and controlled environment, thereby reducing the risk of physical hardware failure. Additionally, real-time support from the Digital Twin allows for continuous monitoring and updating of the autonomous racing system during actual races. This real-time capability provides instantaneous feedback and predictive insights, enabling dynamic adjustments and enhancing overall performance and reliability by preempting potential issues before they manifest on the physical vehicle.

1.2 Motivation

The Autonomous Racing project, in which this dissertation is integrated, is being developed in LSCOE - Laboratory for Systems Control, Optimization and Estimation of SYSTEC-ARISE located in DEEC - FEUP. This is the project's inaugural year, which will form the foundation for subsequent thesis proposals related to this topic.

The domain of autonomous racing represents a revolutionary frontier in the field of motorsports and technology. As the development of autonomous driving technology accelerates, racing offers a unique platform for exploring the limits of what is achievable in this domain. The motivation behind autonomous racing can be distilled into several core factors, including the advancement of technology, the provision of a testing ground for real-world applications, the promotion

of STEM education and innovation, the resolution of urban mobility challenges, and the fostering of international collaboration.

1.2.1 Advancing Technology

The advancement of the technologies that facilitate autonomous vehicles is a fundamental objective of the autonomous racing field. Creating high-pressure environments in which vehicles operate at speed and under dynamic conditions allows engineers and developers to significantly enhance the responsiveness, decision-making capabilities and safety of autonomous systems. This pursuit not only encourages technological innovation but also generates valuable data that can be employed to enhance autonomous vehicles in civilian contexts.

1.2.2 Testing Ground for Real-World Applications

Autonomous racing provides a unique opportunity to test and refine advanced technologies in a real-world setting. Racing circuits emulate the challenging scenarios that autonomous vehicles may encounter on public roads, including obstructions, narrow turns, and variable weather conditions. By subjecting these vehicles to rigorous testing in a controlled setting, developers can identify potential issues and optimise performance before deployment in real-world scenarios. This emphasis on safety and reliability may facilitate the integration of autonomous vehicles into everyday life.

1.2.3 Promoting STEM Education and Innovation

As previously stated in the Sustainable Development Goals (SDGs), the field of autonomous racing is an optimal vehicle for promoting STEM (Science, Technology, Engineering, and Mathematics) education due to its integration of technology, engineering, and innovation. Initiatives such as the Abu Dhabi Autonomous Racing League (A2RL)[2.1] provide a platform for young engineers and students to engage with cutting-edge technology and participate in competitions that challenge their skills. This alignment with education cultivates a new generation of innovators who are equipped with the required skills for tomorrow's technology landscapes.

1.2.4 Addressing Urban Mobility Challenges

The developments in autonomous racing have the potential to significantly enhance urban mobility. By further developing autonomous technologies, it is possible to identify solutions that address a number of significant issues, including traffic congestion, road safety and emissions. This potential is aligned with global initiatives, such as the United Nations' SDGs, which demonstrate how innovative racing technologies can have far-reaching impacts beyond the racetrack.

1.2.5 Fostering International Collaboration

The nature of autonomous racing is such that it is inherently collaborative, drawing on the talents of individuals from a diverse range of disciplines and backgrounds. A collective of engineers, scientists, programmers, and racing enthusiasts engage in the resolution of intricate challenges within a competitive framework. Such collaboration not only accelerates innovation but also fosters a global community that shares knowledge and breakthroughs, thereby contributing to the overall advancement of autonomous technologies.

1.3 Goals

This dissertation is concerned with two principal objectives. The initial objective is to develop an advanced real-time object recognition and tracking system that is specifically designed to address the distinctive challenges posed by the racing environment. It is essential that the system is robust, reliable and accurate, taking into account critical parameters such as camera specifications, variable lighting conditions and the distinctive characteristics of RC cars, including their size and colour schemes. The second objective is to develop a position/orientation estimation system for the vehicles to enhance the overall system's performance by refining and smoothing the data obtained from the camera sensor, thereby improving overall tracking accuracy and reliability. An Extended Kalman Filter was meticulously designed and implemented to fulfil this objective.

It is of the utmost importance that both objectives are implemented in a time-critical manner. The software developed for vehicle state estimation must be highly optimised and efficient, capable of delivering accurate state estimations to the Control Module with minimal latency. This real-time performance is essential to ensure the system's effectiveness in the dynamic racing environment.

1.4 Structure of the Document

The dissertation is structured into 5 chapters. After the Introduction chapter, the document is organised as follows:

- Chapter 2 provides background information and a comprehensive literature review on autonomous racing and perception using cameras as the main visual data collector. It also provides a detailed account of all the components used in the project.
- Chapter 3 describes the work done in the other two units of the overall project: Digital Twin and Control.
- Chapter 4 reports the work done in the Perception Unit. It lists every task executed, including camera calibration, vehicle detection, and vehicle position estimation. It also shows the results of each task.
- Chapter 5 draws the conclusions of this project and suggests avenues for future research, which could form the basis of subsequent thesis proposals.

Chapter 2

Literature Review

This chapter discusses the state of the art in autonomous racing and emphasises the crucial role of perception in this domain.

The subsequent sections delve into various project components, including the racing circuit, remote-controlled (RC) cars, printed circuit boards (PCBs), and cameras, highlighting their significance.

2.1 State of the art - Autonomous Racing

Before discussing autonomous racing, it is essential to acknowledge the rapid advancements in autonomous driving technology. Autonomous vehicles have recently witnessed considerable advancement, largely due to groundbreaking innovations in artificial intelligence, sensor technologies, and computing power. The primary objective in this domain has been to enhance self-driving cars' safety, efficiency, and convenience.

According to a projection by McKinsey, the autonomous driving industry could generate up to 400 billion euros in revenue by 2035, which has sparked a surge of interest and investment from a wide range of companies, from innovative start-ups to established automotive giants. [12]. One such company that is leading the charge in this space is Tesla, a pioneer in the realm of electric vehicles.

Tesla has been at the vanguard of the automotive industry in developing solutions to integrate autonomy into its vehicles. For instance, the *Autopilot* system represents an advanced driver assistance system that enhances the safety and convenience of driving by leveraging deep learning algorithms to provide a more sophisticated and responsive baseline advanced driver assistance system *(ADAS)*. This system significantly advances beyond traditional *ADAS* features, such as lane-keeping and adaptive cruise control. It incorporates a range of advanced capabilities, including automatic lane changes, traffic-aware cruise control, and the ability to navigate on-ramps and off-ramps on highways.

The continued advancements in Tesla's *Autopilot* system, along with broader progress in autonomous driving technology, have paved the way for the exploration of autonomous racing. This emerging field presents unique challenges and opportunities for self-driving vehicles.

Autonomous racing is a testing ground for pushing the limits of autonomous technology in high-speed and dynamic environments. It is an objective evaluation of the capabilities of autonomous technology in challenging conditions. Some key points of autonomous racing are:

- **Competition formats**: The *Roborace* series consists of races in which fully autonomous vehicles compete against each other. Equipped with advanced AI systems, these vehicles are designed for high-performance driving and navigating complex tracks without human intervention.

Figure 2.1: Robocar from Roborace designed by Daniel Simon.

The race format of *Roborace* was still in a testing phase before the project was abandoned. While *Roborace* may have faced setbacks, the broader field of autonomous racing continues to evolve, with new initiatives and competitions emerging to push the boundaries of self-driving technology, such as the Abu Dhabi Autonomous Racing League (*A2RL*).

A2RL represents a novel autonomous racing series that is pioneering the boundaries of self-driving technology. The series was conceptualised by ASPIRE and brought to fruition through the dedicated efforts of engineers, scientists, and programmers. The event is a STEM-focused race and global showcase for the cutting-edge of autonomous systems. The primary objective is to challenge the limits of autonomous vehicle technology, subjecting these vehicles to rigorous race-track testing to ensure their safety on public roads. Beyond this, the initiative also engages in research and development, exploring new frontiers and applications across a range of sectors [1]. Each year, teams from leading educational and technological institutions around the globe compete to develop the fastest and most capable racing artificial intelligence. The competition is intense, with a substantial prize pool of millions of dollars at stake. For the 2024 season, a sum of 2.25 million dollars is at stake, providing an incentive for the teams to push the boundaries of what is possible with autonomous vehicles.[1]

Figure 2.2: A2RL Race car [1].

- **Technology development**: Autonomous racing tests the limits of AI, machine learning, and robotics in extreme conditions, leading to innovations that can be applied to everyday autonomous vehicles. This promotes rapid progress in algorithms, sensor fusion, and decision-making capabilities.

- **Challenges**: Racing environments are challenging, as vehicles must make quick decisions, adapt to rapidly changing conditions, and interact with nearby vehicles, all at high speeds.

Autonomous driving and racing are advancing with a focus on safety, reliability, and performance. As technology progresses, more sophisticated autonomous systems are expected to be seen on the roads, and more exciting developments in autonomous racing will emerge. Two notable instances of this advancement are the Formula Student 'Driverless' competition and the F1tenth competition.

The Formula Student 'Driverless' competition is an international engineering competition in which student teams engage in the design, construction, and racing of small-scale, formula-style racing cars. The 'Driverless' category specifically concerns autonomous vehicles, requiring students to develop sophisticated algorithms and systems for vehicle control, perception, and navigation. This competition advances the frontiers of autonomous racing, offering a forum for advancing innovative technologies and their practical applications [9].

Similarly, the F1tenth competition requires teams to construct autonomous race cars at a 1/10 scale. It is a requirement for participants to implement sophisticated algorithms for perception, planning and control to navigate complex race tracks autonomously. The F1tenth competition places significant emphasis on the technical aspects of autonomous driving while also underscoring the paramount importance of safety and reliability in high-speed racing scenarios. It functions as a miniature representation of the actual challenges encountered in the field of autonomous vehicles, thereby stimulating research and development activities [5].

These competitions demonstrate the accelerated progress and mounting interest in autonomous racing, exemplifying the prospective developments and applications in both racing and the broader domain of autonomous driving technologies.

2.2 State of the art - Perception

Perception, in the context of autonomous vehicles, refers to using sensors[1] (cameras, LIDARs, Radars, ...) to gather real-time environmental data. This technology has evolved significantly and is crucial in modern autonomous driving systems.

Cameras are one of the primary sensors used in a vehicle's perception. They have been used in vehicles for several decades, primarily for rear and side vision assistance. However, the integration of cameras into autonomous driving systems began with the development of computer vision and image processing technologies. The field of computer vision, which focuses on enabling computers to gain a high level of understanding from digital images or video, became fundamental in developing algorithms to recognise objects, identify lane markings, recognise traffic signs and assess depth and distance in the environment using camera data. Developing deep learning and convolutional neural networks (CNNs) has also improved the accuracy of camera-based perception systems. These new techniques have enabled more sophisticated object detection, for example.

Perception in autonomous racing plays a vital role in ensuring that vehicles can manoeuvre through complex tracks at high speeds. There are several advantages of using this technology:

- **Real-time decision making**: Racing environments demand split-second decisions. Cameras, often in combination with other sensors, provide real-time data about the track, obstacles and other vehicles. This information allows racing cars to make quick decisions, adjust trajectories and optimise racing lines.
- **Precision**: Depending on the camera used, it can offer high-resolution and high-fidelity data, enabling precise location.
- **Cost-Efficiency**: Compared to LIDAR [19], cameras are more cost-efficient while providing robust information about the environment, reducing the overall cost of autonomous systems.
- **Adaptability**: Camera-based perception systems can adapt to different lighting conditions but with some limitations. These limitations can be mitigated by combining camera data with other types of sensor data using sensor fusion techniques.

2.3 General Contextualisation

As mentioned in chapter 1, the Autonomous Racings project is divided into three different dissertation projects. The perception system will detect all the cars on the track using a camera and estimate their positions and orientations using the information gathered from the camera, which will be fused with a Kalman filter for non-linear systems - EKF. This new information is then transmitted to all the control platforms via a UDP protocol. Each car will have its own microcontroller with an ESP32-S2FN4R2 WIFI IC.

[1] The table 2.1 will provide a comprehensive overview of the sensors.

The control system will then calculate the necessary control inputs for the cars and send them via a UDP protocol to the embedded car. A machine learning training method called Reinforcement learning [24] will be used for the car to learn and improve the trajectories on the track.

The final part of this project is the Digital Twin, a tool that visually represents a real-world system. It allows developers to structure and optimise their systems in a remote environment. At the same time, it can be directly connected to the physical system.

This project took inspiration from research developed by the Automatic Control Laboratory from ETH Zurich [13]. Figure 2.3 represents the overall view of the system and how its components are connected.

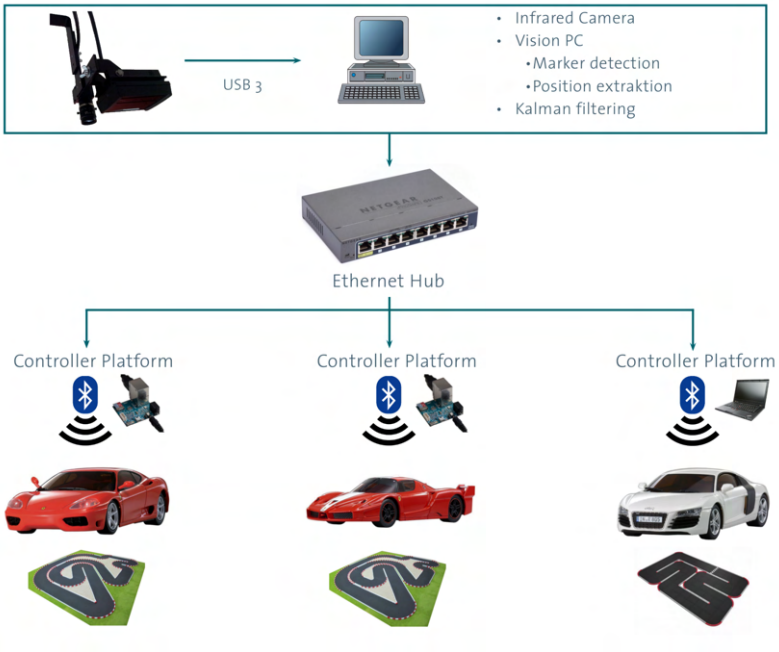


Figure 2.3: Overview of the system [13].

2.4 Components

This section discusses the different components that will be used in the autonomous racing project. The system will include a camera, a mini wifi board based ESP32-S2FN4R2, a racetrack, 1:43 scale remote-controlled cars, and a computer connecting the system. Each of these components plays a critical role in the system's overall functioning, and their integration will facilitate the development of a robust and reliable autonomous racing system. The importance of each component and its role in the system will be discussed in detail.

2.4.1 Racetrack & RC cars

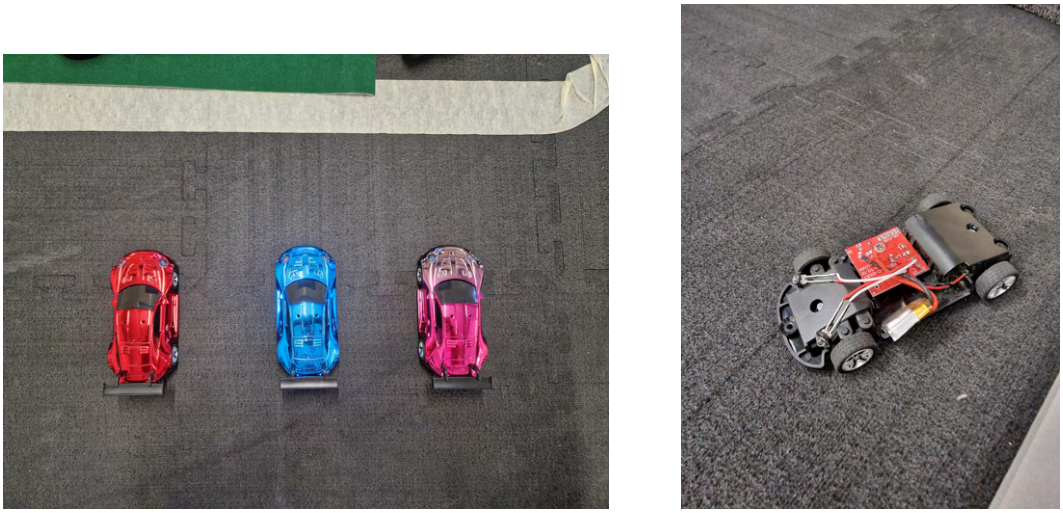
The track is made by 'Kyosho' [16] and measures approximately 2x2 metres. It is made of urethane to ensure good traction for the racing cars and prevent vehicle damage.

(a) Racetrack from Kyosho [16]. (b) Mounted Racetrack.

Figure 2.4: Racetrack.

The Laboratory had limited space, and the order did not contain the requisite number and type of parts to permit the assembly of the track following Figure 2.4a. Consequently, an alternative approach was adopted, as illustrated in Figure 2.4b.

The RC cars are 1/43rd scale diecast, which means they will be about 8-12 cm long. The cars were opened to put the microcontroller inside.



(a) 3 cars used. (b) Car's circuit board.

Figure 2.5: Cars.

Inside each car is a circuit board, Figure 2.5b, that controls various functions such as turning the car on and off, activating the headlights, charging the car, managing the motors and wheels, and communicating with the controller.

To achieve autonomous control of the RC car, the original circuit board has been supplemented with two additional components - an ESP32 S2 Mini microcontroller and a custom-designed control board. The ESP32 S2 Mini is a powerful, low-power microprocessor that will serve as the 'brain' of the autonomous system, generating control signals. The custom control board, on the

other hand, will directly interface with the car's motors and steering mechanism, translating the ESP32's commands into the physical movements of the vehicle.

The objective is to transform the manual RC car into a fully autonomous platform capable of navigating its environment and executing complex manoeuvres without direct human input. This will be achieved by adding new electronic components without altering the car's original form factor. The approach allows the original RC car design to be preserved while expanding its capabilities by integrating advanced control and sensing technologies.

2.4.2 ESP32 S2 Mini

The ESP32 S2 Mini microcontroller is the essential component that enables the autonomous control of the RC car. As the central processing unit of the autonomous system, the ESP32 S2 Mini is responsible for processing sensor data, making decisions, and generating the appropriate control signals to drive the car's motors and steering mechanism.

It is a powerful, low-power microprocessor that offers a range of features well-suited for this application. The microcontroller is equipped with a high-performance single-core Xtensa 32-bit LX7 microprocessor that can operate at up to 240 MHz, providing ample computational power to handle the complex algorithms and real-time decision-making required for autonomous racing [29].

In addition to its processing capabilities, the ESP32 S2 Mini also features extensive connectivity options, including Wi-Fi and Bluetooth Low Energy (BLE) support. This enables the microcontroller to communicate with the custom-designed control board directly connected to the car's motors and steering. The ESP32 S2 Mini can send control signals to the custom board, which then translates these commands into the physical movements of the vehicle [29].

It also offers a range of integrated peripherals such as analogue-to-digital converters (ADCs), digital-to-analogue converters (DACs) and various communication interfaces (UART, SPI, I2C, etc.). These features enable the microcontroller to seamlessly integrate with various sensors, such as cameras, LiDAR or ultrasonic sensors, which can provide the necessary environmental data for autonomous driving algorithms.

Finally, the ESP32 S2 Mini is designed with power efficiency in mind, making it an ideal choice for battery-powered applications such as autonomous RC cars. Its low-power modes and advanced power management features help to extend vehicle life and ensure reliable operation.

Here are some relevant features of the ESP32 S2 Mini for this project:

- **High clock speed**: The 240 MHz clock speed enables the ESP32-S2 Mini to efficiently handle complex computations and high-speed data processing.
- **Single-Core Architecture**: While it has a single-core processor, the Xtensa® LX7 is designed to be highly efficient and capable, making it suitable for a wide range of IoT applications.

- **Advanced Processing**: The CPU architecture is optimised for integer and floating-point operations, supporting up to 32-bit precision, which is useful for applications requiring mathematical computations.
- **Dimensions**: The board's dimensions are 25.4 mm in width and 34.3 mm in height, making it a perfect fit for the RC car.
- **Wireless communication**: It includes wireless connectivity options like WiFi, enabling seamless communication with other devices or networks.

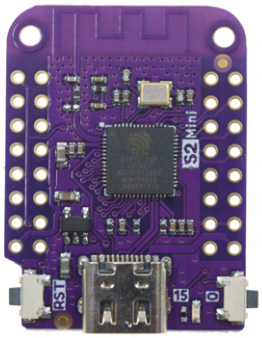


Figure 2.6: ESP32 S2 Mini [29].

The board is characterised by a compact and versatile design, offering a wide range of connectivity options for many applications. As illustrated in Figure 2.7, the board comprises 27 digital input/output (I/O) pins, which afford extensive interfacing with many electronic components and peripherals. The 27 digital I/O pins can be employed for various purposes, including controlling motors, reading sensor data, and communicating with other devices.

In addition to the digital I/O pins, the board also includes several other important pins. The EN pin is employed to enable or reset the board, while the 3.3V pin provides a stable power supply for the board and connected components. The VBUS pin provides both power and data communication via USB, enabling the board to be powered and programmed through a USB connection. Finally, the GND pin serves as the entire system's ground reference, ensuring proper electrical grounding and signal integrity.

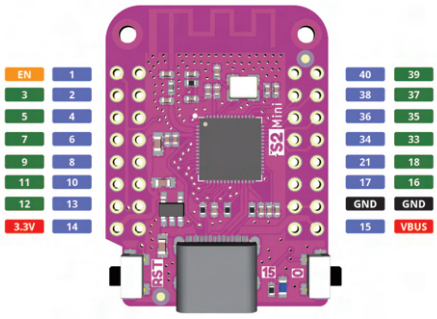


Figure 2.7: Pinout of the ESP32 S2 Mini [29].

The table A.1 on page 72 lists every technical spec of the board [29].

2.4.3 Steering and Motor Drivers

The customised printed circuit board (PCB) for an RC car's steering and motor drivers integrates several essential components to ensure efficient and reliable operation. This section elucidates the underlying principles of the key functional blocks, namely the voltage regulator, motor control, and electromagnetic steering control.

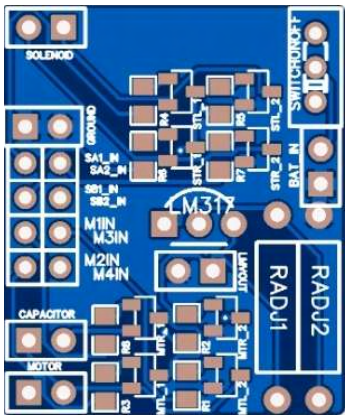


Figure 2.8: Steering and Motor Driver PCB.

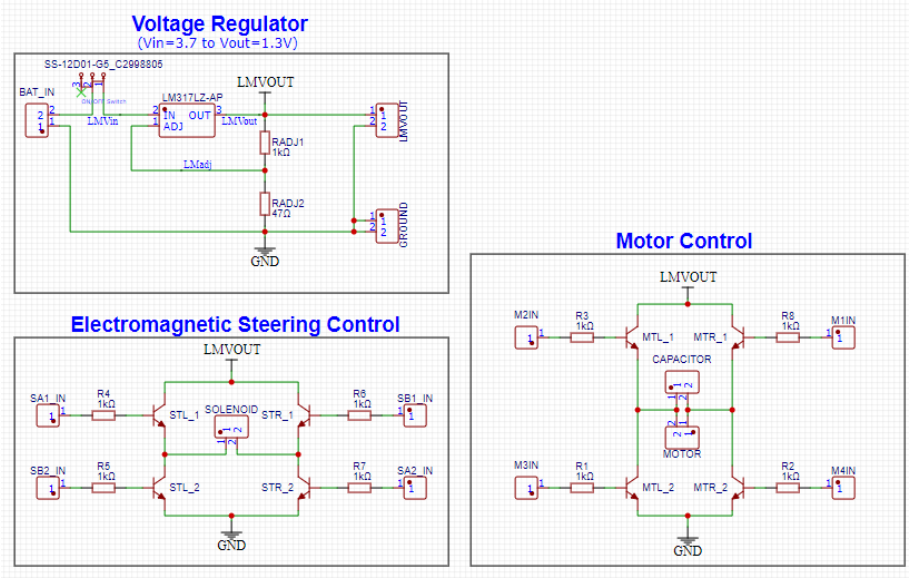


Figure 2.9: Electrical circuit of the PCB.

The primary power source for the system is a battery connected via the 'BAT_IN' terminal. The voltage regulation circuitry is centred around the LM317 linear voltage regulator, chosen for its versatility in providing an adjustable and stable output voltage. The regulator is configured with two adjustable resistors, R_{adj1} (1kΩ) and R_{adj2} (47Ω), which are critical in setting the desired

output voltage according to the formula:

$$V_{out} = V_{ref} * (1 + \frac{R_{adj1}}{R_{adj2}}) + I_{adj} * R_{adj1} \quad (2.1)$$

where V_{ref} is the reference voltage (typically 1.25V for LM317), and I_{adj} is the adjustment pin current. Capacitors are employed at the input and output terminals of the LM317 to filter out noise and ensure stability. The regulated output voltage, designated as LMVOUT, serves as the power supply for the subsequent motor and steering control circuits.

The motor control section drives the RC car's propulsion system. It interfaces with four input signals: M1IN, M2IN, M3IN, and M4IN, each corresponding to control signals for the motor driver. These inputs are processed through a network of resistors (R1, R3, R8, each 1kΩ) and a capacitor to mitigate electrical noise and provide a smooth driving signal.

The processed signals are then directed to the motor terminals (MTR_1 and MTR_2), which control the operational state of the motor, which can be driven in various modes, including forward, reverse, and braking, depending on the input signals, by appropriately switching the connections.

The steering mechanism of the RC car is facilitated by the electromagnetic steering control circuit. The circuit operates through a solenoid, an electromechanical actuator that converts electrical energy into linear motion, thereby steering the car.

The steering control circuit receives four input signals: The inputs are designated as SA1_IN, SA2_IN, SB1_IN, and SB2_IN. These inputs are processed via a network of resistors (R1, R4, R5, R7, each 1kΩ) to ensure that the control signals are appropriately conditioned before actuating the solenoid. Subsequently, the conditioned signals are applied to the solenoid terminals (STR_1 and STR_2).

A current is induced in the solenoid upon activation of the control signals, generating a magnetic field that induces linear motion. This motion is mechanically linked to the steering mechanism of the RC car, thus enabling directional control.

This PCB is connected to the ESP32 S2 Mini, thereby enhancing the control and functionality of the RC car. As previously stated in section 2.4.2, this board is a highly capable microcontroller with integrated Wi-Fi capabilities, enabling wireless communication with the RC car. The device interfaces with the motor and steering control sections via its general-purpose input/output (GPIO) pins. The motor control signals (M1IN, M2IN, M3IN, and M4IN) and the solenoid control signals (SA1_IN, SA2_IN, SB1_IN, and SB2_IN) are generated by the ESP32 S2 Mini based on the user commands received wirelessly. Subsequently, the commands are processed, translated into appropriate control signals, and transmitted to the PCB via a serial communication protocol to drive the motor and control the steering.

2.4.4 Camera

Common sensors for intelligent driving systems include cameras, millimetre wave radar, and lidar. The table 2.1 compares the advantages and disadvantages of each sensor.

Table 2.1: Camera vs Radar vs Lidar.

Camera	**Advantages**	captures high-resolution images providing rich contextual information about the environment
		relatively inexpensive, making it a cost-effective solution
		works well in various lighting conditions and can detect colour, texture, and patterns, providing valuable visual cues
	Disadvantages	Performance can degrade in extreme lighting conditions like low light, glare, or adverse weather, impacting accuracy
		Accurately gauging distances and depth perception is challenging when compared to LiDAR, which affects precise object localization
		Cameras might have a limited range
Radar	**Advantages**	Radar is less affected by adverse weather conditions such as rain, fog, or dust compared to cameras
		Radar systems can detect objects over long distances, making them suitable for long-range object detection
		Capable of measuring relative velocities accurately, aiding in collision avoidance and tracking moving objects
	Disadvantages	Radar lacks the resolution and detailed object recognition capabilities of cameras and LiDAR
		Radar signals can bounce off surfaces, leading to false readings or difficulty in distinguishing multiple objects
		Identifying the nature or type of objects can be challenging for radar alone
Lidar	**Advantages**	LiDAR offers precise 3D mapping and accurate distance measurement, enabling detailed object localization
		Works well in various lighting conditions and can provide detailed data even in low light or darkness
		Capable of identifying objects with high accuracy and distinguishing between multiple objects
	Disadvantages	LiDAR systems tend to be expensive compared to cameras and radar
		LiDAR can still be affected by adverse weather conditions like heavy rain or fog
		Some LiDAR systems have moving parts that can be prone to wear and tear, impacting reliability

After analysing the strengths and limitations of these three sensors, it was concluded that the camera is the best option for this project. The sensor will always be in the same position and environment (indoors), meaning lighting and weather will not be a problem. Additionally, the cost was a determining factor in choosing the camera. As stated in section 2.2, the camera plays a crucial role in the system. It serves as the system's eyes, detecting the car's location, potential obstacles, track boundaries, and other cars on the circuit.

At the beginning of the project, the primary issue was the camera's sub-optimal image quality and frame rate. To address these deficiencies, it was necessary to identify an alternative camera. After evaluating several options, the GoPro Hero 10 Black emerged as the most promising choice.

Figure 2.10: First camera used.

GoPro, Inc. is a renowned American technology company founded by Nick Woodman in 2002. The company is celebrated for its action cameras, designed to capture high-definition video and photographs, particularly in extreme environments. Over the years, GoPro has established itself as a leader in portable, durable, and high-quality imaging devices. The GoPro Hero series has gained considerable acclaim for its innovative features, user-friendly design, and robust performance [23].

The decision to utilise the GoPro Hero 10 Black in this project was influenced by many key factors. Firstly, the image quality of the GoPro Hero 10 Black is exceptional, with a 23.6 MP sensor that delivers images of remarkable clarity and detail. This significant enhancement over the previous camera ensures that the project can achieve the high level of visual fidelity required.

Secondly, the GoPro Hero 10 Black has been designed to be highly user-friendly and easy to handle. The camera's compact and lightweight design makes it convenient for deployment in various scenarios, particularly those requiring mobility and flexibility. This ease of use is paramount for maintaining efficiency and accuracy throughout the project.

Another critical factor was the price and the availability compared to other cameras searched. The GoPro Hero 10 Black costs 300 euros, similar to the other cameras. The GoPro Hero 10 Black is a cost-effective option that offers a high level of functionality. Given the need to purchase a new camera and the desire to avoid lengthy wait times, the GoPro was a convenient choice due to its availability in Portuguese marketplaces.

Another significant advantage of the GoPro Hero 10 Black is its enhanced frame rate, which allows for smooth and detailed video capture of fast-moving subjects, such as the RC cars. The camera can record 5.3K video at 60 frames per second (fps) and 4K video at 120 fps, providing smooth and high-quality footage. This enhancement is vital for capturing fast-moving subjects and ensuring no detail is lost due to lower frame rates. Furthermore, the GoPro Hero 10 Black features a webcam mode, which allows it to be easily connected to a PC. In this mode, the quality is somewhat diminished, yet the camera can still record 1080p video at 60 fps, which is still of a high standard. This functionality facilitates the integration of the camera with other project components, enabling seamless data transfer and live streaming capabilities.

Finally, the ease of integrating the GoPro Hero 10 Black with the code was a significant consideration. GoPro provides extensive support and documentation for developers, including APIs and SDKs, which facilitate the integration of the camera into various software environments. This support ensures that the camera can be effectively used with this project's custom code to meet the project's specific needs.

Figure 2.11: GoPro Hero 10 Black.

The table A.2 highlights the most relevant specifications of the GoPro for this project.

Chapter 3

The Digital Twin and Control Modules

This chapter examines the intricate work undertaken by the Control unit and the Digital Twin unit, which play pivotal roles in developing autonomous driving capabilities.

The Control unit translates the vast perception data into precise and real-time steering, throttle, and braking inputs. This process involves the implementation of sophisticated control algorithms that can rapidly process sensor data, accurately predict the vehicle's future state, and make split-second adjustments to the control inputs with utmost precision. The objective of the control unit is to learn the trajectory of the autonomous vehicle and then to improve the trajectory, just as a skilled human driver would, while maintaining unwavering stability and control, ensuring a seamless and safe driving experience.

In conjunction with the Control unit, the Digital Twin component generates a comprehensive digital representation of the autonomous racing system. This encompasses the vehicle dynamics, sensor suite, and control algorithms. This digital replica enables extensive simulation, testing, and optimisation of the system before deployment on the physical racetrack. The Digital Twin enables the exploration of a wide range of scenarios, the refinement of algorithms, and the validation of system performance in a safe and controlled virtual environment. This reduces the risk of physical hardware failure and enhances the overall reliability and robustness of the autonomous driving capabilities.

3.1 Digital Twin

As mentioned in section 1 (Introduction), the main objective of the Autonomous Racing Digital Twin project is to create a virtual representation of a real-world system, specifically, an autonomous racing car and its environment. This digital twin will enable developers to optimise the performance and decision-making processes of the race car in a remote environment. The digital twin aims to improve monitoring and analysis, allowing the system to adapt and make real-time decisions based on simulations that reflect real-world conditions.

The digital twin's development for the autonomous racing project was structured into three key phases: modelling, monitoring, and integration with the Control unit. This chapter provides a detailed description of each phase, elucidating the methodology and technologies employed.

3.1.1 Modelling

The first step in developing the digital twin was to create accurate physical models of the track and race cars. The physical modelling phase focused on replicating the real-world parameters and dynamics to ensure that the digital twin could deliver reliable simulations.

- **RaceTrack Modelling**: The race track was meticulously designed to replicate the real-life track, including precise measurements and similar curvatures.

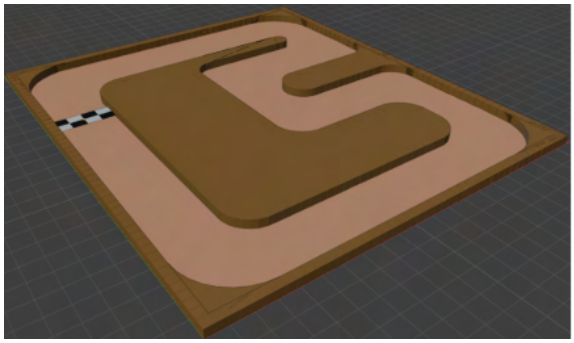


Figure 3.1: Modelled racetrack.

- **Race Car Modelling**: The race cars were modelled with detailed attention to their mechanical properties. This included the car's weight distribution and its components size. The goal was to create a virtual car that responded to controls and environmental conditions in a manner identical to the real vehicle.

3.1.1.1 Race Car Modelling

The race cars were the most complex objects to replicate in this project. Since the race track is flat, the focus of the car modelling was primarily on collision dynamics with the race track walls. Accurate replication of the race car involved several key measurements available in table 3.1.

Table 3.1: Measurements of the car's components.

Car's Components	Measurements
Front Wheels	diametre: 1.6cm width: 0.55cm
Rear Wheels	diametre:1.8cm width: 0.6cm
Wheel Track	Front: 4.23cm Rear: 4.28cm
Wheelbase	6cm
Front Overhang	2.6cm
Car Weight	48g
Center of Mass	center of the car

The aforementioned measurements were employed to create a three-dimensional model of the vehicle in Blender[1]. The process commenced with constructing a central axle cylinder, the foundation for all subsequent components. The front wheels were modelled using three cylinders, one for each wheel and one connecting them. These were subsequently merged into a single object and positioned at the front end of the axle. The rear wheels were modelled similarly, with adjustments made for their larger size.

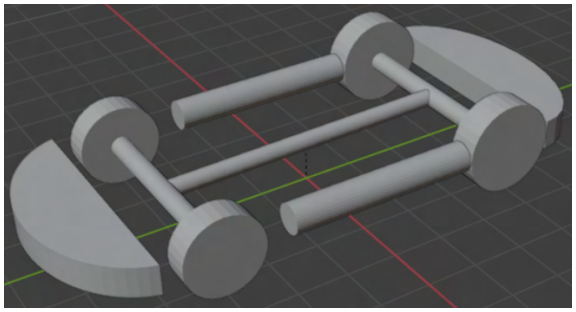


Figure 3.2: Modelled race car.

The car's body, designed to collide with the walls, was modelled with two cylinders on each side, matching the width of the rear wheels. These cylinders were placed near the rear wheels with sufficient space for front-wheel steering. The front and rear structures were created using cylinders sized to fit the wheel tracks and merged to form a single object.

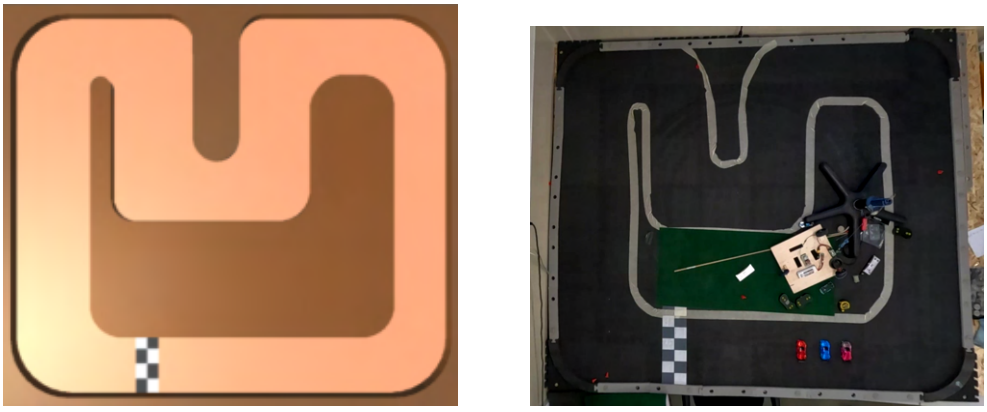
All components were parented to the central axle cylinder to ensure unified movement, and movement scripts were assigned to this central structure. This meticulous assembly process ensured the accurate simulation of the car's physical interactions and dynamics.

[1]Blender is an open-source computer program, developed by the Blender Foundation, for digital modelling and sculpting, texture mapping and UV mapping, rigging, animation, fluid and smoke simulation, rendering, illustration, compositing, motion tracking, and video editing [2].

3.1.1.2 Modelling in Unity

Unity[2] was chosen as the primary platform for the digital twin due to its robust 3D rendering capabilities and versatility. The modelling process in Unity involved:

- **3D Environment Creation**: The race track and surroundings were rendered in great detail, providing a visually accurate and immersive experience.
- **Physics Engine Integration**: Unity's physics engine was utilised to simulate real-world dynamics, ensuring that the virtual race cars behaved consistently with their physical counterparts.
- **Sensor Simulation**: In the Unity environment, simulations are observed through cameras, which are treated similarly to other objects. To facilitate an optimal viewing experience during the simulations, a camera was positioned above the race track to reflect the position of its real-life counterpart. This configuration permits the entire race track to be visible simultaneously, which is of paramount importance for the monitoring component of the digital twin.



(a) Camera view of the racetrack in Unity. (b) Real-life camera view of the racetrack.

Figure 3.3: Comparison between camera view in Unity with real-life.

3.1.2 Monitoring

The monitoring phase involved setting up a system to collect real-time data from the physical race cars and the virtual environment. This data was crucial for validating the digital twin and training the machine learning models. Key components included:

- **Data Acquisition System**: This communication concerns the reception of data regarding the coordinates and the direction each vehicle faces.
- **Real-Time Data Integration**: Data from the physical cars were streamed to the digital twin in real-time using UDP protocol, allowing immediate analysis and feedback.

[2]Unity is a cross-platform game engine developed by Unity Technologies

3.1.3 Integration with Machine Learning Framework

The final step in implementing the digital twin was incorporating the ML-Agents machine learning framework. This open-source tool enabled reinforcement and imitation learning to enhance the car's behaviour, leading to more efficient trajectories and reduced lap times.

3.1.3.1 ML-Agents Implementation

The ML-Agents package was imported into Unity and tested by creating an environment with a plane, cube, and spherical agent 3.4. A script assigned to the agent allowed it to move and receive rewards for colliding with the cube. Initially, the agent fell off frequently, but after training, it became proficient at pursuing the cube without falling, demonstrating the ML-Agents library's effectiveness.

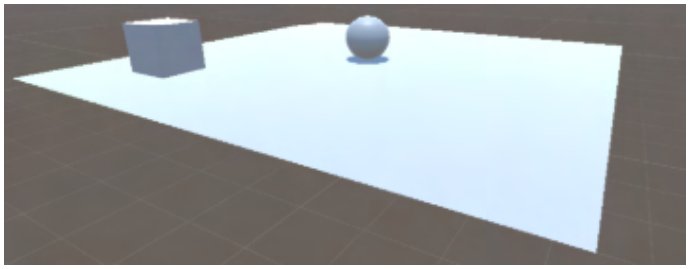


Figure 3.4: Environment for ML-Agents package testing.

3.2 Control

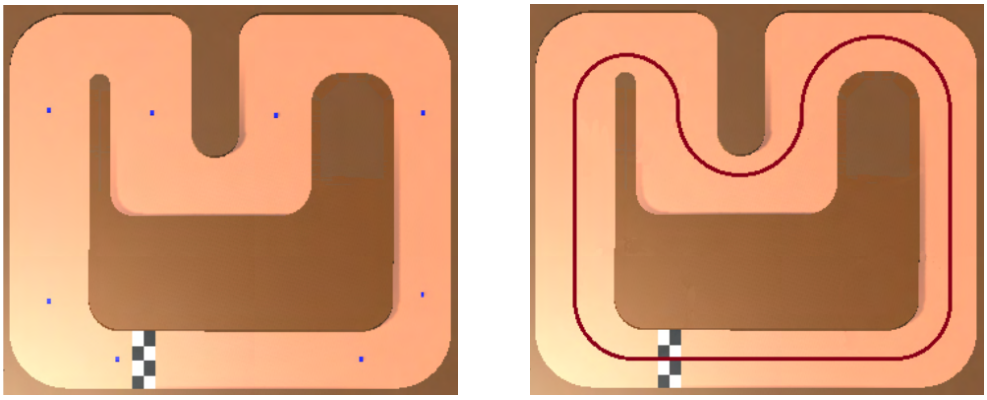
In the field of autonomous racing, the Control project is focused on the development and optimisation of a 1:43 scale autonomous vehicle capable of navigating a race track with high precision and efficiency. This project addresses the significant challenges posed by autonomous racing, including maintaining control at high speeds, managing rapid accelerations, and making split-second decisions. These challenges are further compounded by the necessity for the vehicle to operate at the limits of its performance capabilities in unpredictable and competitive environments.

3.2.1 PID

The project's initial phase involved the implementation of a basic Proportional - Integral - Derivative (PID) controller in a simple simulation environment developed using the Pygame library. The controller is of fundamental importance for maintaining the vehicle's position on the track, ensuring it remains centred and stable, particularly at lower speeds. Specifically, the PID controller was designed to adjust the steering angle and throttle continuously in response to real-time feedback from the camera data. The proportional component of the controller facilitated the correction of the discrepancy between the desired and actual positions of the vehicle, thereby ensuring an immediate response to deviations. The integral component accumulated past errors to eliminate residual steady-state errors. In contrast, the derivative component anticipated future errors based on the

current rate of change, thus providing a damping effect to reduce overshoot. This comprehensive approach permitted the vehicle to maintain a stable and centred path on the track, particularly effective at lower speeds and during gradual manoeuvres.

The PID implementation was further refined to suit a more complex and realistic Unity simulation environment. This environment simulated the car's dynamics more accurately, approximating real-world conditions more closely than the earlier setup. The PID controller's primary task was to ensure the vehicle followed the track's centreline as closely as possible. This was achieved by calculating the cross-track error (CTE), which is the perpendicular distance between the car's current position and the nearest point on the pre-defined trajectory. The PID controller then adjusted the car's steering angle to minimise this error, keeping the car on track. To implement the PID controller effectively, checkpoints were strategically placed along the track, dividing it into segments such as straights and curves. The trajectory segmentation allowed the system to define a clear path, facilitating the accurate calculation of the CTE. The PID controller utilised this error to continuously adjust the steering, ensuring that the car corrected its path whenever deviations occurred.



(a) Checkpoints placed in the Racetrack (blue dots).

(b) Trajectory segmentation.

Figure 3.5: The blue checkpoints allow trajectory segmentation.

The updated PID implementation in Unity significantly improved the car's ability to follow the desired trajectory. However, there were instances of delayed recovery at the end of curves. These delays were attributed to the need for further tuning of the PID parameters and adjustments to the vehicle's maximum steering angle.

Despite the effective performance of the PID controller, achieving optimal results required balancing recovery time and trajectory oscillation. A reduction in recovery time could result in a more oscillatory path, which was undesirable. Therefore, a slower recovery time was preferred to maintain a smoother trajectory. This nuanced tuning of the PID parameters underscored the importance of finding the right balance to achieve stability and responsiveness in the vehicle's navigation.

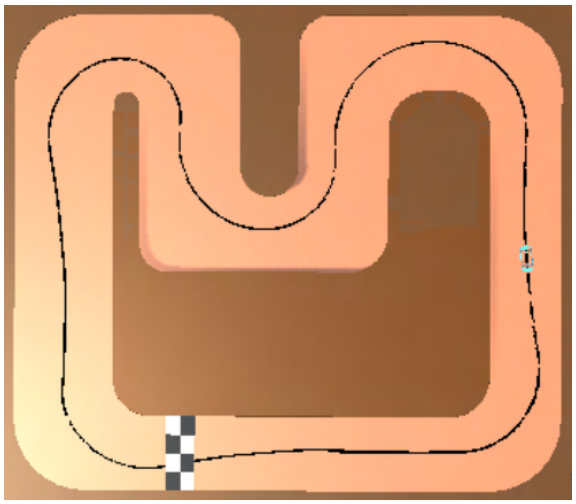


Figure 3.6: Car's path with updated PID.

The successful implementation of the PID controller provided a robust and reliable foundation for further enhancements using more advanced control methods such as imitation learning and reinforcement learning algorithms.

3.2.2 Imitation Learning

Imitation learning is a machine learning technique where a model is trained to replicate the behaviour of an expert by learning from their demonstrations. This technique is particularly useful in scenarios where defining explicit rules for every possible situation is impractical. In autonomous driving, imitation learning involves collecting data from human drivers or well-performing control algorithms and then training a neural network to mimic their actions based on the observed state of the environment. The model learns to predict the appropriate actions to take in various situations, thereby acquiring the ability to perform complex tasks by imitating expert behaviour[20].

Following the implementation of the PID controller, the objective was to utilise imitation learning as a replacement, bridging the gap between classical controllers and those based on reinforcement learning. Initially, trials commenced with a state size of six, encompassing the most recent six instances of cross-track errors encountered by the vehicle. However, the results were not as optimal as had been hoped. To address this, the state size was increased to 12, which resulted in visible improvements, validating the hypothesis that a larger state size enhances performance. Further experimentation with a state size of 20 did not yield additional improvements, indicating that a performance threshold had been reached.

Ultimately, a state size of 12 was selected as it represented an optimal balance between complexity and performance. The neural network architecture, Figure 3.7, comprised an input layer with a size of 12, followed by three hidden layers: the first with 1024 neurons and the second and third with 512 neurons each. The output layer comprised a single neuron, representing the value from the PID controller.

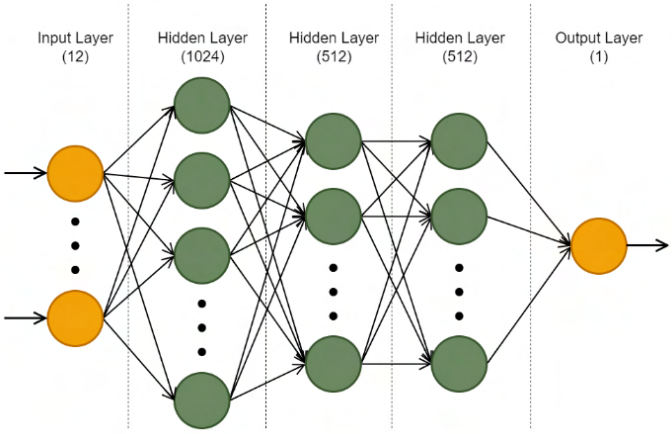


Figure 3.7: Diagram of the Neural Network.

The network was trained using two million data points over 500 epochs, which demonstrated the PID controller's response to various cross-track errors. This training enabled the neural network to effectively mimic and replace the traditional PID controller. The efficacy of this replacement is demonstrated in Figure 3.8, where the intended navigation path is depicted in green, and the path controlled by the neural network through imitation learning is represented in red.

Figure 3.8: CNN controlled path.

3.2.3 Reinforcement Learning

Reinforcement learning (RL) is a machine learning paradigm where an agent learns to make decisions by performing actions in an environment with the objective of maximizing cumulative reward. In contrast to supervised learning, which is dependent on labelled data, RL is predicated on the notion of learning through interaction. The agent receives feedback in the form of rewards or penalties based on the actions it takes, allowing it to learn optimal behaviour through trial and error. This process involves three key components: the policy, which defines the agent's behaviour; the reward function, which provides feedback; and the value function, which estimates

the long-term benefits of different actions [24]. In the context of autonomous racing, RL enables the vehicle to continuously improve its performance by learning from its experiences on the track.

The project examines the potential of reinforcement learning to replace the PID controller and further enhance the vehicle's performance. The objective was to train the autonomous vehicle to navigate the racetrack efficiently, intending to maximise the cumulative reward. This was defined in terms of minimising lap times and maintaining track adherence. The reinforcement learning model was trained using the Proximal Policy Optimisation (PPO) algorithm, which is renowned for its robustness and efficiency in handling complex control tasks.

PPO is a sophisticated reinforcement learning algorithm that addresses complex environments with intricate state and action spaces. PPO is designed to maximise the expected cumulative reward over time, represented by the objective function $G(\theta)$:

$$G(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} \gamma^t R_t\right] \tag{3.1}$$

This function incorporates three key elements: the policy parameters (π_θ), the discount factor (γ), and the reward at time $t(R_t)$ as seen in equation 3.1. PPO introduces a sophisticated surrogate objective function, $L^{CLIP}(\theta)$, to ensure stable and efficient learning. This function incorporates a clipping mechanism ($CLIP$) that prevents drastic policy updates that could disrupt the learning process. The surrogate objective function, $L^{CLIP}(\theta)$, is defined as:

$$L^{CLIP}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t\right)\right] \tag{3.2}$$

This function maintains stability during training with its careful balance between exploration and exploitation. The ratio $r_t(\theta)$ compares the new policy probability with the old policy probability for a chosen action:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \tag{3.3}$$

The utility function $\hat{A}_t$, known as the advantage function, assesses each action's performance relative to the average action:

$$\hat{A}_t = Q(s_t, a_t) - V(s_t) \tag{3.4}$$

where:

- $Q(s_t, a_t)$: The action-value function, Q, represents the expected cumulative reward when starting from state s_t and taking action a_t. This quantifies the value of taking a specific action in a given state.
- $V(s_t)$: The state-value function, V, denotes the expected cumulative reward starting from state s_t without taking any specific action. It represents the average value of all actions from that state.

The PPO workflow involves several iterative steps. First, trajectories are collected through interaction with the environment. Advantage estimates are computed to measure action performance, and the surrogate objective is calculated. Policy parameters are updated using stochastic gradient ascent, respecting the trust region to maintain stability. This trust region limits the size of policy updates, ensuring a balance and avoiding abrupt changes that could hinder progress.

Hyperparameters, such as the clipping parameter (ε), the learning rate, and the number of iterations, must be carefully tuned to optimise PPO's performance in different environments [24, 21].

The $\mathbb{E}$ symbol in the equations for $G(\theta)$ and $L^{CLIP}(\theta)$ represents the expectation. In reinforcement learning, the expectation denotes a random variable's average or expected value (such as rewards or advantage estimates) over all possible outcomes, weighted by their probabilities.

In summary, PPO represents a robust and sophisticated reinforcement learning algorithm. The clipped surrogate objective and the algorithm's focus on the trust region contribute significantly to the stability and effectiveness of the algorithm in navigating the challenges of complex learning environments.

3.2.3.1 State Space

The training process involves the creation of a realistic simulation environment in Unity, which enables the vehicle to explore and learn from its actions safely. The state representation comprises a number of parameters, and the state determines the actions that the agent is able to undertake in order to maximise the reward and reach the desired goal. The reward function was meticulously crafted to motivate the desired behaviours, such as maintaining adherence to the track and penalising collisions or excursions off-track. Furthermore, these actions will have consequences on the subsequent states of the system, which the agent must learn to predict and exploit in order to optimise its behaviour over time. The system's state is presented below in Equation 3.5 with a complementary explanation of what each parameter represents:

$$X = \begin{bmatrix} s & e & \mu & v_x & v_z & w & f & \delta \end{bmatrix} \tag{3.5}$$

where the variable s represents the progress made on the track, which can be expressed as follows: $s = s_t - s_0$. The variable e denotes the current cross-track error (CTE), while the variable μ is the difference in angle between the car and the trajectory direction. This is illustrated in Figure 3.9. The velocity components in the x and z-axes are represented by (v_x, v_z), respectively. The angular velocity is represented by w, while the acceleration control input is represented by f, and the steering angle input is represented by δ.

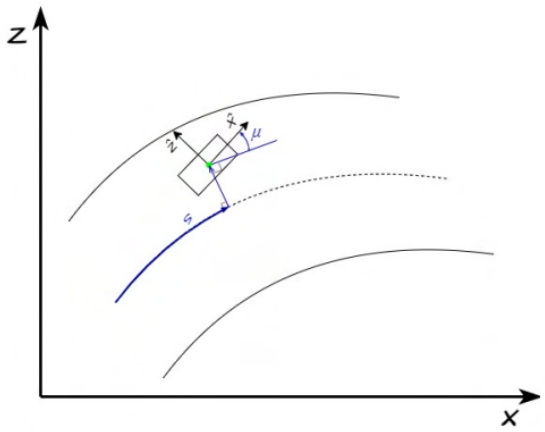


Figure 3.9: Difference in angle between the car and the trajectory direction.

The vehicle continuously interacts with the environment throughout the training period, making decisions based on its current state and receiving rewards or penalties accordingly. The PPO algorithm adjusts the policy to improve the expected cumulative reward over time. This iterative learning process allows the vehicle to develop sophisticated strategies for navigating the track, optimising its speed and trajectory to achieve faster lap times.

3.2.3.2 Training Process

The vehicle was trained using a server with an AMD Ryzen 7 5800X processor and 32GB RAM, which were selected for their processing power and capacity to handle large datasets efficiently. This configuration facilitated the acceleration of the training process, thereby enabling the rapid acquisition of skills within the simulated environment.

In each training episode, the car was randomly spawned near a checkpoint with varying initial positions, orientations, and velocities. Each episode lasted 360 ticks, allowing the vehicle to execute one or two turns per session. This constraint resulted in a steeper learning curve, facilitating a more rapid understanding of vehicle dynamics and adaptation to track challenges. It is possible that longer episodes may impede effective learning and comprehension of the environment.

In order to train the agent with ML-Agents in Unity, it was necessary to specify a number of crucial behaviour parameters. These parameters were instrumental in determining key aspects such as batch size, buffer size, learning rate, neural network structure, and training duration. The batch size determines the quantity of data that the agent processes prior to updating, while the buffer size specifies the amount of stored past experiences. The learning rate affects policy adaptation, while the neural network architecture influences data processing capabilities. The duration of the training period and the number of epochs per episode are essential for ensuring that the agent has sufficient time to learn and refine its strategies.

3.2.3.3 Reward function

The reward function provided an incentive for the vehicle to maximise the distance travelled without incidents. It rewarded progress on the track while penalising misconduct like collisions or reverse driving. This balance encouraged exploration and safe navigation, allowing the reinforcement learning algorithm to improve decision-making, driving efficiency, and safety over time. The implemented reward function is presented in equation 3.6, where Δd is the progress made in the track between two consecutive cycles, $\Delta d = s_t - s_{t-1}$.

$$R = \begin{cases} -5, & \text{if hit a wall} \\ -1, & \text{if } \Delta d \text{ is negative} \\ \Delta d, & \text{if } \Delta d \text{ is positive/zero} \end{cases} \tag{3.6}$$

3.2.3.4 Results

Following the conclusion of the training programme, the agent demonstrated the capacity to navigate the environment in an effective, consistent and reliable manner, thereby completing the requisite task. The results of this training will be presented and discussed in greater detail.

The cumulative reward graph, Figure 3.10, steadily increases over training episodes, indicating the agent's improved performance through reinforcement learning. Furthermore, the increasing episode lengths, Figure 3.11, provide additional evidence to support this assertion, indicating that the agent is navigating the environment in a more efficient manner.

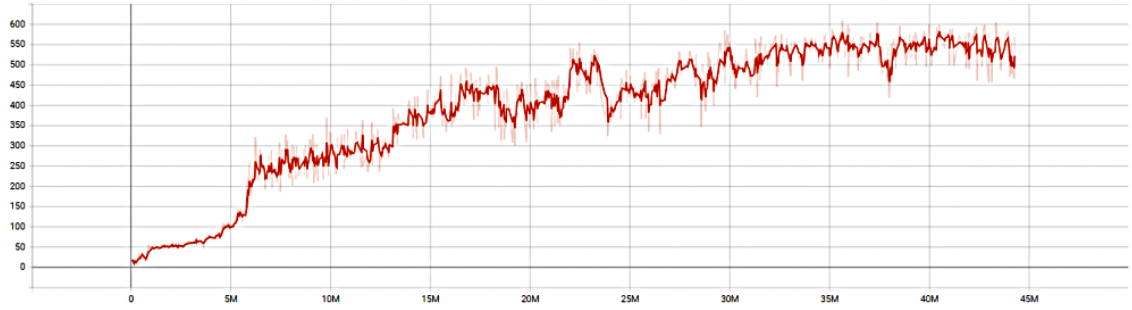


Figure 3.10: Cumulative Reward.

Figure 3.11: Episode Length.

The policy loss graph, Figure 3.12, illustrates the anticipated initial high loss, which subsequently declines as the agent learns. Thereafter, the graph stabilises, indicating that the policy is now stable. However, occasional spikes suggest that the agent is still exploring or that the policy is still unstable. The value loss graph, Figure 3.13 decreases over time, indicating enhanced value predictions. However, this decrease is accompanied by greater fluctuations than those observed in the policy loss graph. This is presumably due to the intricate nature of value estimation.

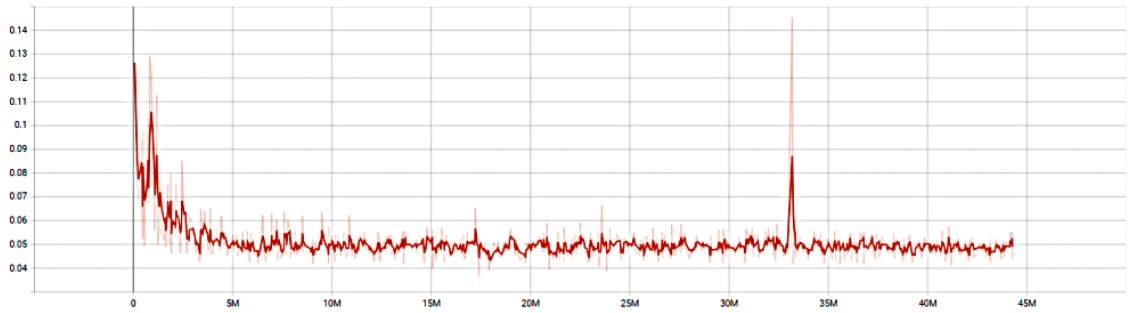


Figure 3.12: Policy Loss.

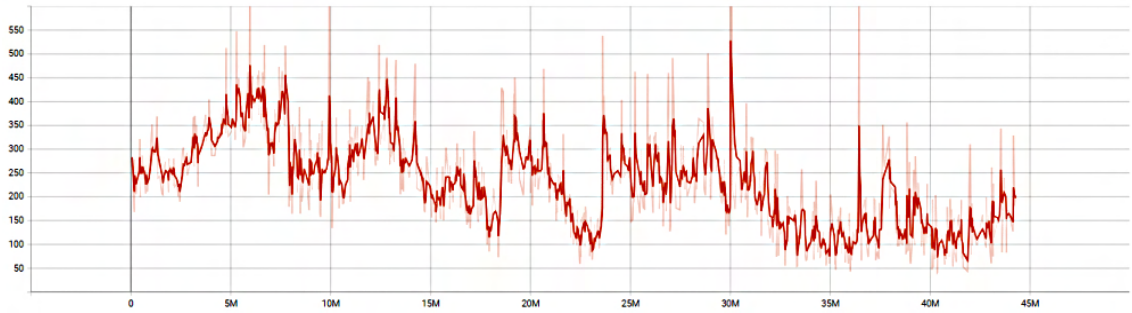


Figure 3.13: Value Loss.

Finally, the evolution of policy parameters over training episodes was examined. The beta coefficient, Figure 3.14, which represents the strength of entropy regularisation, decreases in a linear fashion.

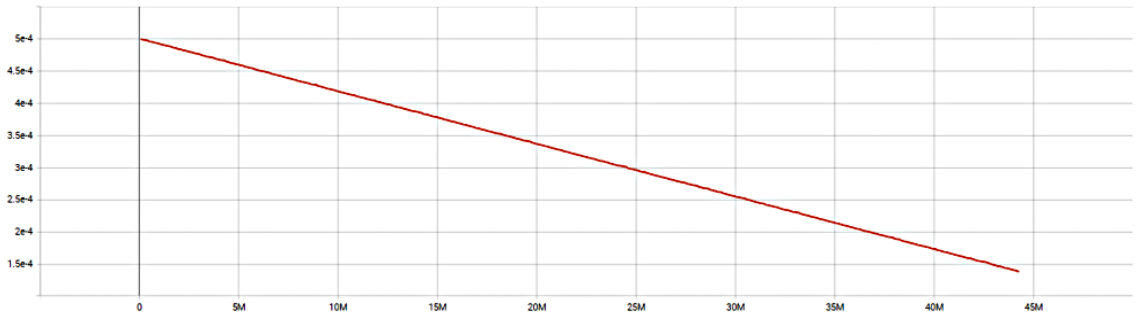


Figure 3.14: Policy Parameter: Beta.

The term 'Epsilon', Figure 3.15, is used to denote the acceptable threshold of policy divergence that occurs during updates.

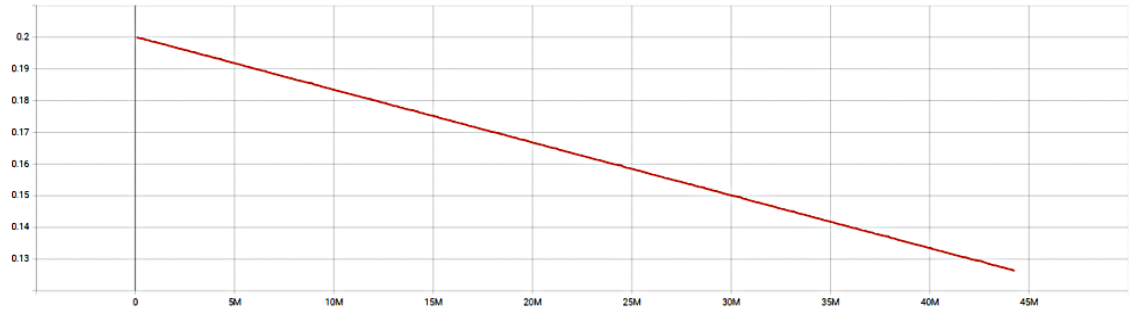


Figure 3.15: Policy Parameter: Epsilon.

The learning rate, Figure 3.16, is indicative of the strength of gradient descent steps.

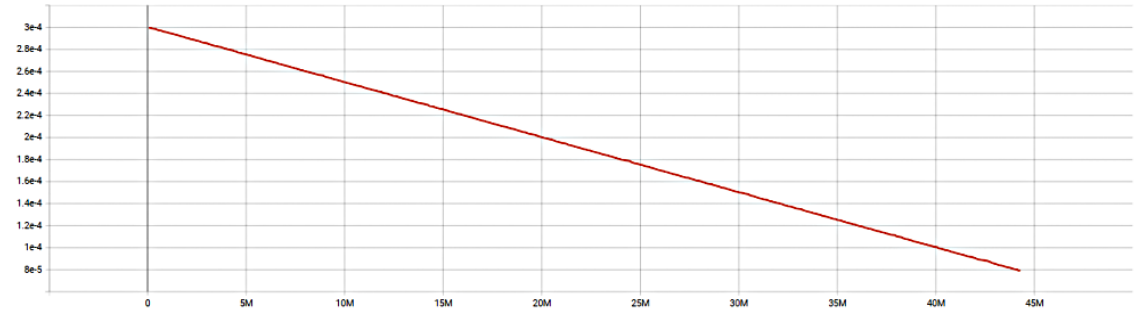


Figure 3.16: Policy Parameter: Learning rate.

Policy entropy, Figure 3.17, which measures action randomness, initially increases, peaks around 5 million steps, then stabilises and shows a slight decline. This indicates a shift from exploration to exploitation as the agent learns effective actions.

Figure 3.17: Policy Parameter: Entropy.

The extrinsic value estimation of the policy, Figure 3.18, demonstrates the agent's expected cumulative reward. It exhibits a rapid initial increase, followed by a continued rise with fluctuations, indicating the implementation of ongoing strategy adjustments. The eventual stabilisation suggests that the agent has developed a consistent understanding of the environmental rewards.

Figure 3.18: Policy Parameter: Extrinsic Value.

Chapter 4

Development of the Perception System Module

This chapter discusses the work done in this dissertation project. Firstly, the camera calibration and image distortion are addressed. Subsequently, it follows a description of how the system detects the RC cars on the racetrack. Finally, the chapter presents the EKF filter for the car's position estimation.

4.1 Camera calibration

Geometric camera calibration, also known as camera resectioning, is a technique that estimates the parameters of a camera's lens and image sensor. These parameters can subsequently be employed to rectify lens distortion, obtain the size of an object in real-world measurements, or determine the camera's position within a scene. Such capabilities have several applications, including machine vision, which employs them for object detection and measurement. They are also used in robotics, navigation systems and three-dimensional scene reconstruction [14].

The parameters of a camera include intrinsic and extrinsic parameters and distortion coefficients. To estimate the camera parameters, it is necessary to have three-dimensional world points and their corresponding two-dimensional image points. These correspondences can be obtained using multiple images of a calibration pattern, such as a chessboard. Once the correspondences have been established, it is possible to solve for the camera parameters [14].

4.1.1 Camera Models

Camera models are mathematical representations used in computer vision to describe the process of image formation. Two of the most common models are the pinhole and fisheye models. Due to the profound distortion that a fisheye lens induces, the pinhole model cannot represent a fisheye camera. In accordance with [15], it is recommended that the pinhole model be employed for a maximum field of view (FOV) of up to 95°. In the case of higher maximum FOVs, it is advisable to apply the fisheye model. The GoPro camera utilises a variety of FOV modes. The selected mode

for this project was the "linear" mode, which has a maximum FOV of 94 degrees. Therefore, the pinhole model will be employed instead of the fisheye model.

4.1.1.1 Pinhole Camera Model

The pinhole camera model is one of computer vision's simplest and most widely used models. It assumes that light rays pass through a single point (the pinhole) and project onto an image plane, as shown in Figure 4.1.

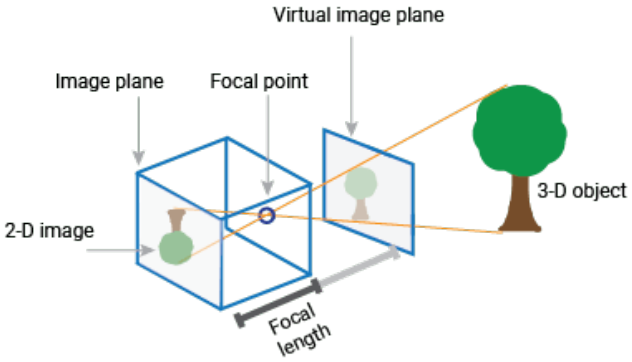


Figure 4.1: Pinhole Model [14].

The pinhole camera parameters are represented in a 3-by-4 matrix designated the camera matrix. This matrix maps the three-dimensional world scene into the image plane. The calibration algorithm calculates the camera matrix using the extrinsic and intrinsic parameters. The extrinsic parameters represent the location of the camera in the three-dimensional scene. The intrinsic parameters represent the optical centre and focal length of the camera. The world points are transformed into camera coordinates using the extrinsic parameters. The intrinsic parameters map the camera coordinates into the image plane.

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

$$P = K \begin{bmatrix} R & t \end{bmatrix} \tag{4.2}$$

Equation 4.1 represents the intrinsic parameters of the camera where ($f_x = \frac{F}{p_x}$, $f_y = \frac{F}{p_y}$) are the focal length in pixels, F is the focal length in world units, and (p_x, p_y) is the size of the pixel in world units. $\begin{bmatrix} c_x & c_y \end{bmatrix}$ is the optical centre (the principal point) in pixels.
Equation 4.2 represents the camera matrix. The extrinsic parameters consist of a rotation matrix, R, and a translation matrix, t. The origin of the camera's coordinate system is at its optical centre, and its x- and y-axis define the image plane [14].

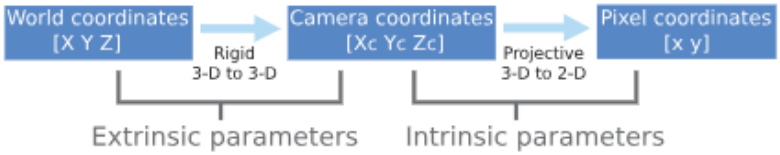


Figure 4.2: World coordinates to Pixel coordinates involving camera parameters [14].

4.2 Distortion in Camera Calibration

Removing image distortion from cameras is important to ensure the data captured is accurate and aligned with the real world. Distortion in cameras refers to imperfections in the lens system that result in deviations from the ideal image formation process. In this context, there are two main types of distortion: radial distortion and tangential distortion [3].

4.2.1 Radial distortion

This distortion results from non-uniform lens magnification across the image, which causes straight lines to appear curved. The distortion increases as points move farther from the centre of the image [3]. This type of distortion encompasses two distinct forms: pincushion distortion and barrel distortion. In pincushion distortion, image magnification increases with the distance from the optical axis. The visible effect of this distortion is that lines that do not pass through the centre of the image are bowed inwards towards the centre of the image like that observed in a pincushion. On the other hand, barrel distortion causes straight lines near the edges of an image to appear curved or bent outwards.

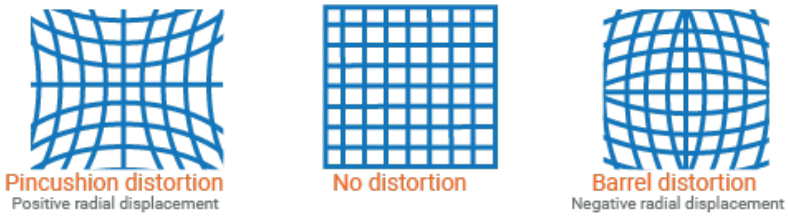


Figure 4.3: Different types of Radial Distortion [14].

The radial distortion coefficients model this type of distortion. The distorted points are denoted as $(x_{distorted}, y_{distorted})$ [14]:

$$x_{distorted} = x(1 + k_1 * r^2 + k_2 * r^4 + k_3 * r^6) \tag{4.3}$$

$$y_{distorted} = y(1 + k_1 * r^2 + k_2 * r^4 + k_3 * r^6) \tag{4.4}$$

$$r^2 = x^2 + y^2 \tag{4.5}$$

The coordinates x and y represent the positions of the undistorted pixels in the image. These coordinates are normalized image coordinates, which are calculated from pixel coordinates by

translating to the optical centre and dividing by the focal length in pixels. Therefore, x and y are dimensionless. k_1, k_2, k_3 are the radial distortion coefficients of the lens.

4.2.2 Tangential distortion

Certain areas of the image may appear closer than others due to the misalignment of the lens and the image sensor (or film) [3].

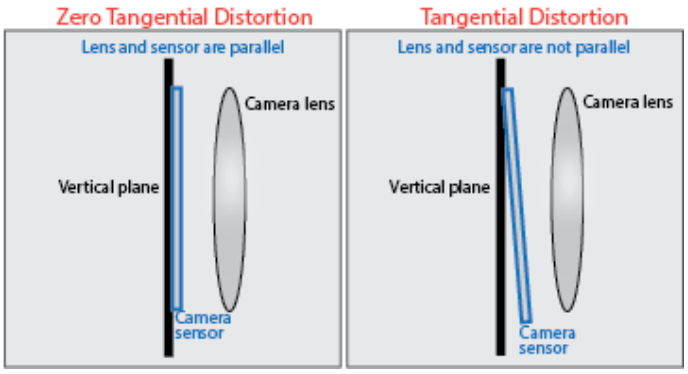


Figure 4.4: Tangential Distortion [14].

The distorted points are denoted as $(x_{distorted}, y_{distorted})$ [14]:

$$x_{distorted} = x + [2 * p_1 * x * y + p_2 * (r^2 + 2 * x^2)] \tag{4.6}$$

$$y_{distorted} = y + [2 * p_2 * x * y + p_1 * (r^2 + 2 * y^2)] \tag{4.7}$$

$$r^2 = x^2 + y^2 \tag{4.8}$$

The coordinates x and y are the same as in radial distortion. p_1 and p_2 are the tangential distortion coefficients of the lens.

4.3 GoPro Calibration and Undistortion

A straightforward and efficient method for camera calibration involves using a chessboard. This requires two essential sets of data: object points (3D real-world points) and image points (2D coordinates in the image). When dealing with a stationary camera and a chessboard positioned at various angles and orientations, the Z-coordinate remains constant (Z=0), provided that the chessboard is stationary on the XY plane. A grid-like structure has been used, with points assigned to denote their respective locations on the chessboard. For example, (0,0), (1,0), (2,0), etc. If the size of the chessboard squares is known (e.g. 30 mm), these points can be represented as (0,0), (30,0), (60,0), etc. This allows calibration results to be obtained in real-world dimensions (mm). However, if the square size is unknown, the points should be passed in terms of the square size to maintain their relative positions [3]. Image points are determined from the locations where two black squares on the chessboard meet in the images taken by the camera.

To start, the size of the chessboard pattern needs to be determined based on the number of inner corners, as shown in the following Figure 4.5.

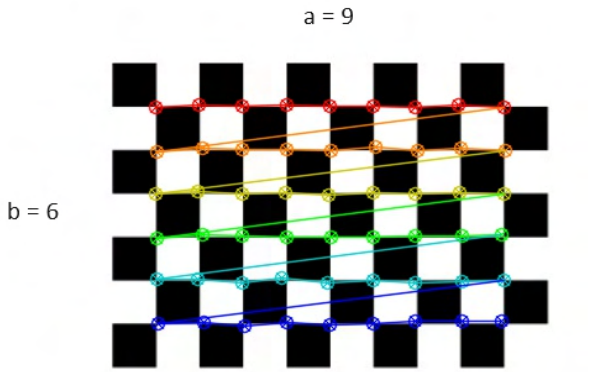


Figure 4.5: Parameters of the chessboard.

In this example, the size is $a = 9$ by $b = 6$. Two arrays are then created to store the 3D and 2D points. The corners of the chessboard can be detected using OpenCV's 'cv2.findChessboardCorners' function. If the corners are found, their accuracy can be increased using 'cv2.cornerSubPix()', and then the corresponding 3D and 2D points are added to their respective arrays. Finally, the camera is calibrated using the collected object points and image points by implementing the 'cv2.calibrateCamera()' function. This returns the camera matrix, distortion coefficients, and rotation and translation vectors. The code is available in the appendix B.1 and on the GitHub repository [18].

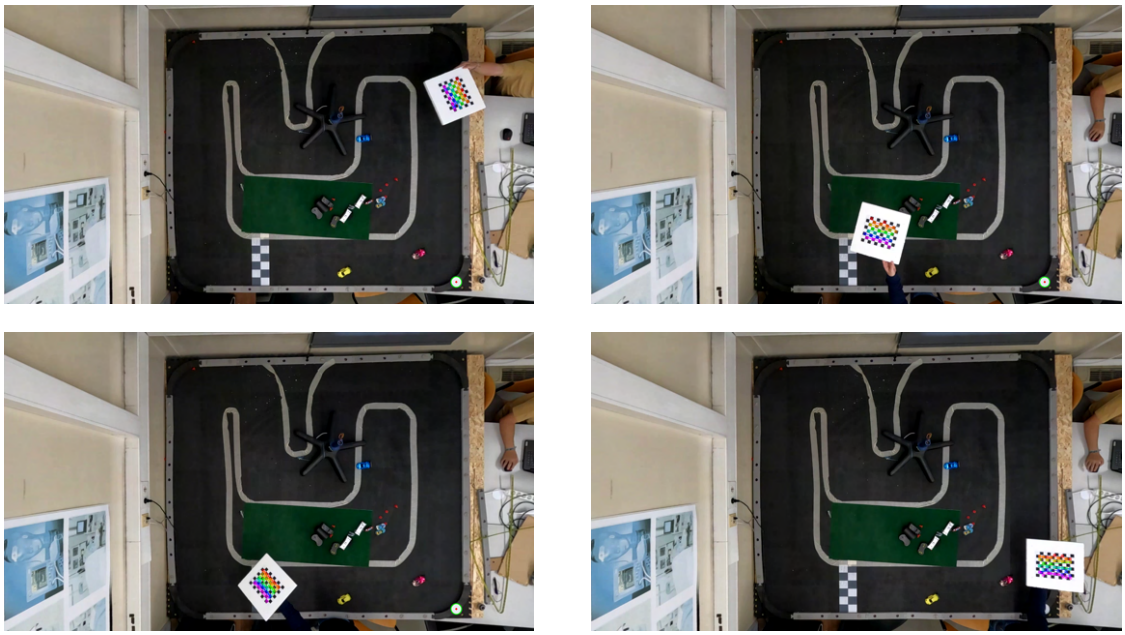


Figure 4.6: Some of the images with chessboard corners found.

Once the calibration process has been completed, removing the distortion from the image is the next step B.2, [18]. This is achieved by first computing a new camera matrix (newCameraMatrix) using the function 'cv2.getOptimalNewCameraMatrix()'. This matrix preserves all the useful parts of the original image while removing any regions that may be outside the valid range after undistortion. The region of interest in the undistorted image, where the valid pixels are located, is represented by the variable 'roi'. Subsequently, the 'cv2.undistort()' function applies distortion correction to the input image 'img' and stores the undistorted image in 'dst'. Finally, it calculates the reprojection error, which is a measure of the discrepancy between the detected image points 'imgpoints' and the reprojected object points 'objpoints' based on the estimated camera parameters 'rvecs', 'tvecs', 'cameraMatrix', 'distCoeffs'.

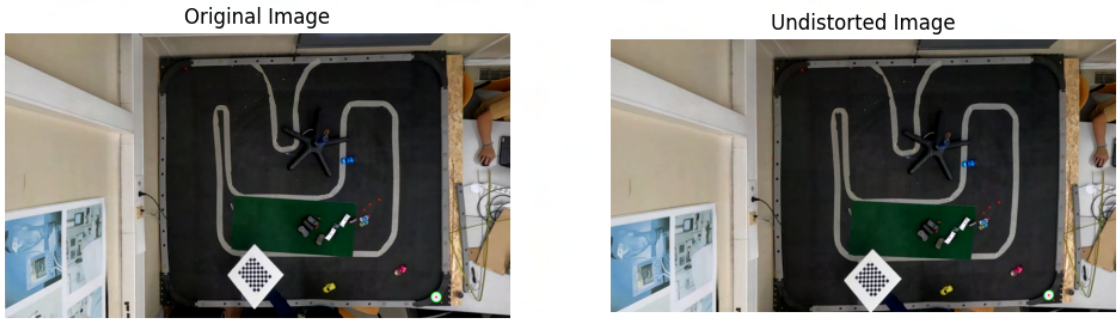


Figure 4.7: Comparison between original image with distortion with undistorted image.

Figure 4.7 compares the original image, captured by the GoPro in the 'linear' FOV, with the undistorted image following the calibration process. Due to the field of view being in the 'linear'

mode, the distortion is not particularly noticeable compared to when it is in the 'wide' mode. However, this does not imply that the image is free from distortion. The original image has a slight curvature to the edges of the scene, which can be attributed to lens distortion. This phenomenon can be observed in the curved lines of the track near the edges and the slight bulging effect near the corners. In the undistorted image, these curved lines appear straighter, and the overall scene looks more rectilinear. The lines of the track and the borders of the frame appear more uniform and straight, correcting the lens distortion present in the original image.

The reprojection error obtained was approximately 0.8 for all of the sample images. For typical computer vision tasks like object detection, tracking, or basic 3D reconstruction, a reprojection error below 1.0 pixels is often considered acceptable[1]. This ensures that the distortion correction is sufficiently accurate for these purposes [3, 8].

4.4 Vehicles detection

Pose estimation is crucial in numerous computer vision applications, including autonomous racing. This process is based on finding correspondences between points in the real environment and their 2D image projection. Synthetic or fiducial markers are often used to simplify this challenging step [4].

The method suggested by the Automatic Control Laboratory from ETH Zurich [13] was a possibility at the beginning. The system used an infrared LED and an infrared light filter to detect cars. The camera would only pick up this type of light, and retro-reflective markers on the cars would reflect only this light back to the camera. This made it easy for the system to detect the cars.

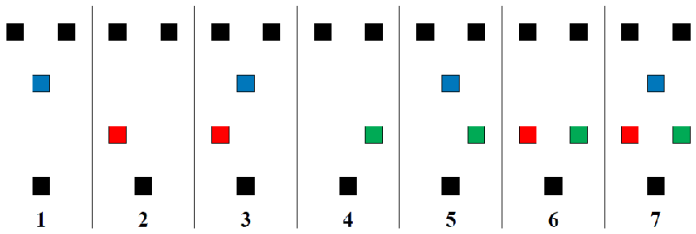


Figure 4.8: Retro-reflective marker patterns.

Although the technique is effective in detecting different cars, it was not feasible for this project because the camera needed to perform other functions. The infrared light filter would have limited the camera's capabilities.

[1] the specific numerical threshold mentioned (1.0 pixels) is based on practical guidelines and expert consensus observed in the field.

4.4.1 Detection with ArUco markers

After some time of research and investigation, another efficient form of detection was found: ArUco markers [4]. An ArUco marker is a specially designed square marker that stands out due to its prominent black border. Inside this border is a coded binary matrix that acts as its unique identifier. The black border helps computers quickly spot and recognise it in images, while the coded pattern enables identification and also enables the use of error detection and correction methods [4]. Figure 4.9 on page 40 illustrates some ArUco markers.

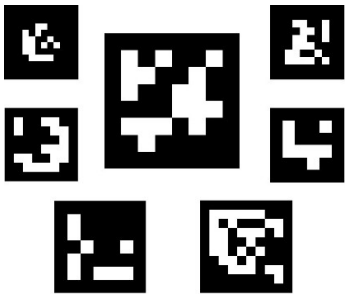


Figure 4.9: ArUco markers [4].

4.4.1.1 Marker creation

The function 'generate_ArUco' is used to create multiple markers. It begins by selecting a dictionary from the ArUco module. In this case, the 'DICT_6x6_250' dictionary was chosen, consisting of 250 markers with 6x6 bits. Therefore, the 'marker_id' ranges from 0 to 249. The function then generates an image of the ArUco marker using the selected dictionary.

```
1  newImagePrefix = "aruco_"
2  newImageExtension = ".png"
3  maxLeadingZeros = 4  # example: img_0000.png, img_0001.png, ...
4  imgCounter = 0
5  # Generate ArUco Marker
6  def generate_ArUco(marker_id, side_pixels):
7      # Define parameters for the ArUco marker
8      aruco_dict = cv2.aruco.getPredefinedDictionary(cv2.aruco.DICT_6X6_250)  #
           Define the dictionary for ArUco markers
9
10     # Generate the ArUco marker image
11     marker_image = cv2.aruco.generateImageMarker(aruco_dict, marker_id, side_pixels
           )
12
13     # Save the generated ArUco marker image
14     global imgCounter
15     img_name = f"{newImagePrefix}{str(imgCounter).zfill(maxLeadingZeros)}{
           newImageExtension}"
16     img_path = os.path.join(saveAruco, img_name)
17     cv2.imwrite(img_path, marker_image)
```

```
18      saved = cv2.imwrite(img_path, marker_image)
19      if saved:
20          print(f"Image saved successfully: {img_path}")
21      else:
22          print(f"Failed to save image: {img_path}")
23      imgCounter += 1
24
25  # Save to path generated arucos
26  saveAruco = os.path.join(os.getcwd(), "goProHero10\src\Camera\ArUco_img")
27  if not os.path.isdir(saveAruco):
28      try:
29          os.mkdir(saveAruco)
30      except Exception as e:
31          print(f"Error creating directory: {e}")
32
33  generate_ArUco(36, 400)
34  generate_ArUco(50,400)
35  generate_ArUco(80, 300)
```

Listing 4.1: Markers generator [18].

Figure 4.10 represents the function 'generate_ArUco' output:

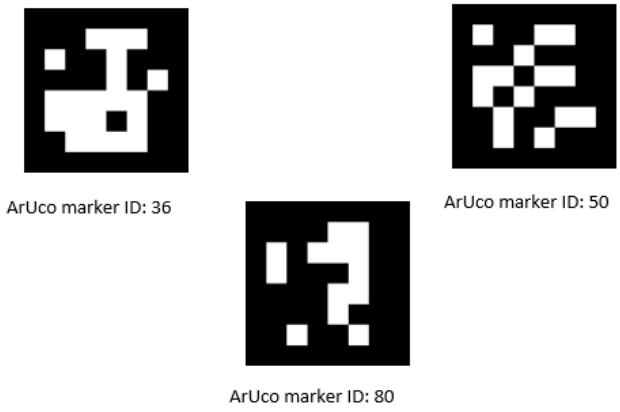


Figure 4.10: 3 markers generated by the function.

4.4.1.2 Marker detection

The markers produced in section 4.4.1.1 were printed to test marker detection. Figure 4.11 displays two printed markers and two similar symbols to verify the implemented function's ability to distinguish between them and only detect the aruco markers.

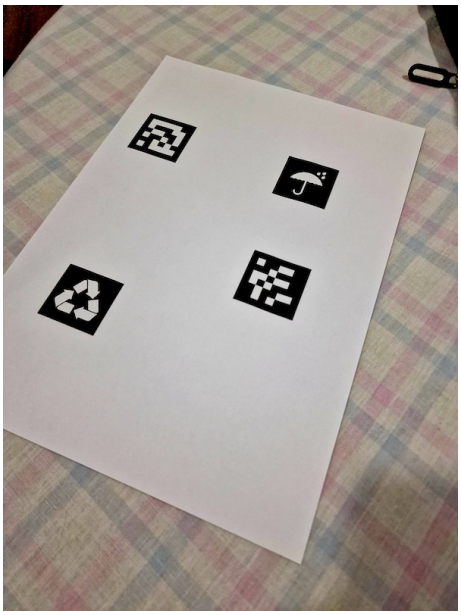


Figure 4.11: Printed ArUco markers.

The 'ArUco_detection' function detects ArUco markers in an input image using marker detection parameters and a dictionary. The function also identifies any potential markers that were not recognized or rejected during the detection process and stores them in 'rejected_candidates'. Finally, the function returns a tuple ('marker_corners', 'marker_ids', and 'rejected_candidates') that contains information about the detected markers.

```
1   def ArUco_detection(original_image_path):
2   # Load input image
3   original_image = cv2.imread(original_image_path)
4
5   # Define parameters for marker detection
6   parameters = cv2.aruco.DetectorParameters()
7   dictionary = cv2.aruco.getPredefinedDictionary(cv2.aruco.DICT_6X6_250)
8
9   # Detect markers in the input image
10  marker_corners, marker_ids, rejected_candidates = cv2.aruco.detectMarkers(
        original_image, dictionary, parameters=parameters)
11
12  return marker_corners, marker_ids, rejected_candidates
```

Listing 4.2: Markers detection.

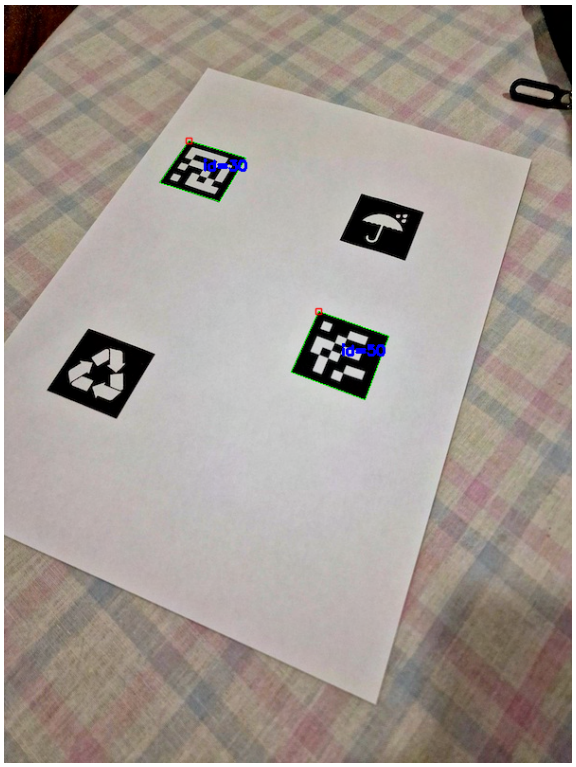


Figure 4.12: Image with detected markers.

Based on Figure 4.12, it can be concluded that the function successfully detected ArUco markers and rejected candidates.

4.4.1.3 Markers Pose Estimation

Pose estimation refers to determining the position and orientation of an object in space. To perform camera pose estimation, it is necessary to know the camera's calibration parameters. These comprise the camera matrix and distortion coefficients, as previously outlined in section 4.3. When estimating the pose with ArUco markers, it is possible to estimate the pose of each marker individually [4]. The function 'pose_estimation', available in the appendix B.3 and on the GitHub repository [18], is designed to identify each ArUco marker placed on the RC cars and determine their poses. The process begins by reading the camera matrix and distortion coefficients obtained from the camera calibration process.

The 'cv2.aruco.detectMarkers' function is then used to detect the markers in the camera frame, utilising a predefined dictionary. If markers are detected, the 'cv2.aruco.estimatePoseSingleMarkers' function estimates the pose of each marker and returns the rotation vectors ('rvec') and translation vectors ('tvec'). These vectors differ from the camera coefficients but are essential for determining the marker's pose. The detected markers and their axes are then drawn onto the image for visualisation.

Another function, 'marker_position', is also created to calculate the marker's position in the camera coordinate system and is available in the appendix B.4, and in the GitHub [18]. This

function uses the rotation and translation vectors to determine the marker's position. The rotation vector is first converted into a rotation matrix using 'cv2.Rodrigues'. The inverse of this rotation matrix is then computed to transform the marker's coordinates into the camera coordinate system. The translation vector is reshaped into a column vector, and the marker's position is calculated using the inverse transformation.

It was not possible to utilise the ArUco marker system for car detection in this instance, as the camera was unable to detect the markers on the racetrack reliably. Detection was only possible when the markers were lifted closer to the camera, as Figure 4.13 demonstrates:

(a) The markers were not detected when on the racetrack floor.

(b) The markers were detected when moved closer to the camera.

Figure 4.13: ArUcos not detected vs. ArUcos detected.

Despite the challenges encountered in this scenario, the advantages of utilising ArUco markers demonstrate their potential for many applications where reliable and precise detection and pose estimation are required. Here are some of the advantages of using these fiducial markers [11]:

- **Robust and Reliable detection**: ArUco markers exhibit high contrast and distinctive binary patterns that facilitate their detection and differentiation from one another, even in the presence of varying lighting conditions. This substantially reduces the probability of false positives, thereby ensuring accurate identification [4].
- **Pose Estimation Capabilities**: These markers facilitate the estimation of both the position and orientation (pose) of objects in three-dimensional space. The algorithms have been optimised for performance, allowing real-time processing on standard computing hardware.
- **Cost-Effectiveness**: ArUco markers can be printed on standard paper using regular printers, representing an inexpensive solution for a wide range of applications. Furthermore, detecting these markers does not necessitate using specialised or costly equipment; a standard camera is sufficient.

4.4.2 RC cars' colour detection

The initial attempt at detecting the RC cars on the racetrack using the first method, ArUco, proved unsuccessful, so it was decided to explore an alternative approach. Given the vibrant and distinct colours of the cars, shown in Figure 2.5a, it was determined that a colour-based detection system could be a viable solution. In order to investigate the use of colour-based detection algorithms that could analyse the unique hues of the individual vehicles and track their positions as they navigated the course, a series of experiments were conducted.

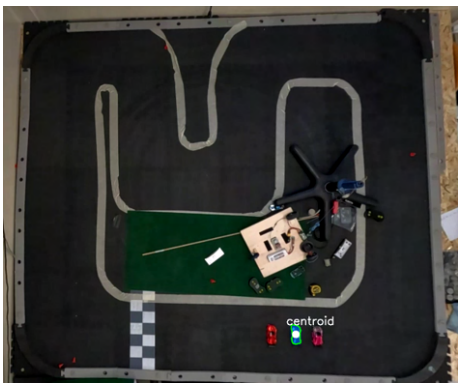
The colour-based detection approach involved the capture of video footage of the racetrack and subsequent processing of the images to identify the different car colours. This could be achieved through techniques such as colour segmentation, where the image is divided into regions based on colour similarities, or by using colour histograms to analyse the distribution of colours within the frame. By establishing a colour profile for each car, the detection algorithm could then track the movements of the vehicles as they raced around the course.

The technique to detect the colours involves colour segmentation in the HSV (Hue, Saturation, Value) colour space, and it is available in appendix B.5 and in GitHub [18].

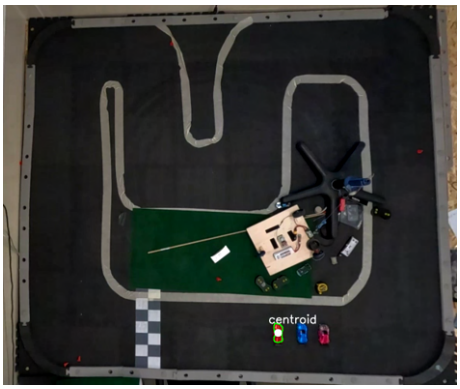
In the initial step of each colour detection function, 'detect_blue_cars', 'detect_yellow_cars', and 'detect_pink_cars', the input image is transformed from the BGR colour space to the HSV colour space using the OpenCV function 'cv2.cvtColor'. Each function specifies a range of HSV values corresponding to the target colour, defined as lower and upper bounds in the HSV space. Mask creation relies on these specified HSV ranges, utilising the 'cv2.inRange' function to generate binary masks. Pixels within the specified colour range are assigned a value of white (255), while all other pixels are assigned a value of black (0).

Subsequently, morphological operations, specifically closing and opening, are applied to refine the mask. This process removes noise and fills holes within the identified regions. The 'cv2.morphologyEx' function is employed to perform these operations. Contours are then identified in the cleaned-up mask using the 'cv2.findContours' function. These contours are filtered based on their area, retaining only those that fall within a specified range, such as the RC cars.

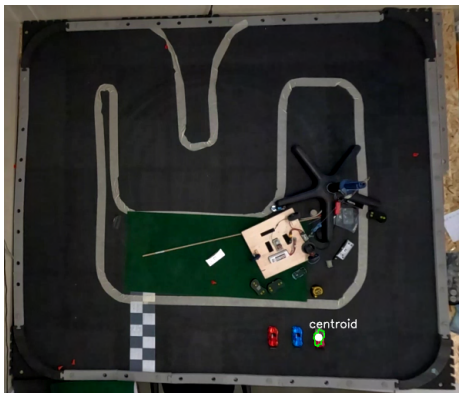
The filtered contours are then superimposed on the original image using the 'cv2.drawContours' function. The centroid of the largest contour is computed using image moments and subsequently drawn on the image. The centroids are returned in a list where missing centroids are represented as arrays of zeros.



(a) Blue car detected.



(b) Red car detected.

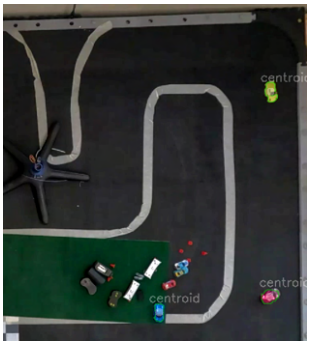


(c) Pink car detected.

Figure 4.14: Car detection results for different colours.

During the car detection in the live video, neither the pink nor the red car were detected very well. Since pink and red have similar HSV values, these values might overlap, causing the algorithm to miss one car when detecting the other. To resolve this issue, the red car was painted yellow. This colour change allowed for more accurate detection, as the HSV values for yellow are distinct from those for pink and red, enabling the algorithm to identify both vehicles successfully.

(a) Yellow car painted.



(b) Yellow car detection.

Figure 4.15: Yellow car for better detection.

4.5 Car's Position Estimation

This section discusses all the work done to estimate the car's position.

4.5.1 Pixel-to-millimetre Ratio

When cars are detected, their positions are updated in each live video frame. These positions refer to the centroids of each car contour, which are obtained in the 'detect_cars' function [18] and are returned in pixel coordinates. To match the position in the real world with the position in the simulation, it is necessary to update each car's position using metric system coordinates (centimetres). For this purpose, a script called 'pixel2mm.py' was designed and is also available in GitHub [18] and in the appendix B.6. The main goal of this script is to obtain the pixel-to-millimetre ratio from an image that includes an A4 paper.

Firstly, the script reads the image and converts it to greyscale. The next step is to convert the image to binary for better paper detection, followed by dilation and erosion processes to clean the image and remove noise. The function 'cv2.findContours' is then used to find all possible contours. Figure 4.16 shows all detected contours outlined in green.

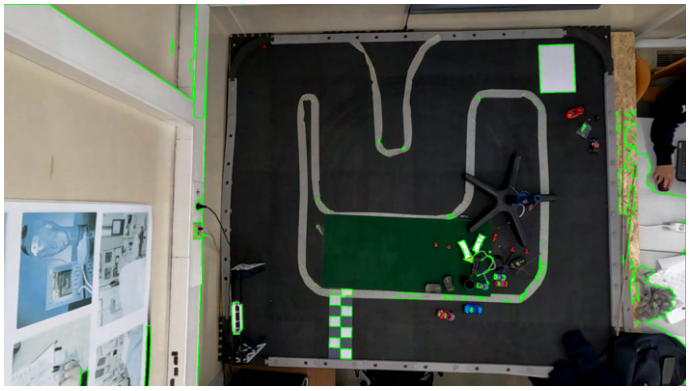


Figure 4.16: All contours resembling an A4 paper.

To filter out contours that do not resemble an A4 paper, a 'for' loop checks each contour for an aspect ratio similar to that of an A4 paper (210:297). Contours that meet this criterion are saved in a list called 'paper_contours'. Among the filtered contours, the largest one is identified to ensure the correct contour is selected (A4 paper).

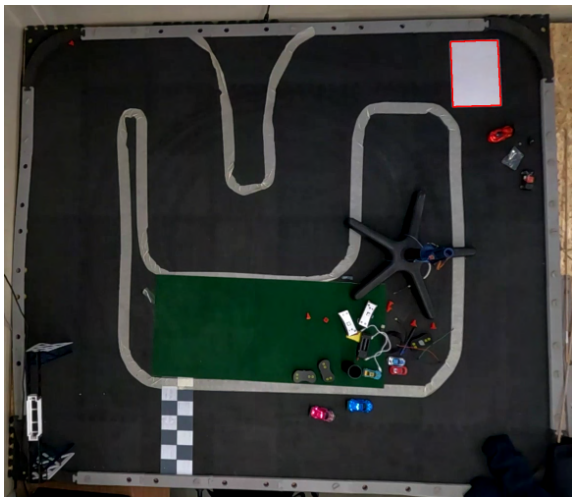


Figure 4.17: Largest Paper-like contour.

As figure 4.17 shows, the paper was outlined with red, meaning the right contour was selected. For further refining, the contour detected was approximated with a 4-sided polygon. Then, the coordinates of the paper vertices were defined as shown in figure 4.18 with small, red crosses.

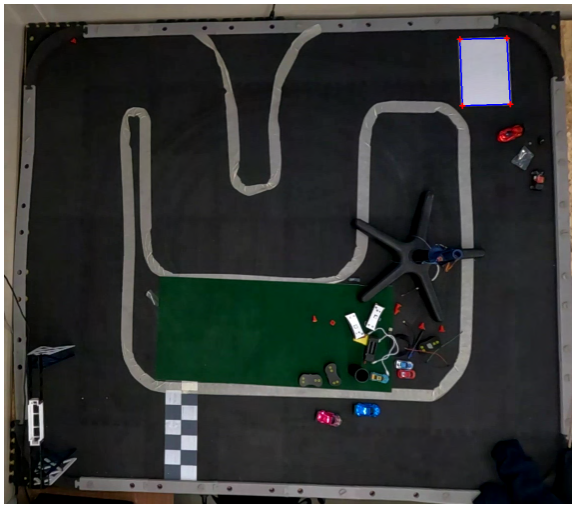


Figure 4.18: Approximated contour.

The process of approximating the contour with a 4-sided polygon involves simplifying the contour to a shape that closely resembles a rectangle. This is achieved using the Douglas-Peucker algorithm, implemented in OpenCV's 'cv2.approxPolyDP' function. The algorithm works by recursively subdividing the contour into smaller segments and removing points that are not essential to the overall shape, based on a specified precision parameter ε. This precision is set as a fraction of the contour's perimeter, calculated using the 'cv2.arcLength' function. If the approximated contour has exactly four vertices, it is considered a quadrilateral, which is necessary for further processing. The coordinates of these vertices are stored as $V_{topleft}$, $V_{topright}$, $V_{bottomright}$, and $V_{bottomleft}$.

Finally, after successfully detecting the A4 paper and calculating its vertices, the height distance was calculated as follows:

$$\text{dst\_height} = \max\left(\sqrt{(V_{\text{topleft,y}} - V_{\text{bottomleft,y}})^2}, \sqrt{(V_{\text{topright,y}} - V_{\text{bottomright,y}})^2}\right) \tag{4.9}$$

where $V_{topleft,y}$, $V_{bottomleft,y}$, $V_{topright,y}$, and $V_{bottomright,y}$ are the y coordinates of each paper's vertices.

The pixel-to-millimetre is obtained by using equation 4.10:

$$px2mm = \frac{dst\_height}{297} \tag{4.10}$$

'dst_height' calculated in equation 4.9, is the height of the A4 paper measured in pixels. It is the maximum vertical distance between the top and bottom vertices of the detected paper contour. The A4 paper is represented in a vertical orientation (portrait). This means that the shorter side (210 mm) is along the horizontal axis, and the longer side (297 mm) is along the vertical axis. So, when the paper has this orientation, its height will be 297.

For converting the pixel coordinate position of the cars to real-world coordinates, the pixel-to-millimetre ratio is applied:

$$centroids_{cm} = \frac{\frac{centroids_{px}}{px2mm}}{10} \tag{4.11}$$

where $centroids_{cm}$ is the position of the cars in centimetres, and $centroids_{px}$ is the position in pixels. The division by 10 is to get the position in centimetres instead of millimetres.

4.5.2 Reference Point - Indication of the Origin

One of the principal aims of this project is to achieve a high level of fidelity, thereby ensuring that the real-world implementation closely resembles the simulation environment developed by the Digital Twin. It was imperative to establish a reference point that serves as the origin, analogous to the one utilised in the virtual simulation. This reference point plays a pivotal role in maintaining spatial consistency and accuracy between the simulated and physical environments.

In the simulation, the reference point is situated in the bottom-right corner of the racetrack. In order to maintain consistency and facilitate a seamless transition between the simulated and real-world environments, it was determined that the physical reference point should be placed in the exact same location on the actual racetrack.

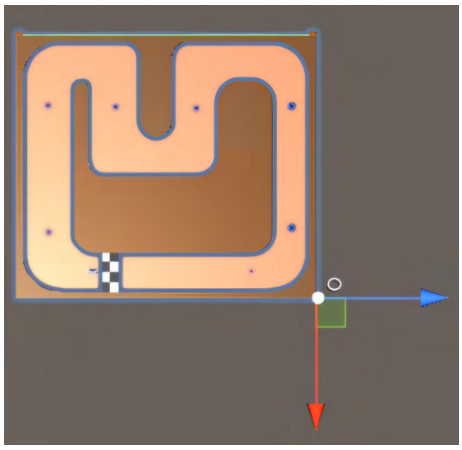


Figure 4.19: Reference point in the simulation environment.

As depicted in Figure 4.19, the reference point is situated in the bottom right corner of the simulated racetrack. This origin serves as the spatial anchor, with the position of the RC car being defined in relation to this fixed point.

A white circular marker was employed to implement the reference point in the physical environment to ensure clear and unambiguous identification. The white colour was selected to provide the greatest contrast against the dark colour of the racetrack. The circular shape was selected for its symmetry and ease of recognition from multiple angles, which is crucial for maintaining accuracy during measurements and alignments.

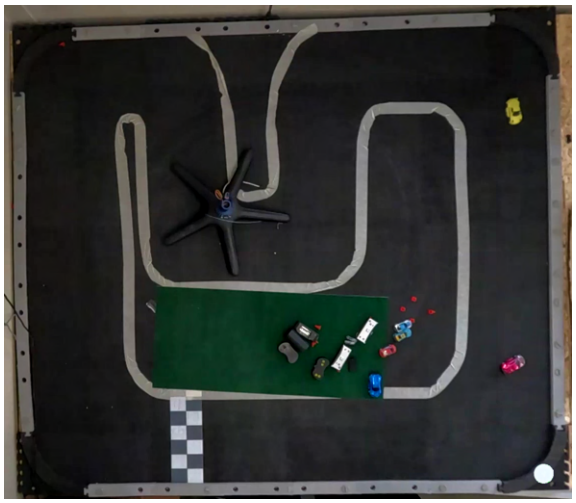


Figure 4.20: Reference point in the physical environment.

The ‘find_circle‘ function [2] was designed to detect circles in an image, similar to the example shown in Figure 4.20, using computer vision techniques provided by the OpenCV library. The

[2]This function is available in the GitHub repository [18] and is also available in the appendix B section B.7

process begins by converting the input image from colour to greyscale. This simplifies the image data and reduces computational complexity, as working with a single channel is easier than handling three (BGR) channels.

Next, an adaptive threshold is implemented to convert the greyscale image into a binary image. This helps distinguish the foreground (potential circles) from the background. The method 'cv2.adaptiveThreshold' adjusts the threshold dynamically for different image regions based on the local mean, which can be more effective for images with varying lighting conditions.

The binary image then undergoes morphological operations to remove small noise. Dilation expands the white regions, while erosion shrinks them, effectively closing small holes and gaps. This noise removal step prepares the image for the core circle detection step, which employs the Hough Circle Transform algorithm.

The Hough Circle Transform algorithm is a powerful computer vision technique used to detect circles in an image. The process works as follows:

- **Edge Detection**: The algorithm first applies an edge detector, such as the Canny edge detector, to identify the edges in the image.
- **Parameter Space**: For each edge point, the algorithm maps out all the possible circles that could pass through that point. This mapping is done in the Hough parameter space, which includes the centre coordinates (x, y) and the radius r.
- **Accumulation**: The algorithm uses an accumulator matrix to count how many times different potential circles pass through the edge points. Each edge point "votes" for the circles in the parameter space to which it could belong.
- **Peak Detection**: The peaks in the accumulator matrix represent the most likely circles, where many edge points align in a circular shape. These peaks are identified as the detected circles.

This process allows the algorithm to robustly detect circles even in noisy images.

Finally, the detected circles are drawn on the original image. The centre coordinates of each detected circle are extracted and stored in the 'centers' list. This step ensures that the detected circles are accurately represented on the input image, clearly representing the identified circular shapes. The function then returns the annotated image and the array of centre coordinates for the detected circles.

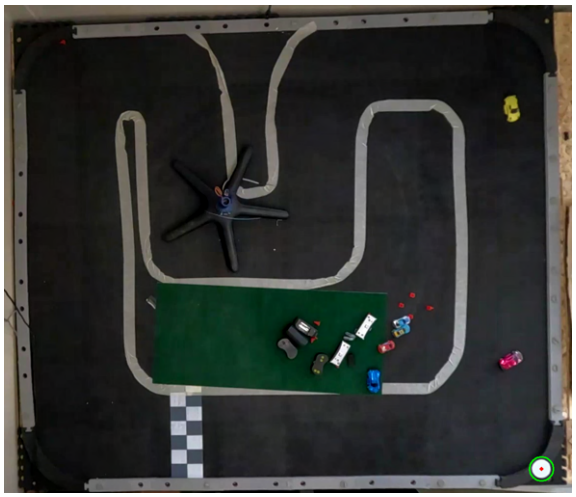


Figure 4.21: Image with the detected reference point.

Following the successful detection of the reference point, its position value is known thanks to the return variables of the ‘find_circle‘ function. In order to determine the position of each RC car in relation to the reference point, it is necessary to subtract the coordinates of the reference point from the coordinates of the current position of the RC car.

$$centroid_{car/origin} = centroid_{car} - origin\_coords \tag{4.12}$$

In order to determine the position of each RC car in relation to the reference point in centimetres, it is possible to make use of the previously established pixel-to-millimetre ratio, which was discussed in section 4.5.1.

$$centroid_{cm} = \frac{\frac{centroid_{car/origin}}{px2mm}}{10} \tag{4.13}$$

By applying this conversion factor, the relative position of the car, expressed in pixel coordinates, can be translated into a more meaningful and intuitive metric of centimetres. This enables the accurate quantification of the spatial relationship between the reference point and the location of the RC car, thereby facilitating precise positioning and control of the vehicles within the monitored environment.

4.5.3 Results

To assess the accuracy of the camera position readings, several physical measurements of the RC car concerning the reference point were taken. Table 4.1 compares these real-world measurements with the corresponding values calculated by the camera system. Additionally, Table 4.2 presents various error metrics and accuracy measures to quantify the discrepancies between the camera and physical measurements.

Table 4.1: Comparison between physical and camera measurements (cm) of the RC car in the racetrack.

	Physical Measurements		Camera Measurements	
Position	x (cm)	y (cm)	x (cm)	y (cm)
1	85	12	88	14
2	18	87	19	89
3	149	110	151	113
4	18	1	18	1
5	150	17	152	19
6	204	58	206	60
7	114	48	116	50

Table 4.2: Error and Accuracy Metrics.

Metric	x (cm) / %	y (cm) / %
Mean Absolute Error (MAE)	1.71	1.86
Root Mean Square Error (RMSE)	1.93	2.04
Mean Absolute Percentage Error (MAPE)	2.07%	5.87%
Overall Euclidean Distance Error (cm)	2.56	
Correlation Coefficient (R^2)	0.99	0.99

The Mean Absolute Error (MAE) indicates that, on average, the camera's x-coordinate measurements deviate from the physical measurements by approximately 1.71 cm, while the y-coordinates measurements differ by around 1.86 cm.

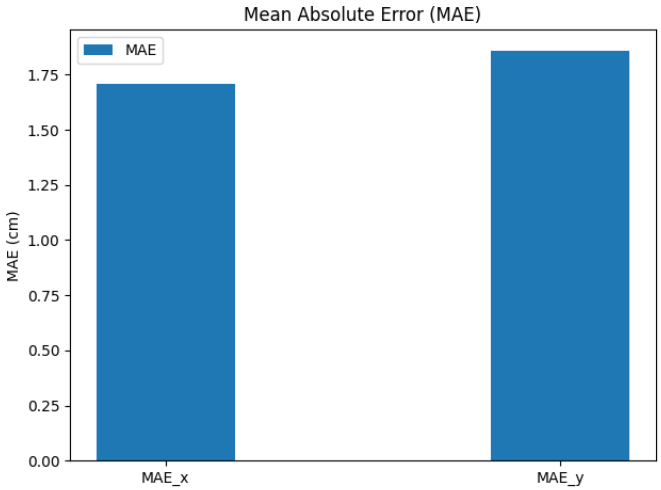


Figure 4.22: Mean Absolute Error.

The Root Mean Squared Error (RMSE) values suggest that the typical deviation of the camera measurements from the physical measurements is 1.93 cm for the x-coordinates and 2.04 cm for the y-coordinates. The slightly higher RMSE values compared to the MAE imply that larger errors may affect the measurements.

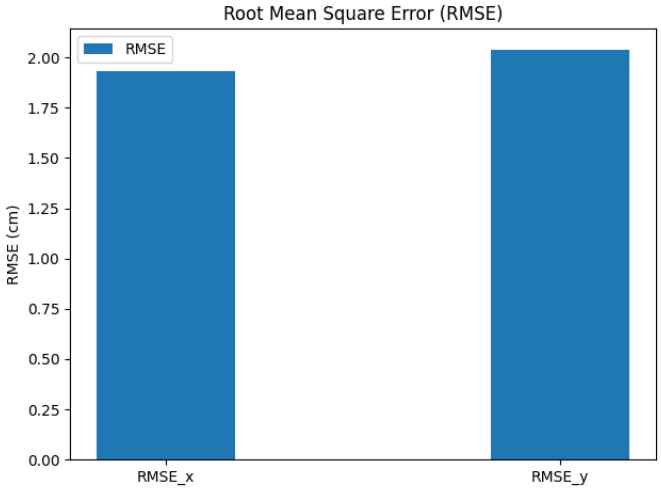


Figure 4.23: Root Mean Squared Error.

Furthermore, the Mean Absolute Percentage Error (MAPE) reveals a relatively low average percentage error of 2.07% for the x-coordinate measurements, indicating good accuracy in that direction. However, the y-coordinate measurements have a slightly higher average percentage error of 5.87%, suggesting less accuracy in the y-direction.

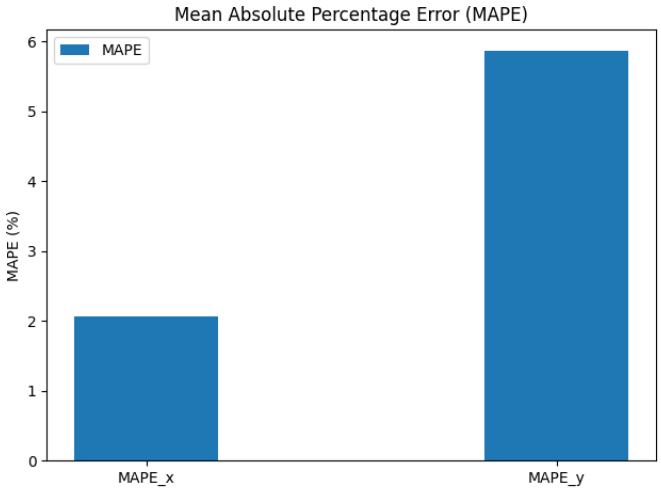


Figure 4.24: Mean Absolute Percentage Error.

The Overall Euclidean Distance Error provides a single measure of the average positional error in the racetrack, which is 2.56 cm. This metric combines the errors in both the x and y coordinates.

Notably, the R^2 values for both the x and y coordinates are close to 1, indicating an excellent correlation between the camera and physical measurements. This suggests that the camera measurements are highly consistent with the physical measurements despite the observed errors.

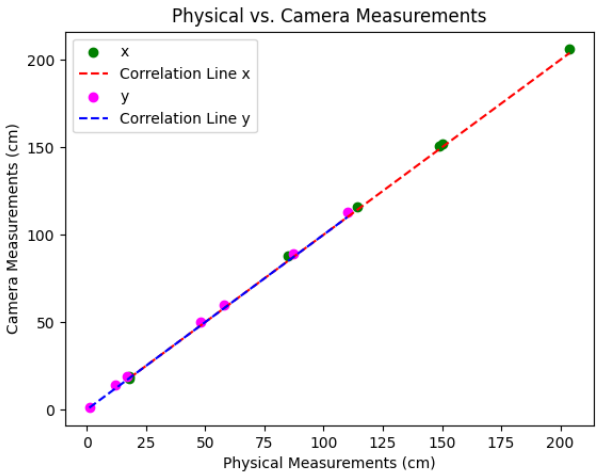


Figure 4.25: Physical vs Camera measurements.

Overall, the results demonstrate that the camera system is quite accurate, particularly in the x-direction. However, there is room for improvement in the y-coordinate measurements. The consistently high R^2 values are a positive sign, showing that the camera measurements are well-aligned with the physical reality.

4.5.4 Extended Kalman Filter - EKF

In the course of selecting an appropriate algorithm for vehicle localisation and state estimation, a number of potential methods were considered. Each algorithm possesses distinctive strengths and weaknesses, rendering them well-suited to specific aspects of autonomous vehicle navigation.

One of the principal algorithms subjected to evaluation was the Particle Filter. This method is renowned for its resilience in addressing nonlinear systems and non-Gaussian noise. The Particle Filter employs a set of random samples, or particles, to represent the posterior distribution of the system state. Its strength lies in its ability to manage highly nonlinear dynamics and measurement models, which are common in real-world scenarios. However, the Particle Filter can be computationally intensive, especially as the number of particles required for accurate estimation increases. This makes it less suitable for real-time applications where computational resources are limited [17].

Another algorithm that was considered was the Unscented Kalman Filter (UKF). The UKF addresses the limitations of the standard Kalman Filter by employing a deterministic sampling technique, the Unscented Transform. This approach provides a more accurate representation of the mean and covariance of the system state distribution, particularly in highly nonlinear systems. The UKF is generally more accurate than the Extended Kalman Filter (EKF) in many scenarios

due to its superior handling of nonlinearity. However, it can also be more complex to implement and may require more computational resources than the EKF [10].

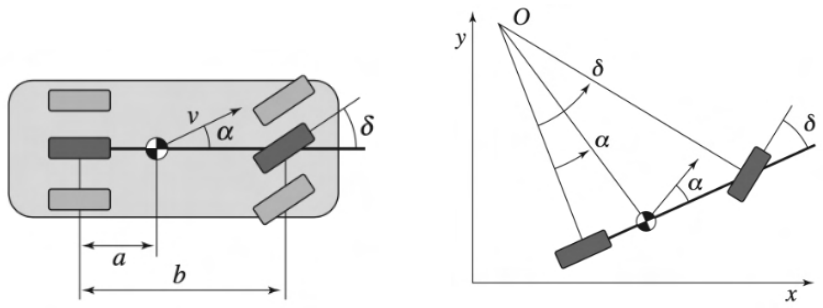
After a thorough evaluation, the Extended Kalman Filter (EKF) was chosen as the most suitable algorithm for this dissertation project. The Extended Kalman Filter (EKF) is a sophisticated algorithm that serves as a nonlinear extension of the traditional Kalman Filter. It was selected because it offers a good balance between accuracy and computational efficiency, making it well-suited for real-time autonomous vehicle navigation. The EKF linearises the nonlinear models at each time step, providing a more accurate state estimation than the standard Kalman Filter while being less computationally demanding than the Particle Filter and the UKF [22]. This powerful tool has found widespread applications in various fields, including robotics, navigation systems, and autonomous vehicles [27]. In this project, the EKF is implemented to estimate the position and orientation of a vehicle using a kinematic bicycle model, showcasing its effectiveness in real-world applications.

The EKF operates by linearising the nonlinear functions around the current state estimate, allowing it to handle complex systems more accurately than its linear counterpart. This linearisation process involves computing Jacobian matrices of the system dynamics and measurement models, which are then used to update the state estimate and covariance matrix.

In this specific implementation, the vehicle's state vector typically includes its position (x, y coordinates), orientation (heading angle), and velocity. The EKF recursively estimates these state variables based on two primary inputs: the vehicle's velocity and steering angle measurements. The velocity is measured using the camera, and the steering angle is computed using the custom PCB, previously mentioned in section 2.4.3.

4.5.4.1 Kinematic Bicycle Model

Before implementing the EKF, it is crucial to describe the vehicle's kinematic model. It represents the vehicle's motion, allowing the filter to predict the car's future states and update its estimated position and orientation based on sensor measurements.



(a) Overhead view. (b) Bicycle model.

Figure 4.26: Vehicle steering dynamics [30].

Figure 4.26 describes the vehicle steering dynamics. The figure 4.26a displays an overhead view of a four-wheeled vehicle with a wheelbase of b and a centre of mass at a distance a forward of the rear wheels. To facilitate the analysis of the motion of the front and rear pairs of wheels, an abstraction known as the bicycle model is employed, as illustrated in 4.26b. This model offers a simplified yet effective representation of a vehicle's motion, striking a balance between computational efficiency and accuracy. This model, which is widely used in automotive control systems and robotics, assumes that the vehicle consists of a steerable front wheel and a driving rear axle. This effectively reduces the complexity of a four-wheeled vehicle to that of a two-wheeled bicycle-like structure [30].

The vehicle's motion in this model is governed by a set of differential equations that describe its state evolution over time:

$$\dot{x} = v\cos(\alpha+\theta) \tag{4.14}$$

$$\dot{y} = v\sin(\alpha+\theta) \tag{4.15}$$

$$\dot{\theta} = \frac{v}{b}\tan\delta \tag{4.16}$$

where x and y represent the vehicle's position in a 2D plane, θ is the vehicle's heading angle, v is the vehicle's velocity, δ is the steering angle of the front wheel and α represents the slip angle which accounts for the difference between the steering angle and the vehicle's actual direction of travel. It is calculated as:

$$\alpha = \arctan(\frac{a\tan\delta}{b}) \tag{4.17}$$

Here, α modifies the orientation θ to account for the actual direction of travel, considering the geometry of the RC car and steering angle δ.

4.5.4.2 EKF Implementation

The EKF implementation involves several steps, as shown in the following code extracts. The code is also available in the GitHub repository [18].

```
class EKF:
    def __init__(self, dt, a, b, Q, R, initial_state, initial_covariance):
        self.dt = dt
        self.a = a
        self.b = b
        self.Q = Q
        self.R = R
        self.X = initial_state
        self.P = initial_covariance
```

Listing 4.3: EKF class initialisation.

The process begins by initialising the EKF parameters, including the time step 'dt', the wheelbase 'b', the distance 'a' between the rear axle and the centre of mass, the process noise covariance 'Q', the measurement noise covariance 'R', the initial state 'X', and the initial covariance 'P'.

```
    def wrap_to_pi(self, angle):
        return (angle + np.pi) % (2 * np.pi) - np.pi
```

Listing 4.4: Normalise the angles into a specified range.

The function 'wrap_to_pi' ensures that angles are normalised to the $[-\pi, \pi]$ range.

Prediction Step

```
    def predict(self, v, delta):
        theta = self.X[2]
        alpha = np.arctan((self.a * np.tan(delta)) / self.b)

        self.X[0] += v * np.cos(alpha + theta) * self.dt
        self.X[1] += v * np.sin(alpha + theta) * self.dt
        self.X[2] += (v / self.b) * np.tan(delta) * self.dt
        self.X[2] = self.wrap_to_pi(self.X[2])

        Fx = np.array([
            [1, 0, -v * np.sin(alpha + theta) * self.dt],
            [0, 1,  v * np.cos(alpha + theta) * self.dt],
            [0, 0,  1]
        ])

        Fw = np.array([
            [np.cos(alpha + theta) * self.dt, 0],
            [np.sin(alpha + theta) * self.dt, 0],
            [0, self.dt]
        ])

        self.P = Fx @ self.P @ Fx.T + Fw @ self.Q @ Fw.T
```

Listing 4.5: Prediction Step.

Based on the kinematic bicycle model equations, the 'predict' method updates the state vector 'X':

$$X = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{4.18}$$

The state transition function $f(X,u)$ describes how the state evolves over time:

$$X_{k+1} = X_k + f(X_k, u_k) \cdot dt \tag{4.19}$$

where u_k represents the control inputs - velocity and steering angle:

$$u_k = \begin{bmatrix} v \\ \delta \end{bmatrix} \tag{4.20}$$

It calculates the new position (x,y) and the orientation θ. For a discrete time step 'dt', the state update equations are:

$$x_{k+1} = x_k + v_k \cos(\alpha_k + \theta_k) \cdot dt \tag{4.21}$$

$$y_{k+1} = y_k + v_k \sin(\alpha_k + \theta_k) \cdot dt \tag{4.22}$$

$$\theta_{k+1} = \theta_k + \frac{v_k}{b} \tan(\delta_k) \cdot dt \tag{4.23}$$

The Jacobian matrices F_x and F_w linearise the nonlinear state transition model around the current estimate. This linearisation enables the Kalman filter equations, which are designed for linear systems, to be employed. The Jacobian matrix F_x represents the state transition function with respect to the state variables $[x,y,\theta]$. It illustrates how minor alterations in the state variables affect the subsequent state. Furthermore, it propagates the state covariance matrix forward in time.

$$F_x = \frac{\partial f}{\partial x} = \begin{bmatrix} 1 & 0 & -v\sin(\alpha+\theta)dt \\ 0 & 1 & v\cos(\alpha+\theta)dt \\ 0 & 0 & 1 \end{bmatrix} \tag{4.24}$$

On the other hand, the Jacobian matrix F_w represents the state transition function with respect to the process noise. It shows how the process noise affects the state transition and is used to account for the uncertainty in the process model. The original process noise Jacobian matrix F_w is derived from the partial derivatives of the state transition equations with respect to the control inputs $[v,\delta]$:

$$F_w = \frac{\partial f}{\partial u} = \begin{bmatrix} \cos(\alpha+\theta)dt & -v\sin(\alpha+\theta)(\frac{a\cdot dt}{b\cos^2\delta}) \\ \sin(\alpha+\theta)dt & v\cos(\alpha+\theta)(\frac{a\cdot dt}{b\cos^2\delta}) \\ 0 & \frac{v\cdot dt}{b\cos^2\delta} \end{bmatrix} \tag{4.25}$$

The equation 4.25 can be simplified due to the limited steering angles that the RC car can produce and the necessity for high-frequency updates. The terms involving $-v\sin(\alpha+\theta)(\frac{a\cdot dt}{b\cos^2\delta})$ and $v\cos(\alpha+\theta)(\frac{a\cdot dt}{b\cos^2\delta})$ are higher-order terms that may have a minor impact on the overall system dynamics. The matrix can be simplified by omitting these terms, thereby reducing the computa-

tional complexity without significantly affecting the filter's accuracy.

$$F_w = \begin{bmatrix} \cos(\alpha+\theta)dt & 0 \\ \sin(\alpha+\theta)dt & 0 \\ 0 & dt \end{bmatrix} \tag{4.26}$$

These two Jacobian matrices are used to propagate the covariance matrix 'P', which represents the uncertainty in the state estimate:

$$P_{k+1} = F_x P_k F_x^T + F_w Q F_w^T \tag{4.27}$$

where Q is the process noise covariance matrix and is defined as:

$$Q = \begin{bmatrix} \sigma_v^2 & 0 \\ 0 & \sigma_\delta^2 \end{bmatrix} \tag{4.28}$$

The process noise covariance has two parameters: σ_v^2 is the variance of the velocity noise; σ_δ^2 is the variance of the steering noise.

- **Velocity noise** (σ_v^2): This value captures the uncertainty in the car's velocity. Factors influencing this could include wheel slip and variable surface traction.
- **Steering angle noise** (σ_δ^2): This value captures the uncertainty in the car's steering angle. Factors influencing this could include steering mechanism backlash and alignment issues.

Update Step

```
    def update(self, z):
        x, y, theta = self.X

        h = np.array([
            np.sqrt(x**2 + y**2),
            self.wrap_to_pi(np.arctan2(y, x - self.b) - theta)
        ])

        H = np.array([
            [x / np.sqrt(x**2 + y**2), y / np.sqrt(x**2 + y**2), 0],
            [-y / (x**2 + y**2), x / (x**2 + y**2), -1]
        ])

        S = H @ self.P @ H.T + self.R
        K = self.P @ H.T @ np.linalg.inv(S)

        y = z - h
        y[1] = self.wrap_to_pi(y[1])

```

```
20          self.X = self.X + K @ y
21          self.P = (np.eye(len(self.X)) - K @ H) @ self.P
```

Listing 4.6: Update Step.

The update step in the Extended Kalman Filter (EKF) is crucial for refining the state estimate by incorporating new measurements from the camera. This step adjusts the predicted state estimate based on the sensor readings, reducing the overall uncertainty in the state estimation.

First, the state vector $\mathbf{X}$ is unpacked into its individual components: the position (x, y) and the orientation θ. The measurement prediction $\mathbf{h}$ is then calculated using the current state estimates. Given that the measurement model is direct (the variables $\mathbf{x}$, $\mathbf{y}$, and θ are measured directly), $\mathbf{h}$ is simply the state vector:

$$\mathbf{h} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{4.29}$$

Next, the Jacobian of the measurement function, $\mathbf{H_x}$, is the 3x3 identity function in this case. This is because the measurement function directly observes the state components without any transformation:

$$\mathbf{H_x} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.30}$$

The innovation or residual measurement $\mathbf{y}$ is then calculated as the difference between the actual sensor measurement $\mathbf{z}$ and the predicted measurement $\mathbf{h}$:

$$\mathbf{y} = \mathbf{z} - \mathbf{h} \tag{4.31}$$

The innovation $\mathbf{y}$ represents the discrepancy between the predicted state and the actual measurement and is used to update the state estimate.

Calculating the Kalman Gain $\mathbf{K}$ involves two steps:

- Computing the innovation covariance $\mathbf{S}$ by transforming the state covariance $\mathbf{P}$ into the measurement space and adding the measurement noise covariance $\mathbf{R}$.
- Computing the Kalman Gain $\mathbf{K}$ using the state covariance $\mathbf{P}$ and the inverse of the innovation covariance $\mathbf{S}$.

$$\mathbf{S} = \mathbf{H_x} \mathbf{P} \mathbf{H_x}^\top + \mathbf{R} \tag{4.32}$$

$$\mathbf{K} = \mathbf{P} \mathbf{H_x}^\top \mathbf{S}^{-1} \tag{4.33}$$

The measurement noise covariance matrix, denoted by **R**, represents the uncertainty in the measurements. It influences the degree to which the measurements are trusted relative to the predictions. It is defined as:

$$\mathbf{R} = \begin{bmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_\theta^2 \end{bmatrix} \tag{4.34}$$

In the event that the noise is assumed to be uncorrelated, the matrix is diagonal. This assumption can be made on the grounds that the camera processes each measurement (position and orientation) independently. High-resolution cameras, such as the GoPro Hero 10 Black, are designed to minimise cross-talk between different measurements. The noise in the x and y coordinates is typically attributed to pixel-level inaccuracies, whereas the noise in the orientation θ is presumed to originate from the positional measurements, assuming a simplified model that could approximate them as uncorrelated. Assuming uncorrelated noise also simplifies the implementation and computation. The variance of the noise in the x-coordinate measurement is designated as σ_x^2. The variance of the noise in the y-coordinate measurement is designated as σ_y^2. The variance of the noise in the orientation measurement, denoted by σ_θ^2, is defined as the standard deviation of the error in the measurement of the angle θ.

Finally, the state vector **X** is updated using the Kalman gain **K** and the innovation **y**:

$$\mathbf{X} = \mathbf{X} + \mathbf{K}\mathbf{y} \tag{4.35}$$

and the covariance matrix **P** is also updated using the Kalman gain **K** and the Jacobian measurement $\mathbf{H_x}$:

$$\mathbf{P} = (\mathbf{I} - \mathbf{K}\mathbf{H_x})\mathbf{P} \tag{4.36}$$

where **I** is the identity matrix.

This update step allows the EKF to incorporate the new sensor measurements and refine the state estimate, reducing the overall uncertainty in the system's state. The following Figure 4.27 depicts the entire operation process of the EKF:

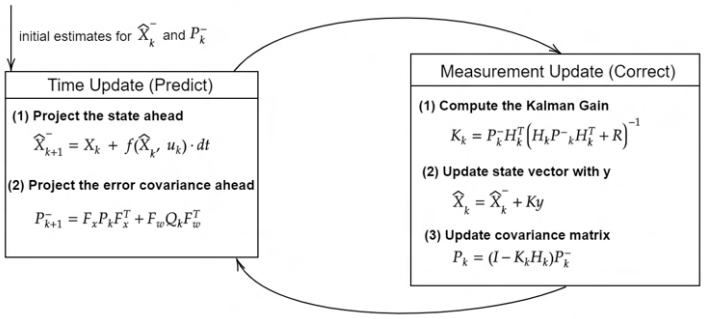


Figure 4.27: A complete picture of the operation of the EKF.

4.5.5 Results of EKF

Due to time limitations, the EKF could only be assessed in a simulated setting. Nevertheless, this simulation closely reflected the actual conditions, thereby ensuring a robust testing process and highly reliable results.

In the absence of a camera in the simulated environment, it is necessary to simulate the sensor measurements and feed them to the EKF. The same bicycle kinematic model is implemented in Unity for the purpose of replicating the vehicle's motion. The same kinematic equations, previously described in section 4.5.4.1, are employed in the simulation during the prediction phase to update the car's state (position, velocity, and orientation) based on control inputs (velocity and steering angle). The camera measurements are simulated by adding Gaussian noise to the true positions and orientations of the car in the simulation. If the true position is represented by the vector (x, y, θ), the noisy measurements could be generated as follows:

$$\tilde{x} = x + n_x \tag{4.37}$$

$$\tilde{y} = y + n_y \tag{4.38}$$

$$\tilde{\theta} = \theta + n_\theta \tag{4.39}$$

$$z = \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{\theta} \end{bmatrix} \tag{4.40}$$

where n is the noise vector (n_x, n_y, n_θ), and z is the vector representing the noisy measurements.

For a more realistic simulation of the real-world environment, the errors presented in table 4.2 are used to calculate the noise in the simulation. Firstly the noise in the x, y and θ measurements is based on a Gaussian distribution:

$$n_x \sim \mathcal{N}(0, \sigma_x^2) \tag{4.41}$$

$$n_y \sim \mathcal{N}(0, \sigma_y^2) \tag{4.42}$$

$$n_\theta \sim \mathcal{N}(0, \sigma_\theta^2) \tag{4.43}$$

Subsequently, the n_x and n_y are scaled based on the percentage error MAPE:

$$n_x := n_x \times (1 + MAPE_x) \tag{4.44}$$

$$n_y := n_y \times (1 + MAPE_y) \tag{4.45}$$

Finally, the Euclidean distance of the noise is checked and adjusted to ensure it does not exceed the overall error metric:

$$EuclideanError = \sqrt{n_x^2 + n_y^2} \tag{4.46}$$

If $EuclideanError > 2.56cm$, the noise vector is scaled to ensure it stays within this bound:

$$n := n \times \frac{2.56}{EuclideanError} \tag{4.47}$$

This method ensures that the synthetic noise closely matches the characteristics of the real-world measurement errors, leading to more realistic simulations. At each time step, the noisy measurements are calculated and then fed into the EKF update step. Ultimately, both the true state (derived from the simulation) and the estimated state (derived from the EKF) are logged, allowing for a comparison and evaluation of the EKF's performance.

The following Figure 4.28 compares the results of the car's trajectory when it is being simulated versus when the EKF is implemented with synthetic noise added to the system. The black line represents the car's trajectory without any noise, while the green line shows the trajectory estimated by the EKF.

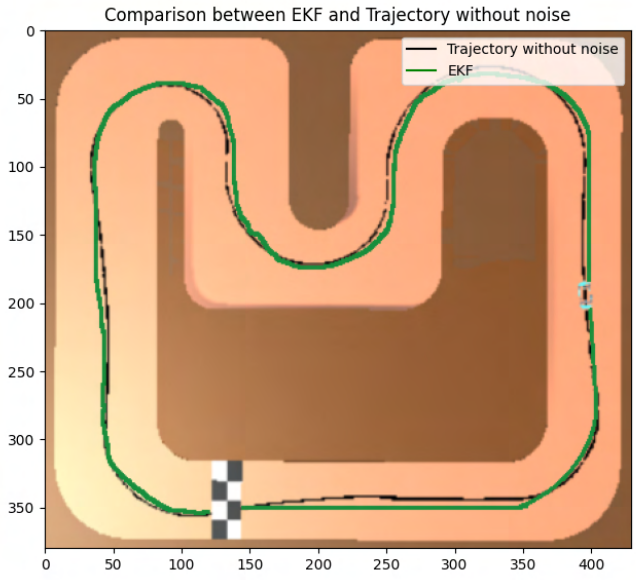


Figure 4.28: Results of the EKF.

After carefully analysing this graph, it can be concluded that the EKF successfully estimates the car's state despite the presence of noise. The green trajectory closely follows the black trajectory, indicating that the EKF can effectively filter out the noise and provide an accurate estimate of the car's position and orientation over time. This is a crucial capability for autonomous vehicles and other systems that must operate reliably in the presence of sensor noise and other uncertainties.

Overall, the results presented in this figure highlight the importance of using advanced filtering techniques like the EKF to handle the real-world challenges autonomous systems face. By effectively mitigating the effects of noise and uncertainties, the EKF enables the car to maintain a reliable and accurate understanding of its state, which is essential for safe and efficient operation.

4.5.6 Trajectory

The Python script, entitled 'Trajectory.py', which is accessible via the GitHub repository [18], defines a trajectory class that has been designed to facilitate the management and analysis of the path traversed by the RC car on the racetrack. The racetrack is divided into eight checkpoints, which serve to demarcate various segments such as straights and curves. These checkpoints, implemented in the real-world environment, are analogous to those designed in the simulation environment shown previously in Figure 3.5a.

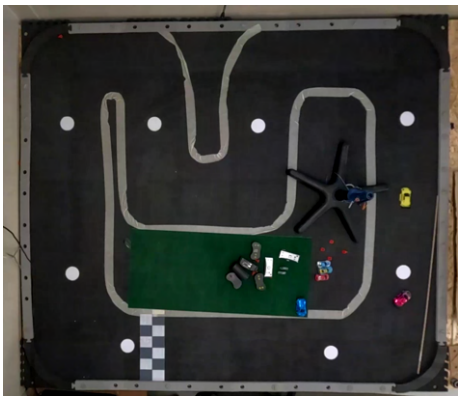


Figure 4.29: 8 circular checkpoints.

The detection method for these checkpoints is similar to the reference point mentioned in Section 4.5.2. They have the same circular shape as the reference point, allowing the same function to be used for their detection. However, while the origin marker is always present on the racetrack, the checkpoints were detected once and then saved in a list with a specific order.

When the camera is turned on or off, it may shift slightly due to the power button being pressed. This could introduce an issue, as the checkpoints were saved with a different camera position. To address this problem, an image containing all the checkpoints and the reference point was used as a reference. When the live video starts, it only detects the reference point and its new position. This position is then subtracted from the position of the reference image to establish a translation vector. The new coordinates of the checkpoints are simply the coordinates of the checkpoints in the reference image plus the translation vector.

$$t = \begin{bmatrix} x \\ y \end{bmatrix} \tag{4.48}$$

$$t = origin_{cur} - origin_{ref} \tag{4.49}$$

$$CheckPoints_{cur} = CheckPoints_{ref} + t \tag{4.50}$$

This 'Trajectory' class encompasses a range of methods, with its primary functionalities encompassing distance calculations, determination of the car's position relative to the track, and management of diverse curve types within the track.

4.5.6.1 Curves

The method 'shortest_point_to_curve' finds the closest point on a curve to the car's estimated position. In this specific racetrack, the defined checkpoints divide the segment into curves or straights. In the case of curves, they can be -90º, 90º or 180º types of curves in degrees. Depending on the type of curve, the method calculates the closest point differently:

- **-90º curves**: For a -90° curve, the code first defines two points that extend or offset from the actual curve endpoints. These points help shape the curve's turn in the virtual environment. With these offsets in place, the code estimates a centre point for this curve. Next, it checks the car's angle relative to the start and end of the curve to see if it falls within the -90° arc. If the car is within this arc, the closest point on the curve is calculated as a point on the circular path at the same distance (radius) from the centre. This closest point is then determined along the direction of the vector between the car's position and the curve centre.

- **90º curves**: The process for the 90° curve is similar to the -90° curve. The code uses two extended points that define the curve's arc, which helps set the orientation and shape of the 90° turn. It calculates the centre point of this arc and determines the radius or distance from this centre to the curve. The code then checks if the car's position falls within the 90° arc's range of angles. If it does, the closest point on the curve is found as the point along the circular path that lies on the same radius as the car's position, effectively projecting the car's position onto the curve's path.

- **180º curves**: For the 180° curve, the curve shape is set to form a half-circle, so the code needs to place the centre point between the two endpoints of the curve. Extended points also ensure the correct orientation and distance between the endpoints. Once the centre and radius are established, the code checks whether the car's position lies within the arc of this 180° curve. If it does, the closest point is calculated by projecting the car's position onto the circular path around the centre. This projection ensures that the closest point aligns with the curve's radius and is the shortest path to the car's position along the curve.

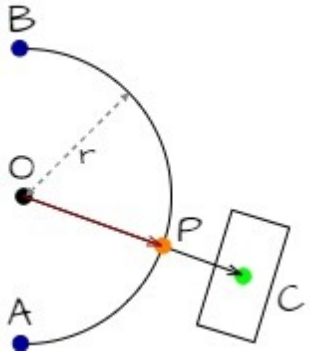


Figure 4.30: Closest point on a curve.

4.5.6.2 Straights

The 'shortest_point_to_line' method identifies the point on a straight segment that is closest to the car's current position. The method calculates the projection of the car's position onto the line segment. This projection is constrained to ensure that it remains within the bounds of the segment.

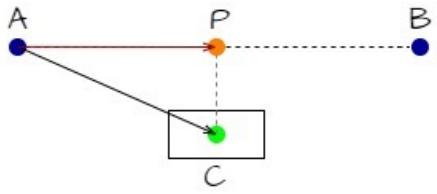


Figure 4.31: Closest point on a straight.

4.5.6.3 Distance to Trajectory

The 'get_distance_to_trajectory' method calculates the distance between the car's current position and the point on the trajectory that is the closest to it. This method identifies the point on the current segment, which may be either a straight or a curved line, that is closest to the car's position. In the event that the segment is a curve, the 'shortest_point_to_curve' method, previously described, will be employed. In the event that the segment is a straight line, the 'shortest_point_to_line' method will be employed. Furthermore, in order to determine whether the vehicle is situated on one side of the straight line or the other, or whether it is within or beyond the curve, the 'get_sign_of_distance_to_trajectory' method was designed to return a sign that will tell the car in which direction it must turn to reach the intended trajectory.

The aforementioned sign, in conjunction with the deviation between the vehicle's position and the closest point to the line/curve, constitutes the cross-track error (CTE). This error represents the perpendicular distance from the vehicle's current position to the closest point on the planned trajectory. Figure 4.32 shows a visual representation of the CTE.

$$CTE = get\_distance\_to\_trajectory() \cdot get\_sign\_of\_distance\_to\_trajectory() \tag{4.51}$$

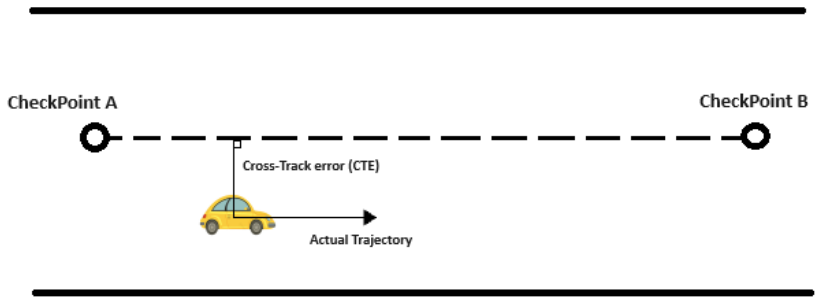


Figure 4.32: Cross-Track error example.

The CTE is then conveyed to the control unit, which utilises it in the PID to adjust the steering.

4.6 Source Code

The GitHub repository [18] contains source code that enables communication with the GoPro Hero 10 Black camera through HTTP requests and provides functionality to query and control various camera parameters. Furthermore, it includes a class that uses OpenCV for live previewing of the camera feed and allows for manipulating visualisation parameters.

4.6.1 GoPro Hero 10 Webcam Control Class 'goProHero10.py'

This class was based on the OpenGoPro HTTP API (2.0) [7], which allows developers to create apps and utilities that interact with and control a GoPro camera.

Features of this class:

- **HTTP Communication**: Implements robust HTTP communication with the GoPro Hero 10 Black camera, ensuring seamless interaction and control over the camera.
- **Camera Value Retrieval**: Allows users to fetch current settings and statuses of various camera parameters such as resolution, frame rate, battery status, and FOV.
- **Parameter Adjustment**: Enables real-time adjustment of camera parameters, including resolution, frame rate and FOV.
- **Media Download**: Supports downloading of media files (photos) directly from the camera to the connected device.

The GoPro camera will be in webcam mode when connected to a computer. This mode is subject to certain constraints. Resolution limitations apply, with the user able to select from 480p,

720p and 1080p for the Hero 10 Black [6]. FOV limitations also apply, with the user able to choose between these four options: Wide, Narrow, Superview, and Linear [6].

4.6.2 OpenCV Live Feed Preview Class 'camera.py'

This class implements and handles the live feed, keypress events, image processing, object detection, and other functions through the openCV library.

Features of this class:

- **Real-time Preview**: Utilises OpenCV to provide a live feed preview of the GoPro Hero 10's webcam, ensuring low latency and high-quality streaming.
- **Image Segmentation**: Offers tools for segmenting the live feed into different regions, which can be used for background subtraction, object tracking, and other vision-based tasks.
- **Object Detection**: Enables detection of objects (RC cars) within the live feed using colour segmentation.
- **Customisation**: Allows users to implement custom image processing techniques and algorithms tailored to specific tasks or research requirements.

In the appendix C, a sequence diagram details how operations are carried out between both classes, the 'main.py' and the Camera itself.

In addition, the source code incorporates the previously mentioned codes of camera calibration, ArUco markers, colour detection of cars, and position estimation. The camera calibration code facilitates the correction of lens distortion and the determination of the camera's intrinsic parameters, which are crucial for accurately tracking objects and estimating their positions. The ArUco marker code enables the detection and identification of these fiducial markers, which can establish a coordinate system and track the position and orientation of objects in the scene. The cars' colour detection code utilises computer vision techniques to identify and segment RC cars based on their distinctive colours, facilitating their tracking and analysis.

Finally, the position estimation code obtains the position from each car in pixel coordinates, which are obtained in the colour detection code, and transforms them into centimetre coordinates. Subsequently, the estimated position in centimetres is fed into an EKF for position estimation. The estimated position is then transmitted to the control unit.

Chapter 5

Conclusions

This dissertation successfully achieved several key objectives in developing an autonomous remote-controlled (RC) car system. Primarily, it established a robust communication link between the GoPro camera and the PC using the HTTP protocol, leveraging the OpenGoPro HTTP API. This open-source tool proved instrumental in facilitating seamless interaction and control of the GoPro camera [7].

The research also accomplished accurate camera calibration, enhancing the fidelity of visual data by rectifying the slight distortion observed in the GoPro's linear mode. Furthermore, the system demonstrated proficiency in detecting all vehicles within the camera frame. While the initial approach using ArUco markers showed promise, their small size relative to the camera-racetrack distance rendered them undetectable. Consequently, a more effective method was developed, utilising the chassis colours for RC car detection. This approach proved robust under various lighting conditions, speeds, and positions.

A significant achievement was transforming pixel coordinates to real-world metric coordinates successfully. This was accomplished by calculating a pixel-to-millimeter ratio and defining the racetrack's bottom-right corner as the origin. Physical measurements verified the accuracy of these camera-based position readings, confirming a small error.

The Extended Kalman Filter (EKF) implementation in the Unity-based simulation environment was another notable success. To enhance realism and test system robustness, synthetic noise was introduced, simulating real-world sensor imperfections. This approach served multiple purposes:

- **EKF Performance Validation**: Assessing the filter's ability to provide accurate state estimates amidst noise.
- **Reinforcement Learning (RL) Preparation**: Training the RL algorithm with noisy data to enhance its real-world applicability.
- **Early Issue Identification**: Enabling the detection and addressing of potential problems before real-world deployment.

These measures ensured that the EKF algorithm was thoroughly tested and capable of handling real-world uncertainties.

Despite these achievements, the research encountered several challenges requiring further investigation. The ArUco marker detection issue necessitated alternative approaches. Optimising colour and circle detection demanded extensive research to fine-tune detection parameters. Hardware limitations of the available laptop posed difficulties in achieving high accuracy in real-time vehicle pose and motion processing.

It is worth noting that the EKF implementation was confined to the simulation environment and not extended to real-world testing. As this project was in its inaugural year, substantial time was devoted to theoretical research and problem comprehension, laying a solid foundation for future developments in this field.

Future Work

Building upon the foundation laid by this thesis, several avenues for future work and potential enhancements emerge:

1. **Real-world Implementation**: The primary focus should be transitioning the system from a controlled experimental environment to real-world scenarios. This involves rigorous testing and refinement to ensure the system's robustness against unpredictable variables and challenges encountered in actual racing conditions.

2. **Obstacle Avoidance System**: Enhancing autonomous vehicles with sophisticated obstacle detection and avoidance capabilities is crucial. This can be achieved by developing advanced algorithms to process this data in real-time, allowing vehicles to navigate safely around obstacles. The integration of supplementary sensors, such as LiDAR, may also prove beneficial. However, their costs are typically higher than that of a camera.

3. **Advanced Machine Learning for Vehicle Detection**: Leveraging state-of-the-art machine learning algorithms, particularly deep learning models, can significantly improve vehicle detection accuracy and reliability. Training these models on diverse, extensive datasets will enhance the system's ability to identify and track vehicles under varying lighting conditions, speeds, and angles.

By addressing these recommendations, the project can overcome current limitations, push the boundaries of autonomous racing technology, and pave the way for more advanced, reliable, and exciting autonomous racing experiences.

Appendix A

Technical Specs

A.1 ESP32 S2 Mini specs

Table A.1: Tech Specs of the ESP32 S2 Mini [25].

Processor	ESP32-S2FN4R2 WiFi SoC
Clock Speed	up to 240 MHz
Flash Memory	4MB
PSRAM	2MB
Operating Voltage	3.3V
Digital I/O Pins	27
USB	Type-C USB
Interfaces	ADC, DAC, I2C, SPI, UART, USB OTG
Dimensions	34.3mm x 25.4mm
Weight	2.4g
Default Firmware	MicroPython
Compatibility	MicroPython, Arduino, CircuitPython, ESP-IDF

A.2 GoPro Hero 10 Black Tech specs

Table A.2: Specifications of GoPro Hero 10 Black.

Video Resolution	5.3K, 5K, 4K, 2.7K, 1080p
Frame Rate (FPS)	up to 240 fps
Digital Lenses	SuperView, Wide, Linear, Linear + Horizon Leveling, Narrow
Aspect Ratios	16:9, 4:3
Connectivity	WI-FI, Bluetooth, USB-C
Webcam Mode	Yes
Battery	1720mAh rechargeable Lithium-ion

Appendix B

Code

B.1 Camera Calibration

```
b = 7
a = 9
chessboardSize = (a, b)
frameSize = (640,480)

# termination criteria
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

# prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
objp = np.zeros((chessboardSize[0] * chessboardSize[1], 3), np.float32)
objp[:,:2] = np.mgrid[0:chessboardSize[0],0:chessboardSize[1]].T.reshape(-1,2)

size_of_chessboard_squares_mm = 20
objp = objp * size_of_chessboard_squares_mm

# Arrays to store object points and image points from all the images.
objpoints = [] # 3d point in real world space
imgpoints = [] # 2d points in image plane.

#images = glob.glob('/images/*.png')
images = glob.glob(os.path.join(os.getcwd(), 'goProHero10/src/Camera/img_cali', '*.
    png'))

savePath = os.path.join(os.getcwd(), "goProHero10/src/Camera/cali_chess_linear")
if (os.path.isdir(savePath) is not True):
    try:
        os.mkdir(savePath)
    except:
        pass
#end-if-else

```

```
31 newImagePrefix = "img_cal_"
32 newImageExtension = ".png"
33 maxLeadingZeros = 4
34
35 imgCounter = 0
36 for image in images:
37     print(f"Reading {image} ...")
38     img = cv2.imread(image)
39     gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
40
41     # Find the chess board corners
42     ret, corners = cv2.findChessboardCorners(gray, chessboardSize, None)
43
44     # If found, add object points, image points (after refining them)
45     if ret == True:
46         objpoints.append(objp)
47         corners2 = cv2.cornerSubPix(gray, corners, (11,11), (-1,-1), criteria)
48         imgpoints.append(corners)
49
50         # Draw and display the corners
51         cv2.drawChessboardCorners(img, chessboardSize, corners2, ret)
52         cv2.imshow('img', img)
53         cv2.waitKey(1000)
54
55         #---------------
56         filename = newImagePrefix + (maxLeadingZeros - len(str(imgCounter)))*"0"+
               str(imgCounter) + newImageExtension
57         filePath = os.path.join(savePath, filename)
58
59         cv2.imwrite(filePath, img)
60         print("image saved!")
61         imgCounter += 1
62     else:
63         print("Not found ...")
64
65 cv2.destroyAllWindows()
66
67 ############## CALIBRATION ############################################################
68
69 ret, cameraMatrix, distCoeffs, rvecs, tvecs = cv2.calibrateCamera(objpoints,
       imgpoints, frameSize, None, None)
```

Listing B.1: Camera Calibration [18].

B.2 Camera Undistortion

```
############## UNDISTORTION ####################################################

img = cv2.imread('goProHero10/src/Camera/img_cali/img_0008.png')
h,  w = img.shape[:2]
newCameraMatrix, roi = cv2.getOptimalNewCameraMatrix(cameraMatrix, distCoeffs, (w,h), 1, (w,h))

# Undistort
dst = cv2.undistort(img, cameraMatrix, distCoeffs, None, newCameraMatrix)

# crop the image
x, y, w, h = roi
dst = dst[y:y+h, x:x+w]
cv2.imwrite('caliResult1.png', dst)

# Undistort with Remapping
mapx, mapy = cv2.initUndistortRectifyMap(cameraMatrix, distCoeffs, None, newCameraMatrix, (w,h), 5)
dst = cv2.remap(img, mapx, mapy, cv2.INTER_LINEAR)

# crop the image
x, y, w, h = roi
dst = dst[y:y+h, x:x+w]
cv2.imwrite('caliResult2.png', dst)

# Reprojection Error
mean_error = 0

for i in range(len(objpoints)):
    imgpoints2, _ = cv2.projectPoints(objpoints[i], rvecs[i], tvecs[i], cameraMatrix, distCoeffs)
    error = cv2.norm(imgpoints[i], imgpoints2, cv2.NORM_L2)/len(imgpoints2)
    mean_error += error

print( "total error: {}".format(mean_error/len(objpoints)) )
```

Listing B.2: Camera Undistortion [18].

B.3 Pose Estimation

```
def pose_estimation(self, frame, aruco_dict, matrix_coefficients,
    distortion_coefficients):
    ret, frame = self.cap.read()

    with open("cameraMatrix.pkl", "rb") as f:
        matrix_coefficients = pickle.load(f)

    with open("distCoeffs.pkl", "rb") as f:
        distortion_coefficients = pickle.load(f)
    aruco_dict=cv2.aruco.getPredefinedDictionary(cv2.aruco.DICT_6X6_250)

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    parameters = cv2.aruco.DetectorParameters()

    corners, ids, rejected_img_points = cv2.aruco.detectMarkers(gray, aruco_dict,
        parameters=parameters)

    positions_aruco = []  # List to store positions of all detected markers

    # If markers are detected
    if len(corners) > 0:
        for i in range(0, len(ids)):
            # Estimate pose of each marker and return the values rvec and tvec---(
                different from those of camera coefficients)
            rvec, tvec, markerPoints = cv2.aruco.estimatePoseSingleMarkers(corners[
                i], 0.02, matrix_coefficients, distortion_coefficients)

            # Draw a square around the markers
            cv2.aruco.drawDetectedMarkers(frame, corners)


            # Draw Axis
            cv2.drawFrameAxes(frame, matrix_coefficients, distortion_coefficients,
                rvec, tvec, 0.01)

            # Calculate marker position
            position_aruco = self.marker_position(tvec, rvec)

            # Append position to the list
            positions_aruco.append(position_aruco)

    self.positions_aruco = positions_aruco
    return frame, positions_aruco
```

Listing B.3: Pose Estimation [18].

B.4 Marker Position

```
def marker_position(self, tvec, rvec):

    # Convert rotation vector to a rotation matrix
    rotation_matrix, _ = cv2.Rodrigues(rvec)

    # Invert rotation matrix to obtain the transformation from marker to camera
    rotation_matrix_inv = np.linalg.inv(rotation_matrix)

    # Convert translation vector to a column vector
    tvec = tvec.reshape((3, 1))

    # Calculate marker position in camera coordinates using inverse transformation
    marker_pos = -np.dot(rotation_matrix_inv, tvec)

    return marker_pos
```

Listing B.4: Marker Position [18].

B.5 Colour Detection

```
def detect_cars(self, image):
    def detect_blue_cars(image):
        # Convert the image to HSV color space
        image_hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

        # Define the lower and upper bounds for blue color in HSV
        lower_blue = np.array([90, 100, 100])
        upper_blue = np.array([120, 255, 255])

        # Create a mask using the specified range
        mask = cv2.inRange(image_hsv, lower_blue, upper_blue)

        # Perform morphological operations to clean up the mask
        kernel = np.ones((5, 5), np.uint8)
        mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)

        # Find contours in the mask
        contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.
            CHAIN_APPROX_SIMPLE)

        # Filter contours based on area
        min_area = 500
        max_area = 1500
        filtered_contours = [cnt for cnt in contours if min_area < cv2.contourArea(
            cnt) < max_area]

        # Draw the filtered contours on the original image
        result = image.copy()
        cv2.drawContours(result, filtered_contours, -1, (0, 255, 0), 2)

        # Extract the centroid of the largest contour
        centroid = None
        max_contour_area = 0
        for cnt in filtered_contours:
            area = cv2.contourArea(cnt)
            if area > max_contour_area:
                moments = cv2.moments(cnt)
                if moments["m00"] != 0:
                    cX = int(moments["m10"] / moments["m00"])
                    cY = int(moments["m01"] / moments["m00"])
                    if 600 <= cX <= 1650 and cY > 85:  # Check if centroid is
                        within the specified bounds
                        centroid = np.array([cX, cY])
                        max_contour_area = area

        # Draw the centroid on the result image
```

```
44          if centroid is not None:
45              cv2.circle(result, centroid, 10, (255, 255, 255), -1)
46              cv2.putText(result, "centroid", (centroid[0] - 25, centroid[1] - 25),
                    cv2.FONT_HERSHEY_SIMPLEX, 1.0, (255, 255, 255), 2)
47
48          return centroid, result
49
50      def detect_yellow_cars(image):
51          # Convert the image to HSV color space
52          image_hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
53
54          # Define the lower and upper bounds for yellow color in HSV
55
56          lower_yellow = np.array([20, 100, 100])   # Lower bound for yellow
57          upper_yellow = np.array([30, 255, 255])   # Upper bound for yellow
58
59
60          # Create masks using the specified ranges
61          mask1 = cv2.inRange(image_hsv, lower_yellow, upper_yellow)
62          #mask2 = cv2.inRange(image_hsv, lower_red2, upper_red2)
63
64
65          # Perform morphological operations to clean up the mask
66          kernel = np.ones((5, 5), np.uint8)
67          mask1 = cv2.morphologyEx(mask1, cv2.MORPH_CLOSE, kernel)
68          mask1 = cv2.morphologyEx(mask1, cv2.MORPH_OPEN, kernel)
69
70          # Find contours in the mask
71          contours, _ = cv2.findContours(mask1, cv2.RETR_EXTERNAL, cv2.
                CHAIN_APPROX_SIMPLE)
72
73          # Filter contours based on area
74          min_area = 500
75          max_area = 1500
76          filtered_contours = [cnt for cnt in contours if min_area < cv2.contourArea(
                cnt) < max_area]
77
78          # Draw the filtered contours on the original image
79          result = image.copy()
80          cv2.drawContours(result, filtered_contours, -1, (0, 255, 0), 2)
81
82          # Extract the centroid of the largest contour
83          centroid = None
84          max_contour_area = 0
85          for cnt in filtered_contours:
86              area = cv2.contourArea(cnt)
87              if area > max_contour_area:
88                  moments = cv2.moments(cnt)
89                  if moments["m00"] != 0:
```

```
90                      cX = int(moments["m10"] / moments["m00"])
91                      cY = int(moments["m01"] / moments["m00"])
92                      if 600 <= cX <= 1650 and cY > 85:  # Check if centroid is
                            within the specified bounds
93                          centroid = np.array([cX, cY])
94                          max_contour_area = area
95
96          # Draw the centroid on the result image
97          if centroid is not None:
98              cv2.circle(result, centroid, 10, (255, 255, 255), -1)
99              cv2.putText(result, "centroid", (centroid[0] - 25, centroid[1] - 25),
                    cv2.FONT_HERSHEY_SIMPLEX, 1.0, (255, 255, 255), 2)
100
101         return centroid, result
102
103     def detect_pink_cars(image):
104         #Convert the image to HSV color space
105         image_hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
106
107         # Define the lower and upper bounds for pink color in HSV
108         lower_pink1 = np.array([150, 50, 50])
109         upper_pink1 = np.array([170, 255, 255])
110
111         lower_pink2 = np.array([0, 50, 50])
112         upper_pink2 = np.array([10, 255, 255])
113
114         # Create masks using the specified ranges
115         mask1 = cv2.inRange(image_hsv, lower_pink1, upper_pink1)
116         mask2 = cv2.inRange(image_hsv, lower_pink2, upper_pink2)
117
118         # Combine masks
119         mask = cv2.bitwise_or(mask1, mask2)
120
121         # Perform morphological operations to clean up the mask
122         kernel = np.ones((7, 7), np.uint8)  # Increase kernel size for better noise
                reduction
123         mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)
124         mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)  # Additional opening
                to remove smaller noise
125
126         # Find contours in the mask
127         contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.
                CHAIN_APPROX_SIMPLE)
128
129         # Filter contours based on area
130         min_area = 500  # Adjust minimum area threshold
131         max_area = 2500  # Adjust maximum area threshold
132         filtered_contours = [cnt for cnt in contours if min_area < cv2.contourArea(
                cnt) < max_area]
```

```
133
134         # Draw the filtered contours on the original image
135         result = image.copy()
136         cv2.drawContours(result, filtered_contours, -1, (0, 255, 0), 2)
137
138         # Extract the centroid of the largest contour
139         centroid = None
140         max_contour_area = 0
141         for cnt in filtered_contours:
142             area = cv2.contourArea(cnt)
143             if area > max_contour_area:
144                 moments = cv2.moments(cnt)
145                 if moments["m00"] != 0:
146                     cX = int(moments["m10"] / moments["m00"])
147                     cY = int(moments["m01"] / moments["m00"])
148                     if 600 <= cX <= 1650 and cY > 85:  # Check if centroid is
                            within the specified bounds
149                         centroid = np.array([cX, cY])
150                         max_contour_area = area
151
152         # Draw the centroid on the result image
153         if centroid is not None:
154             cv2.circle(result, centroid, 10, (255, 255, 255), -1)
155             cv2.putText(result, "centroid", (centroid[0] - 25, centroid[1] - 25),
                    cv2.FONT_HERSHEY_SIMPLEX, 1.0, (255, 255, 255), 2)
156
157         return centroid, result
158
159
160     def adjust_intensity(image, alpha=0.35, beta=0):
161         # Perform intensity adjustment using alpha blending
162         adjusted_image = cv2.convertScaleAbs(image, alpha=alpha, beta=beta)
163         return adjusted_image
164
165     # Detect blue cars
166     blue_centroids, blue_result = detect_blue_cars(image)
167
168     # Adjust intensity of blue result
169     blue_result_adjusted = adjust_intensity(blue_result)
170
171     # Detect red cars
172     yellow_centroids, yellow_result = detect_yellow_cars(image)
173
174     # Adjust intensity of red result
175     yellow_result_adjusted = adjust_intensity(yellow_result)
176
177     # Detect pink cars
178     pink_centroids, pink_result = detect_pink_cars(image)
179
```

```
180     # Adjust intensity of pink result
181     pink_result_adjusted = adjust_intensity(pink_result)
182 
183     # Combine adjusted results
184     combined_result = cv2.add(cv2.add(blue_result_adjusted, yellow_result_adjusted)
            , pink_result_adjusted)
185 
186     # Combine centroids. If they are none (0, 0) array is put
187     all_centroids = [blue_centroids if blue_centroids is not None else np.array([0,
             0]),
188                      yellow_centroids if yellow_centroids is not None else np.array
                          ([0, 0]),
189                      pink_centroids if pink_centroids is not None else np.array([0,
                           0])]
190 
191     all_cent = np.array(all_centroids)
192     return all_cent, combined_result
```

Listing B.5: Cars' colour detection.

B.6 Pixel-to-millimetre Ratio

```
# Read image
img = cv2.imread("goProHero10/src/Camera/images/linear/img_0006.png")
img_crop = crop_img(img, 600, 80, 1000, 1200)

# Convert image to grayscale
img_gray = cv2.cvtColor(img_crop, cv2.COLOR_BGR2GRAY)
show_image("Grayscale Image", img_gray)

# Convert images to binary for paper detection
_, img_bin = cv2.threshold(img_gray, 160, 255, cv2.THRESH_BINARY)
show_image("Binary Image", img_bin)

# Opening binary image to remove noise around the paper
close_kernel = np.ones((9, 9), np.uint8)
img_dilated = cv2.dilate(img_bin, close_kernel, iterations=1)
img_closed = cv2.erode(img_dilated, close_kernel, iterations=1)
show_image("Opened Image", img_closed)

# Find contours
image_contours, _ = cv2.findContours(img_closed, cv2.RETR_EXTERNAL, cv2.
    CHAIN_APPROX_SIMPLE)

# Filter contours based on area and aspect ratio
paper_contours = []
for contour in image_contours:
    area = cv2.contourArea(contour)
    if area < 1000:  # Ignore very small contours
        continue

    x, y, w, h = cv2.boundingRect(contour)
    aspect_ratio = float(w) / h
    # Check if the contour has an aspect ratio similar to A4 paper (210:297 ~
        0.707)
    if 0.6 < aspect_ratio < 0.85:
        paper_contours.append(contour)

# Draw all contours for debugging purposes
img_contours = cv2.drawContours(img_crop.copy(), image_contours, -1, (0, 255, 0),
    2)
show_image("All Contours", img_contours)

# If no paper-like contours are found, raise an error
if not paper_contours:
    raise ValueError("No contours found with the expected area and aspect ratio")

# Find the largest contour among the filtered contours
```

```
largest_contour = max(paper_contours, key=cv2.contourArea)

# Draw the largest paper-like contour
img_largest_contour = cv2.drawContours(img_crop.copy(), [largest_contour], -1, (0, 0, 255), 2)
show_image("Largest Paper-Like Contour", img_largest_contour)

# Approximate the contour with a 4-sided polygon
epsilon = 0.02 * cv2.arcLength(largest_contour, True)
approx = cv2.approxPolyDP(largest_contour, epsilon, True)

if len(approx) != 4:
    raise ValueError("Could not find a quadrilateral. Found vertices: {}".format(len(approx)))
else:
    # Define the coordinates of the paper's vertices
    top_left = approx[0][0]
    top_right = approx[1][0]
    bottom_right = approx[2][0]
    bottom_left = approx[3][0]
    paper_coords = np.float32([top_left, top_right, bottom_right, bottom_left])

    # Draw the approximated contour
    img_approx = cv2.drawContours(img_crop.copy(), [approx], -1, (255, 0, 0), 2)
    for point in paper_coords:
        cv2.drawMarker(img_approx, tuple(point.astype(int)), (0, 0, 255), markerType=cv2.MARKER_CROSS,
                       markerSize=10, thickness=2, line_type=cv2.LINE_AA)
    show_image("Approximated Contour", img_approx)

    # Define the coordinates of the rectangle's vertices
    dst_height = max(np.linalg.norm(top_left - bottom_left), np.linalg.norm(bottom_right - top_right))

    pixel_to_mm = dst_height / 297.0
    print("Pixel to millimeter ratio =", pixel_to_mm, "pixels/mm")
```

Listing B.6: px2mm.py.

B.7 Circle Detection

```python
def find_circle(self, img):
    # Convert image to gray
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Adaptive thresholding for better binary conversion
    img_bin = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.
        THRESH_BINARY, 11, 2)

    # Opening binary image to remove noise around the circles
    close_kernel = np.ones((3, 3), np.uint8)
    img_dilated = cv2.dilate(img_bin, close_kernel, iterations=1)
    img_closed = cv2.erode(img_dilated, close_kernel, iterations=1)

    # Find circles using Hough Circle Transform
    circles = cv2.HoughCircles(
        img_closed,
        cv2.HOUGH_GRADIENT,
        dp=1.0,
        minDist=100,  # Adjusted minimum distance between circles
        param1=50,
        param2=25,  # Adjusted parameter for circle detection sensitivity
        minRadius=15,
        maxRadius=25
    )

    # List to store the centers of the circles
    centers = []

    # If circles are detected, draw them on the original image and store the
        centers
    if circles is not None:
        circles = np.uint16(np.around(circles))
        for i in circles[0, :]:
            center = np.array([i[0], i[1]])
            centers.append(center.tolist())  # Append as a list
            # Draw the outer circle
            cv2.circle(img, (i[0], i[1]), i[2], (0, 255, 0), 2)
            # Draw the center of the circle
            cv2.circle(img, (i[0], i[1]), 2, (0, 0, 255), 3)

    return img, np.array(centers)
```

Listing B.7: find_cicle.

Appendix C

Sequence Diagram

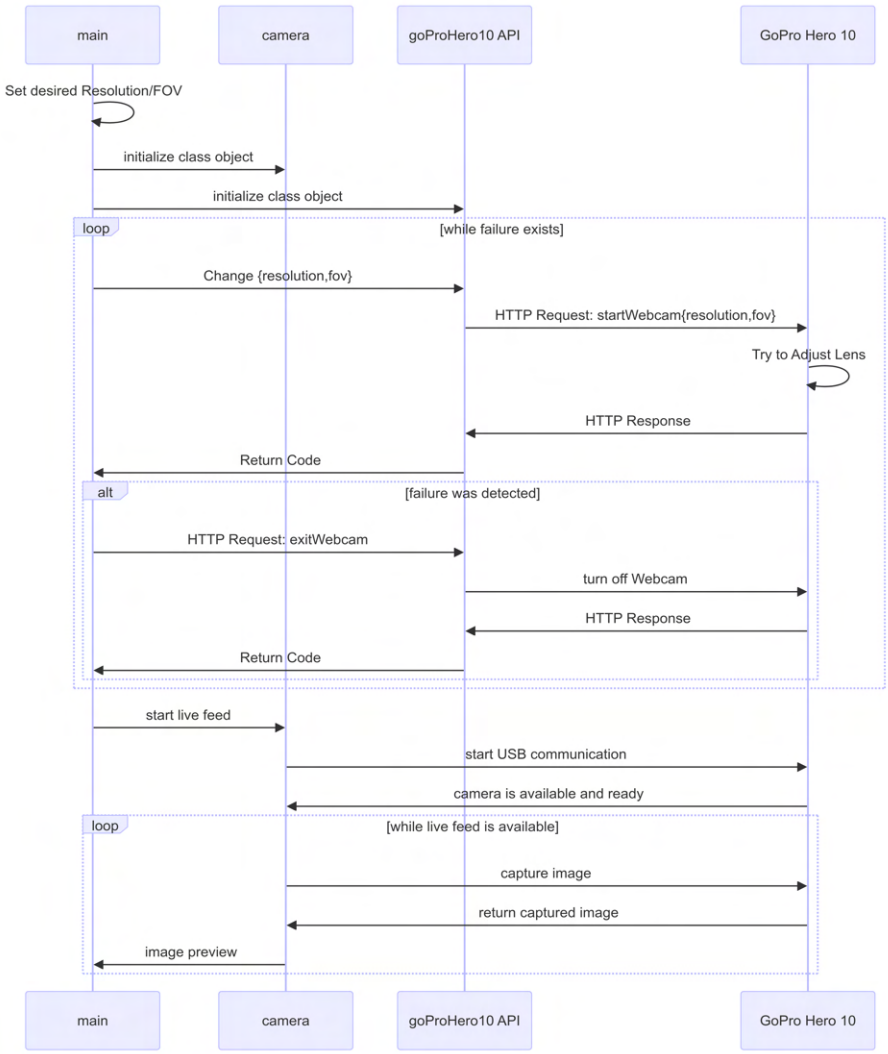


Figure C.1: Webcam Sequence Diagram.

References

[1] A2RL. Abu dhabi autonomous racing league, 2024. Available at https://a2rl.io/.

[2] Blender Foundation. Blender features, 2024. Available at https://www.blender.org/features/.

[3] Gary Bradsky. Camera calibration with opencv. Available at https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html.

[4] Gary Bradsky. Detection of aruco markers with opencv. Available at https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html.

[5] F1tenth. F1tenth autonomous racing competition, 2024. Available at https://f1tenth.org/.

[6] GoPro, Inc. Gopro webcam: How to use your gopro as a webcam, 2024. Available at https://community.gopro.com/s/article/GoPro-Webcam?language=en_US#adjustresolution.

[7] GoPro, Inc. Open gopro http api documentation, 2024. Available at https://gopro.github.io/OpenGoPro/http.

[8] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, Cambridge, UK, 2nd edition, 2003.

[9] Institution of Mechanical Engineers. Formula student driverless, 2024. Available at https://www.formulastudent.de/pr/news/details/article/formula-student-germany-ev-honored-with-bertha-and-carl-benz-prize/.

[10] S.J. Julier and J.K. Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004.

[11] Ho Chuen Kam, Ying Kin Yu, and Kin Hong Wong. An improvement on aruco marker for pose tracking using kalman filter. In *2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pages 65–69, 2018.

[12] Marcus Law. Top 10: Autonomous vehicle companies, september 2023. Available at https://technologymagazine.com/top10/top-10-autonomous-vehicle-companies.

[13] Alex Liniger. Autonomous rc car racing, 2015. Available at https://control.ee.ethz.ch/research/team-projects/autonomous-rc-car-racing.html.

[14] MathWorks. *Camera Calibration*, 2024. Available at https://www.mathworks.com/help/vision/ug/camera-calibration.html.

[15] MathWorks. *Fisheye Calibration Basics*, 2024. Available at https://www.mathworks.com/help/vision/ug/fisheye-calibration-basics.html.

[16] Pole Position Modelismo. Kyosho 87033-01b - urethane circuit 30 expansion set (63pcs). Available at https://www.poleposition-pt.com/en/kyosho-87033-01b-urethane-circuit-30-expansion-set-63pcs-K87033-01B/p711.

[17] A. Paulo G. M. Moreira. Probabilistic localization. PowerPoint presentation, 2023. Presented at the Faculdade de Engenharia da Universidade do Porto, Mestrado Integrado em Engenharia Eletrotécnica e de Computadores, Sistemas Autónomos – MEEC 2022/23.

[18] Gonçalo Mota. goprohero10. https://github.com/Gugon11/goProHero10, 2024.

[19] National Oceanic and Atmospheric Administration. What is lidar?, january 2023. Available at https://oceanservice.noaa.gov/facts/lidar.html.

[20] Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J. Andrew Bagnell, Pieter Abbeel, and Jan Peters. An algorithmic perspective on imitation learning. *Foundations and Trends® in Robotics*, 7(1-2):1–179, 2018.

[21] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[22] Dan Simon. *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*. John Wiley & Sons, 2006.

[23] SuccessStory. Gopro story - profile, history, founder, founded, ceo, 2024. Available at https://successstory.com/companies/gopro-inc.

[24] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA, 2nd edition, 2018.

[25] Espressif Systems. *ESP32-S2 Datasheet*, June 2020. Available at https://www.espressif.com/sites/default/files/documentation/esp32-s2_datasheet_en.pdf.

[26] United Nations. Sustainable development goals, 2024. Available at https://sdgs.un.org/goals.

[27] Eric Wan. Sigma-point filters: An overview with applications to integrated navigation and vision assisted control. In *2006 IEEE Nonlinear Statistical Signal Processing Workshop*, pages 201–202, 2006.

[28] Greg Welch and Gary Bishop. An Introduction to the Kalman Filter. Technical report, University of North Carolina at Chapel Hill, Department of Computer Science, 1995.

[29] WEMOS. S2 mini-wemos documentation, 2021. Available at https://www.wemos.cc/en/latest/s2/s2_mini.html.

[30] Karl Johan Åström and Richard M. Murray. *Feedback systems: an introduction for scientists and engineers. 2nd Edition*. Princeton university press, 2019.