UNIVERSITEIT•STELLENBOSCH•UNIVERSITY
jou kennisvennoot • your knowledge partner

Guidance and Control for Intercepting a Moving Target using an Autonomous Vehicle

Gideon J. Boshoff
20760523

Report submitted in partial fulfilment of the requirements of the module Project (E) 448 for the degree Baccalaureus in Engineering in the Department of Electrical and Electronic Engineering at Stellenbosch University.

Supervisor: Dr JAA Engelbrecht

November 2020

Acknowledgements

I firstly would like to thank Dr Japie Engelbrecht for making this project possible and for all the help and good conversations you provided. I am grateful for the guidance you gave me, gently pointing out errors and generously supporting any ideas I had for this project.

I would like to give a big thank you to my family at home. To my parents, Jaco and Annalie Boshoff, thank you for never hesitating to help me and support my every need wherever you could. Every little bit of effort you both did for me never went unnoticed. To my two sisters, Collette and Mia Boshoff, thank you for celebrating with me every time I shared my progress with you. It was really encouraging to have you two at home cheering me on.

I would like to thank my beautiful fiancé, Clarischa Ray, from the bottom of my heart. You have supported me from the first day of this project and encouraged me all the way until the end. I am the luckiest man in the world to have such a selfless and strong woman by my side who was always willing to be there for me, even when life was crazy on your side. I would like to extend my thanks to the Ray family for always welcoming me into your home and giving me much needed support wherever you could. I will always be reminded of your love and hospitality whenever I look back on this project.

To my Eendrag brothers and my close friends outside of Eendrag, thank you for always being there to listen to my troubles and to go out of your way to help me with any problems I faced. The selflessness you all showed me is something I learnt from every day.

I would also like to thank my Hillsong church family for the love, support, and community you all provided to me during this project and my time at Stellenbosch University. I am grateful for you all and I am thankful for the journey you have all walked with me.

Lastly, and most importantly, I would like to thank my Lord and Saviour Jesus Christ. All glory goes to You for carrying me through this project and it is by Your Grace alone that I was able to finish this project. Your faithful love endures forever.

But he said to me, "My grace is sufficient for you, for my power is perfected in weakness." Therefore, I will most gladly boast all the more about my weaknesses, so that Christ's power may reside in me. So I take pleasure in weaknesses, insults, hardships, persecutions, and in difficulties, for the sake of Christ. For when I am weak, then I am strong.

2 Corinthians 12:9-10

UNIVERSITEIT•STELLENBOSCH•UNIVERSITY
jou kennisvennoot • your knowledge partner

Plagiaatverklaring / *Plagiarism Declaration*

1. Plagiaat is die oorneem en gebruik van die idees, materiaal en ander intellektuele eiendom van ander persone asof dit jou eie werk is.
   Plagiarism is the use of ideas, material and other intellectual property of another's work and to present is as my own.

2. Ek erken dat die pleeg van plagiaat 'n strafbare oortreding is aangesien dit 'n vorm van diefstal is.
   I agree that plagiarism is a punishable offence because it constitutes theft.

3. Ek verstaan ook dat direkte vertalings plagiaat is.
   I also understand that direct translations are plagiarism.

4. Dienooreenkomstig is alle aanhalings en bydraes vanuit enige bron (ingesluit die internet) volledig verwys (erken). Ek erken dat die woordelikse aanhaal van teks sonder aanhalingstekens (selfs al word die bron volledig erken) plagiaat is.
   Accordingly all quotations and contributions from any source whatsoever (including the internet) have been cited fully. I understand that the reproduction of text without quotation marks (even when the source is cited) is plagiarism

5. Ek verklaar dat die werk in hierdie skryfstuk vervat, behalwe waar anders aangedui, my eie oorspronklike werk is en dat ek dit nie vantevore in die geheel of gedeeltelik ingehandig het vir bepunting in hierdie module/werkstuk of 'n ander module/werkstuk nie.
   I declare that the work contained in this assignment, except where otherwise stated, is my original work and that I have not previously (in its entirety or in part) submitted it for grading in this module/assignment or another module/assignment.

20760523 Studentenommer / *Student number*	 Handtekening / *Signature*
GJ Boshoff Voorletters en van / *Initials and surname*	09/11/2020 Datum / *Date*

Abstract

English

Proportional navigation is a guidance law commonly used in guiding air-to-air missiles to intercept aircrafts. This project aims to apply this guidance law to navigate an autonomously-controlled vehicle to track and intercept a moving target.

A radio-controlled car was modelled using a kinematic bicycle model, and a turning rate controller was designed to interface with the car model using proportional navigation. A controller was designed for a vision-based seeker system to point at a target. The model and controllers were verified with simulations. The results proved that the proportional navigation was implemented successfully - with some inefficiencies caused by physical limitations of the model.

A practical demonstrator was designed and built to test the system in the real world. The demonstrator was more ineffective than the theoretical system, yet still successful.

This report shows that proportional navigation can be adapted for an autonomous car; however, the physical limitations of a car restrict the guidance system's effectiveness.

Afrikaans

Proporsionele navigasie is die leidings reël wat deur lug-tot-lug missiele gebruik word om vliegtuie te onderskep. Die doel van hierdie projek is om die beginsel van proporsionele navigasie toe te pas deur 'n outonoom-beheerde voertuig 'n bewegende teiken te laat agtervolg en onderskep.

'n Kinematiese fiets model is gebruik in die modellering van 'n radio-beheerde voertuig tesame met 'n hoeksnelheid beheerder wat ontwerp is om die model voertuig te koppel met 'n proporsionele navigasie sisteem. 'n Beheerstelsel is ontwerp vir 'n visie-gebaseerde soeker sisteem vir teiken bepaling. Die model en beheerders is met simulasies geverifieer. Die resultate bewys dat proposionele navigasie suksesvol toegepas was alhoewel die model voertuig se fisieke beperkinge vir verminderde effektiwiteit verantwoordelik was.

Die sisteem is getoets deur 'n praktiese demonstrasie. Die demonstrasie was nie so effektief soos die teoretiese sisteem nie, maar was tog suksesvol.

Hierdie verslag bewys dat proporsionele navigasie wel vir 'n outonomiese voertuig gebruik kan word. Die enigste voorbehoud was die voertuig se fisieke beperkinge wat die effektiwiteit ietwat ingeperk het.

Contents

List of Figures

List of Tables

Nomenclature

Variables

V	Velocity
X	Position on the x-axis
Y	Position on the y-axis
ω	Angular rate
δ	Turning angle
ψ	Heading angle
β	Slip angle at the center of gravity
l	Length dimension
N	Proportional constant

Subscripts

T	Target car
M	Main car
f	Front dimension
r	Rear dimension
$\_0$	Initial condition
R	Length of line of sight
cam	Camera

Acronyms and abbreviations

CAD	Computer-Aided Design
DOF	Degrees Of Freedom
ESC	Electronic Speed Controller
GPS	Global Positioning System
Gyro	Gyroscope
I2C	Inter-Integrated Circuit
IBVS	Image-Based Visual Servoing
IDE	Integrated Development Environment
I/O	Input/Output
LiPo	Lithium Polymer
LOS	Line Of Sight
MP	Mega-Pixel
OS	Operating System
PBVS	Position-Based Visual Servoing
PD	Proportional-differential
Pi-Cam	Raspberry Pi Camera
PWM	Pulse-Width Modulation
RC	Radio-Controlled
R-Pi	Raspberry Pi
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus

Chapter 1

Introduction

This chapter provides context for the research topic by presenting the background. The problem this project aims to solve is defined to provide a clear direction for this report. The scope and objectives for the problem statement are outlined and a summary of the report content is given.

1.1. Background

Proportional navigation is a guidance technique used to guide air-to-air missiles to intercept fighter aircrafts. It is a guidance law which is based on the fact that two vehicles will collide if each vehicle's line-of-sight (LOS) does not change direction as the distance between them closes [2]. This scenario is instigated by the navigation system which induces a missile turning rate proportional to the angular rate of the missile's LOS. This will command the missile to *intercept* a target - which is in contrast to pure-pursuit navigation which simply tries to command a missile to follow behind its target until a collision occurs [3].

This method of navigation is generally used for air-to-air interceptor missiles, which have a full three-dimensional range of motion and a seeker system to track the target. However, this guidance law can be used on any vehicle; therefore, a simplified, two-dimensional version, can be designed for an autonomous car with an adapted seeker system.

An autonomous car is defined as a car that can operate without any human intervention or control. In most cases, autonomous cars are able to sense their environment, detect objects, interpret sensory inputs, and identify navigation paths. The concept of controlling a car using computer vision has revolutionized the development of the autonomous car and it is currently the basis of today's driverless systems [4].

This project aims to bring the field of missile guidance and autonomous vehicles together to create a unique solution for guiding an autonomous car to intercept a target using a vision-based seeker system.

1.2. Problem Statements

This project aims to solve two main problems:

- *To design a guidance and control system to enable a car to autonomously track and intercept a moving target.*
- *To design and build a physical system to demonstrate the implementation of the theoretical system.*

1.3. Scope

This problem can be solved and demonstrated using any vehicle, as long as it can be mathematically modelled and controlled. A missile or an unmanned aircraft is generally used in proportional navigation, but this would create a problem that would need to be solved in three dimensions. Due to time limitations of this project, the vehicle in context of the problem statement will be a car so that the scope is limited to only two dimensions. This proposed system could also be applied to a life-size autonomous car; however, due to time and financial restrictions, the scope of this project will focus on implementing the guidance and control system on a radio-controlled (RC) model car.

There are a variety of methods used to track targets. A few examples are radar guidance, radio-frequency tracking, radiant-infrared energy tracking, GPS or laser guidance [3]. The physical system limits the scope to colour detection from a camera input for target tracking because this method is more software-based, therefore less hardware dependant. Computer vision control of an autonomous car is also the most popular choice for driverless systems. This will function as the vehicle's seeker system.

1.4. Objectives

Certain objectives need to be met to create a solution for the problem statements. These objectives can be split into two categories:

Theoretical System

- Model the equations of motion of a car in two dimensions.
- Design a controller to manipulate the car model's motion.
- Design a guidance system (with a seeker system) which can interface with the car model to control it to intercept a moving target.
- Create a simulation of the entire system to validate the model and controllers.

Physical System

- Implement the theoretical control systems on an embedded system to control a pre-built RC car.
- Design an image processing system to control a camera to point at a target.
- Interface between the image processing system, on-board sensors, and the RC car controller.
- Set up a demonstration environment to validate the physical system.

1.5. Report Overview

Chapter 1: Introduces and defines the research topic and problem statements, and also provides an overview of the project background and objectives.

Chapter 2: Reviews literature based on the fields covered by this report and gives context for the research topic in relation to previous projects with similar elements.

Chapter 3: Gives an overview of the theoretical system before explaining the design process behind the mathematical models and controllers.

Chapter 4: Explains the simulation environment and reports the results from a variety of simulations used to validate the theoretical solution.

Chapter 5: Provides an overview of the proposed physical system and breaks the system down into its components with motivations for each choice of component.

Chapter 6: Details the hardware design process of the solution's physical system, with reference to each physical component's interface and connections.

Chapter 7: Describes the high level functionality of the code used for the software design of the physical system.

Chapter 8: Reports on the measured results obtained from individual practical tests as well as the final integrated tests done on the physical system.

Chapter 9: Discusses the results obtained from the theoretical and practical tests, and provides conclusions and recommendations based on these discussions.

Chapter 2

Literature Review

The literature reviewed for this project includes literature on the development of autonomous cars, the various mathematical models that describe a car in motion, methods for guidance control for interception purposes, and control systems based on camera input. This chapter provides background and context for the research topic.

Autonomous cars have been around since the 1920s. In 1980, Mercedes-Benz made a pivotal breakthrough in this industry after developing a vision-guided autonomous car, which paved the way for autonomous cars of the future [4]. Today, most driverless systems create an internal map of their surroundings, based on a variety of sensors, including high-powered cameras [5]. As one would expect, the literature reveals that the methods used for cars to sense their environment are greatly simplified for radio-controlled (RC) cars. These systems tend to rely on simple cameras for vision-based control due to the availability and cost of technology at such a small physical scale [6].

A mathematical model suitable for controlling a car is required when designing an autonomous car control system. The two primary models used when modelling a car are the full-track model and the bicycle model [1] - each with a varying number of degrees of freedom. The full-track model describes a car with four wheels that may experience slippage and are separate from the car body. The bicycle model simplifies a car to a single track with only a front and a rear wheel. This simplified model is believed to capture enough of the vehicle's dynamics with suitable accuracy [1].

When comparing a two-degrees-of-freedom bicycle model with a fourteen-degrees-of-freedom full-track model, the reviewed literature shows that the simplified bicycle model is adequate for modelling a car for control purposes [7]. Other literature has also made comparisons between a kinematic bicycle model and a linear dynamic bicycle model [8]. The kinematic bicycle model focuses on the motion of a car model without regard for the forces that cause it, and the dynamic bicycle model includes the forces that cause the vehicle's motion. The literature concluded that the kinematic bicycle model performs satisfactorily for autonomous driving [8].

These mathematical models have also been applied to the modelling and control of an RC car. Donkers, Hendrix, and Romijn implemented the kinematic bicycle model in their paper on an RC car using an integrated controller for front steering and rear torque vectoring [9]. This paper highlighted the success of modelling and autonomously controlling an RC car in this way, however, there were some apparent control challenges with the system when operating the RC car at very high speeds [9]. A paper done by students from the Chalmers University of Technology had a similar RC car set-up with the aim of controlling an RC car at high speeds [10]. The conclusions made were very similar: in both papers, the car performed satisfactorily using a kinematic bicycle model, and model inaccuracies were noticed at very high speeds.

The reviewed literature highlights two missile guidance systems: proportional navigation and line-of-sight command guidance [3]. Proportional guidance involves either radio-frequency or radiant-infrared energy which is tracked by a seeker-head on the missile. This energy signal is either emitted by the target or by an illuminating radar on the ground. Using this emitted energy, the line of sight (LOS) of the missile seeker will aim towards its target and the missile is then navigated by commanding a missile turning rate proportional to the LOS turning rate [3]. The ratio between the turning rate of the missile and the LOS is called the proportional constant. This guidance system should have a proportional constant greater than 1 so that the missile can turn faster than its LOS, which allows the missile to build up a lead angle relative to the LOS. This lead angle will put the missile on a collision course with its target, given that both missile and target are moving at a constant velocity without manoeuvring [3].

There were very few resources available on the topic of choosing an appropriate constant for proportional navigation. Shukla and Mahapatra tested multiple guidance scenarios with changes in the proportional constant with no clear conclusion [11]. However, there was an indication that the maximum value for the proportional constant could be limited by the vehicles physical capabilities [11]. Choosing a value in the range from 2 to 6 was the only other guide found for selecting an appropriate proportional constant value [3].

Line-of-sight command guidance systems use two tracking radars to track the missile and target simultaneously from the launch site. The literature also describes two forms of line-of-sight command guidance: three-point guidance and lead-angle guidance - where the latter system is a variation of the former system [3]. Three-point guidance navigates the missile so that its flight path remains on the LOS (or tracking line) between the point of launch and the target [3]. Lead-angle guidance uses the difference between the point of launch, the target, and the missile to calculate a distance as a function of this difference. The missile is then commanded to fly at this calculated distance ahead of the

LOS. According to the literature, the dynamic analysis of these two forms of command guidance is the same. [3]. The seeker systems used for missile guidance seem to be more suited for large scale use and can be difficult to implement on a smaller physical scale.

Camera technology is simple and cost-effective enough to be used on smaller systems. Visual servoing is a method used for vision-guided control and involves cameras acquiring information for feedback control of a robot or parts of it [12]. The reviewed literature classifies visual servoing into two physical categories: eye-to-hand and eye-in-hand. Eye-to-hand visual servoing uses a fixed camera separate from the controller/manipulator, and eye-in-hand servoing uses a camera mounted onto the controller/manipulator [12]. Eye-to-hand (or indirect/external) visual servoing has an "easy" control law and is appropriate for situations with slow movements. Eye-in-hand (or direct/internal) visual servoing has a more complex control law involving kinematics or dynamics and is more appropriate for fast movement situations [12]. Both these methods are applicable in the field of autonomous RC car control and examples of each method were found in the reviewed literature. Eye-to-hand visual servoing has been previously used for path planning control for an autonomous RC car [10], whereas eye-in-hand visual servoing has been more appropriately used for sensory control and object detection for an autonomous RC car [6].

Chapter 3

Theoretical System Design

This chapter provides an overview of the proposed theoretical system with a breakdown of the system into its components. Design choices for each component are stated and motivated in this chapter with appropriate calculations to support any design choices. This chapter aims to validate the role of each component in achieving the theoretical project objectives.

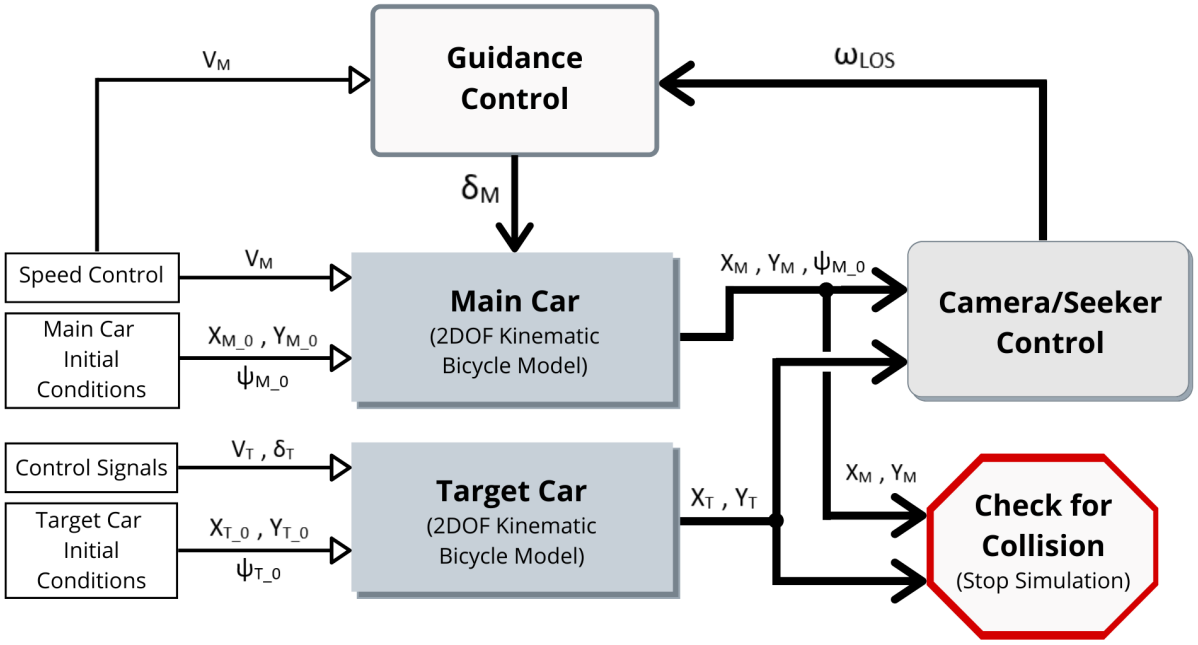


Figure 3.1: Theoretical system diagram

3.1. Mathematical Car Model

Based on the reviewed literature, it was decided to use a kinematic bicycle model with two degrees of freedom for my car model. This decision was based on the fact that this simplified model captures enough of the vehicle's dynamics with suitable accuracy for control purposes. This model has also been previously implemented in autonomously controlling an RC model car [8].

This vehicle model also includes an angle of slip (β). This parameter simplifies the model so that only the inertial axis system (X,Y) is considered and the vehicle's axis system is ignored. This is because the turning rate (ω) for this model is calculated using angle of slip (Equation 3.3) instead of the vehicle's axis system. Alternative models implement the steering angle (δ) with the vehicle's axis system to calculate the turning rate [8]. For the kinematic bicycle model, the steering angle is only used with the equation for the angle of slip (Equation 3.4), allowing the model to base all its equations on only inertial axis system.

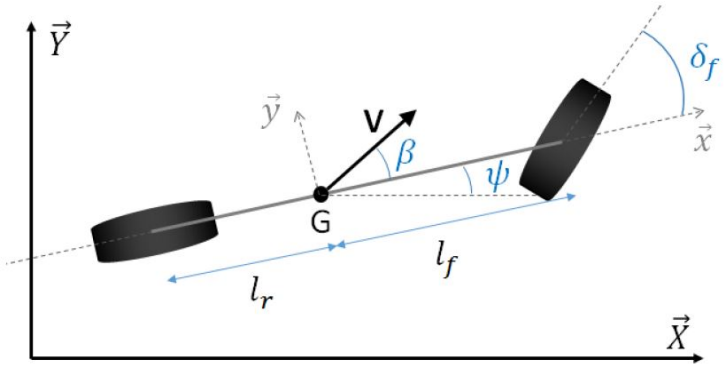


Figure 3.2: Kinematic bicycle model of vehicle (from [1])

Figure 3.2 shows the kinematic bicycle model used in this design. In this model, both the two front and back wheels are represented as a single wheel (like a bicycle). The following assumptions are made when using this model [13]:

- It is a rigid body moving on a geometric plane.
- Longitudinal and lateral load transfers are neglected.
- Rolling and pitching motions of the body are neglected.
- Aerodynamic forces are neglected.
- Any suspension system is neglected.
- Tire forces are neglected.

This model is governed by the following equations [1]:

$$\dot{X} = V cos(\psi + \beta) \tag{3.1}$$

$$\dot{Y} = V sin(\psi + \beta) \tag{3.2}$$

$$\dot{\psi} = \omega = \frac{V}{l_r} sin(\beta) \tag{3.3}$$

$$\beta = arctan(\frac{l_r}{l_f + l_r} tan(\delta)) \tag{3.4}$$

The variables in the Equations 3.1-3.4 and in Figure 3.2 are described below:

- $\dot{X}, \dot{Y}$ - Velocity component in the X and Y directions.
- V - Magnitude of the velocity of the vehicle's centre of gravity (G) in the direction it is moving.
- ψ - Heading angle of car (direction car is currently pointing) relative to the inertial axis [X,Y].
- β - Angle of slip; which is the difference between where the vehicle is heading (ψ) and where it's actually moving (direction of V).
- ω - Turning rate of vehicle.
- l_r, l_f - Front and rear length dimension from the centre of gravity to wheel axle.
- δ - Turning angle on the front wheel (front wheel steering) relative to the vehicle.

The two inputs to this model are V and δ. V controls the overall speed of the vehicle and δ turns the front tires to change the heading of the vehicle. Turning the tires with δ changes β which in turn changes the direction of V.

3.2. Guidance Control

The reviewed literature clearly concludes that of all the methods commonly used for missile guidance, proportional guidance is the superior method [3]. To meet the theoretical project objectives, a proportional guidance controller will be designed for the modelled vehicle.

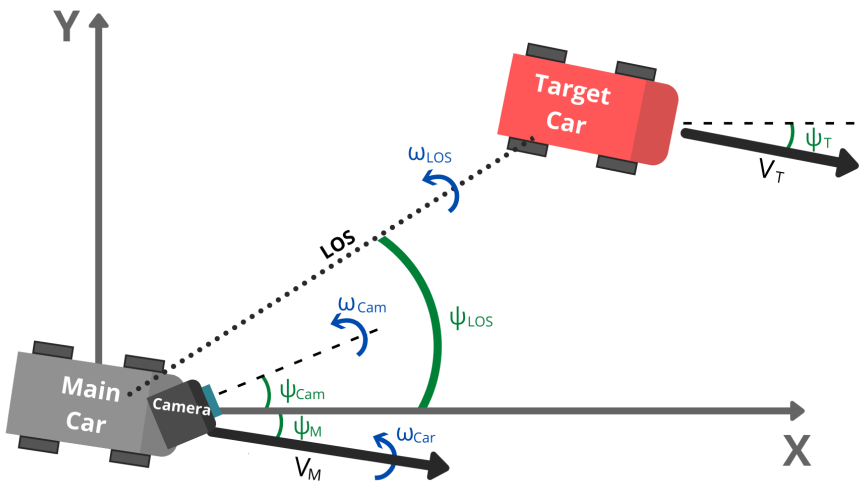


Figure 3.3: Reference diagram for heading angles and turning rates

3.2.1. Proportional Guidance Constant

The line of sight (LOS) of the vehicle is an invisible line between the main car and its target. The LOS in this proposed system will be the line of vision of the camera, which will be controlled to point at the target (refer to Section 3.3). As the camera turns to look at the target, a LOS turning rate will be induced (ω_{LOS}). According to the proportional navigation guidance law [3], the vehicle will then be navigated by commanding a vehicle turning rate (ω_{car}) proportional to ω_{LOS}. The ratio between the turning rate of the missile and the LOS is called the proportional constant (N), represented in Equation 3.5.

$$\omega_{car} = N\omega_{LOS} \tag{3.5}$$

The value of N is always greater than 1 and usually ranges from 2 to 6 [3]. This means that the vehicle will turn faster than the LOS, which allows the vehicle to build up a lead angle with respect to the LOS. This lead angle will put the vehicle on an interception course with the target if the target is not maneuvering and both the target and the vehicle have a constant velocity [3].

Since this constant will determine the turning rate of the vehicle, the choice of N will depend on the physical limits of the vehicle [11]. If the vehicle is only able to turn at a certain angular rate, then it will be impractical to command a turning rate greater than what the vehicle is capable of. The scenario where this is most likely will be when the ω_{LOS} is at a maximum. Therefore, to calculate the maximum value possible for N without exceeding the vehicle's limits, the following adaption of Equation 3.5 can be used:

$$N_{max} = \frac{\omega_{car_{max}}}{\omega_{LOS_{max}}} \tag{3.6}$$

The values $\omega_{car_{max}}$ and $\omega_{LOS_{max}}$ can be obtained by testing the system in a scenario where these values will be a maximum. The scenario in question would be when the target is moving right in front of the main vehicle, with a heading that is perpendicular to the main vehicle's heading. In this scenario, the camera will need to make an extremely fast adjustment to align with the target and the car would then need to make a sharp turn to try and intercept the target. This scenario is depicted in Figure 3.4.

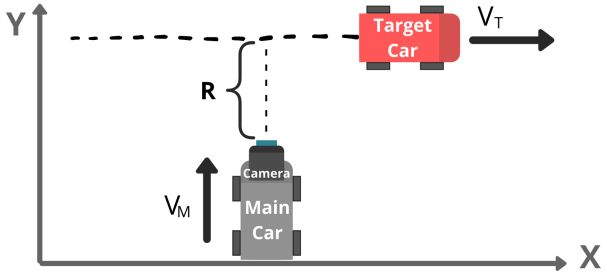


Figure 3.4: Scenario used to obtain maximum turning rates

Equation 3.7 is a general way of calculating ω_{LOS} [3] - with R being the length of the LOS:

$$\omega_{LOS} = \frac{V_M sin(\psi_{LOS} - \psi_M) - V_T sin(\psi_{LOS} - \psi_T)}{R} \tag{3.7}$$

The described scenario will create a heading difference of 90°, which will make the $sin()$ values equal to 1. This simplification is represented by Equation 3.8:

$$\omega_{LOS_{max}} = \frac{V_M - V_T}{R} \tag{3.8}$$

The value for V_M needs to be greater than V_T for the main car to be able to catch up with and intercept the target. These values were chosen to be 5m/s and 2m/s based on the RC car speeds seen in the various literature sources. If we assume R to be a small value of about 0.4m then, according to Equation 3.8, $\omega_{LOS_{max}}$ will equal 7.5rad/s.

To get $\omega_{car_{max}}$, the designed vehicle model is simulated to make a 90° turn at a constant speed of 5m/s. The results of the simulation shows the car model reaching a $\omega_{car_{max}}$ of approximately 15rad/s. Substituting $\omega_{car_{max}}$ and $\omega_{LOS_{max}}$ into Equation 3.6 yields a value of 2 for N_{max}. This is also the lowest value this gain can be [3]; therefore, **N is chosen to be a value of 2**.

3.2.2. Vehicle Turning Rate Control

The control inputs to the vehicle model are V and δ. For proportional navigation to be successful, both the target car and main car need to be moving at constant velocities [3]; therefore, V for both the main vehicle and the target will be a constant value.

The guidance controller interfaces with the vehicle by commanding a steering angle δ needed to induce the desired ω_{car}. Substituting Equation 3.4 into Equation 3.3 creates a non-linear equation that has δ as an input and ω_{car} as an output:

$$\omega_{car} = \frac{V}{l_r} sin(arctan(\frac{l_r}{l_f + l_r} tan(\delta))) \tag{3.9}$$

It would be beneficial to design guidance controller with a linear relationship between ω_{car} and δ. This will allow the controller to calculate a required steering angle by simply multiplying the desired turning rate with a certain constant. To find a linear relationship between ω_{car} and δ, a small-angle approximation can be used to linearize and simplify Equation 3.9 to get the following equation:

$$\omega_{car} = \frac{V}{l_f + l_r} \delta \tag{3.10}$$

This equation can be rearranged to get Equation 3.11 for the steering angle:

$$\delta = \frac{l_f + l_r}{V}\omega_{car} \tag{3.11}$$

Since the vehicle's velocity and dimensions remain constant, Equation 3.11 now describes a linear relationship where the controller can interface with the vehicle by using the desired turning rate to command the necessary steering angle to achieve this turning rate.

The complete guidance controller is shown in Figure 3.5 below:

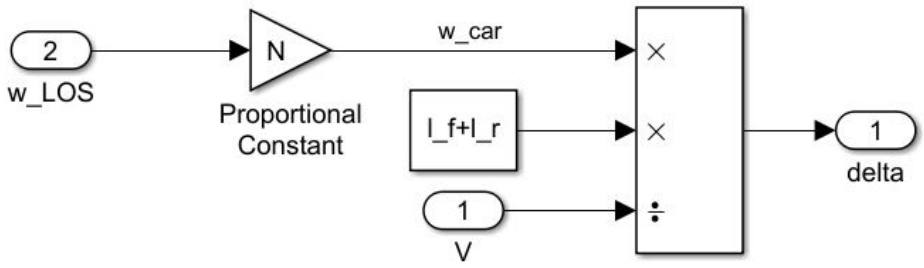


Figure 3.5: Guidance controller

3.3. Camera/Seeker Control

Proportional navigation relies on a seeker to track and aim at a target. As mentioned in the Scope (Chapter 1), the seeker will use vision-based inputs from a camera to track and follow the target. This project will make use of eye-in-hand visual servoing (camera attached to actuator) because of the success this method has had in previous automated RC car projects [6], and for the advantages this method has for fast-moving situations [12].

3.3.1. Camera Controller Design

The camera will need to be independently controlled, therefore it requires its own control system design. The camera video will be processed to obtain the angle between the camera's LOS (ψ_{cam}) and the target. The control system is then required to align the camera's LOS with the target by making the difference between the angle of the actual LOS (ψ_{LOS}) and ψ_{cam} zero (see Figure 3.3 for a visual representation of these values).

A simple PID controller - based on the angle error single between ψ_{LOS} and ψ_{cam} - can be used to control the camera. This controller aims to induce an angular rate which will align the camera's LOS with the actual LOS. For such a simple controller, only the integrator component (I) in the PID is required. The integrator block integrates the ω_{cam} to produce ψ_{cam} for the camera controller. The ω_{cam} value in this controller is used as ω_{LOS} in the guidance control.

The complete camera controller design is shown in Figure 3.6 below:

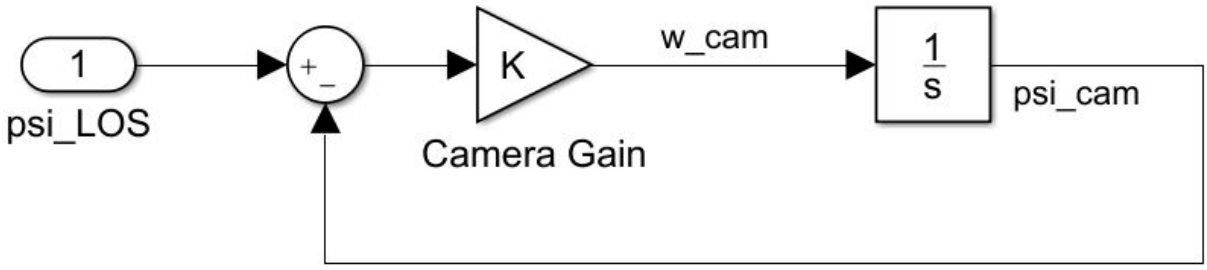


Figure 3.6: Camera/Seeker controller

3.3.2. Camera Gain

The final step in designing this controller is to choose an appropriate camera gain (K). The value of K determines where the poles of this system are. Increasing K moves the closed-loop pole further left on the real axis of a pole-zero plot (see Figure 3.7). The further left the closed-loop is, the faster the response of the system will be; which means that the more K is increased, the faster the control system responds. The control system needs to respond as quickly as possible to keep up with the target and not slow the system down since this controller functions as an inner loop of the entire system. Therefore, the maximum possible value for K is the optimal choice for this gain value.

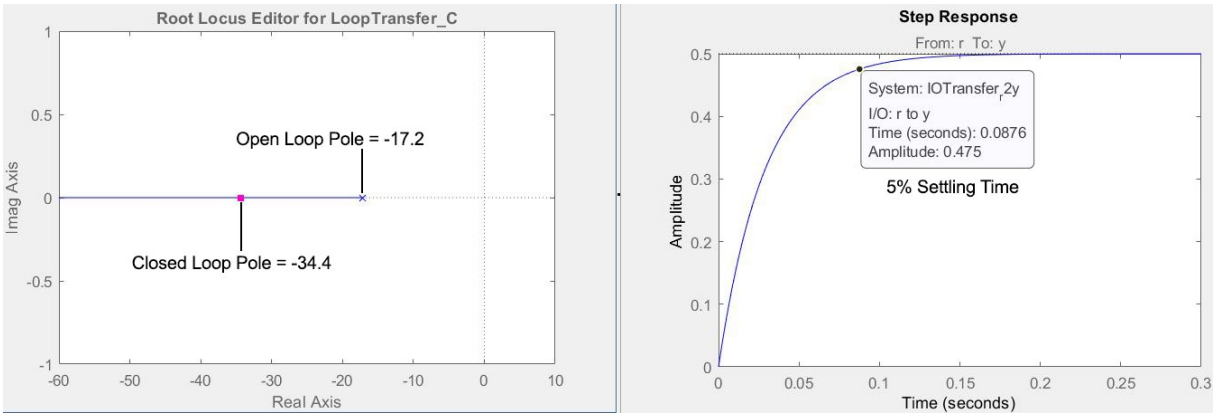


Figure 3.7: Final camera controller root locus and step response (K = 17.2)

Like the proportional constant in the guidance controller (N), the value of K is limited by the physical ability of the camera system. It is impractical to choose a gain which would command a camera turning rate that is faster than what the camera can turn. To assess the limits of the camera system, we would need to look at the physical components used in this system (more information in Chapter 5).

The physical camera system will be made up of a camera and a servo motor which will rotate the camera to aim it towards the target. These two components each have their own limitations:

- The camera has a limited angle of view of about 60° or 1.05 radians [14].
- The servo motor has a maximum turning rate of about 9rad/s [15].

The camera controller induces a turning rate based on the error signal between the LOS and the camera's LOS (ψ_{error}). The maximum turning rate which this control system will command occurs when this error signal is a maximum. Because of the camera's limited angle of view, it is only able to pick up objects which are 0.524 radians (1.05 radians ÷ 2) to the left or right of the centre of its view (see Figure 3.8). This means that the largest value that the controller's error signal could be is 0.524.

The camera gain cannot increase this error signal greater than 9rad/s. Therefore, the following Equation 3.12 can be used to calculate the maximum value possible for the gain K.

$$K_{max} = \frac{\omega_{cam_{max}}}{\psi_{error_{max}}} \tag{3.12}$$

This equation yields a **K value of 17.2**. The step response of the camera controller with this gain value is shown in Figure 3.7. The step response shows an overdamped response with a damping ratio of 1.2. This response has a 5% settling time of approximately 90ms and a rise time of about 65ms.

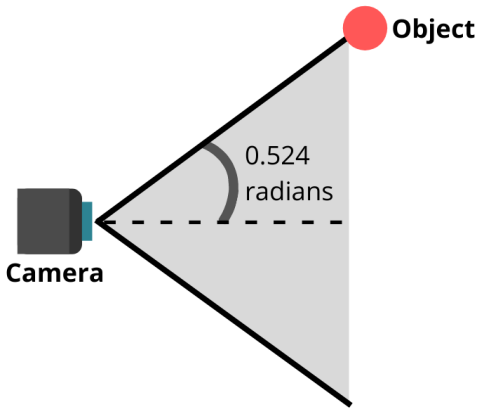


Figure 3.8: Diagram representing the camera's angle of view

Chapter 4

Simulation

This chapter aims to meet the objective of simulating the entire system to validate the model and controllers. The simulation environment is introduced and a variety of simulation tests are described and motivated. The results of each simulation test is also provided.

4.1. Simulation Environment

The designed components from Chapter 3 were combined into a single system in Simulink (shown in Figure 4.1). The *Main Car* block contains the kinematic bicycle model and was created using Equations 3.1 to 3.4. The contents of the *Guidance Controller* block and the *Camera Controller* block are shown in Figure 3.5 and 3.6, respectively. The additional *Target Car* block contains the same equations as the *Main Car* block, with its own control signals. Finally, the *Angle Measurement & Collision Check* block is added to emulate the measurement done by the camera of the angle between the main car and the target. This block also contains the collision detection block which stops the simulation when the main car intercepts the target. The contents of this block are shown in Figure 4.2.

All MATLAB code used to run the simulations can be found in Appendix G.

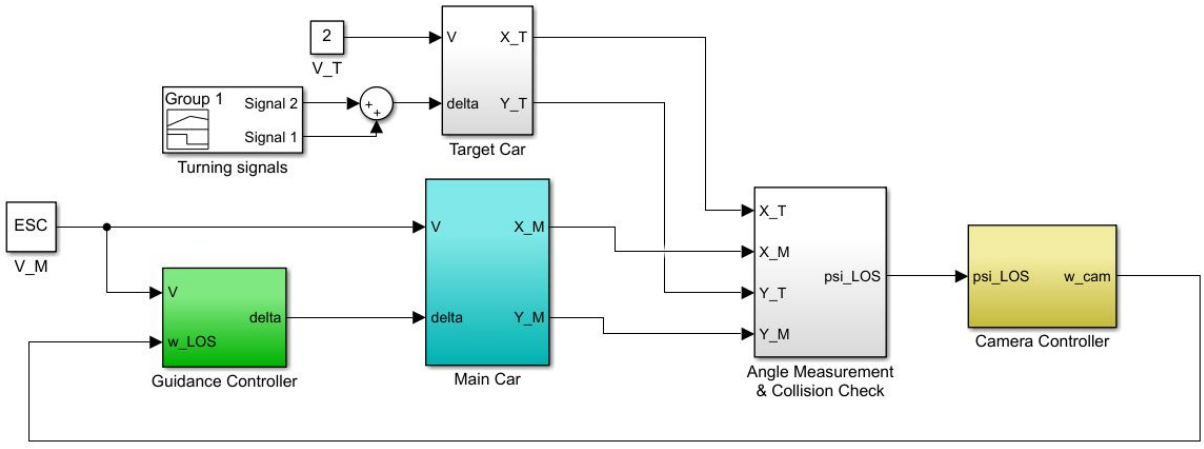
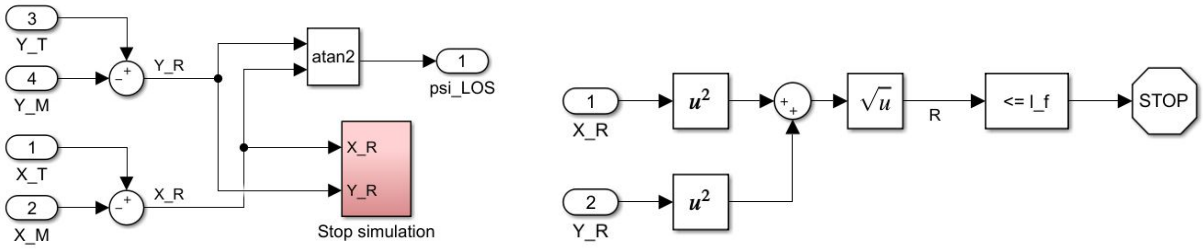


Figure 4.1: Simulation environment



(a) Angle measurement **(b)** Stop simulation

Figure 4.2: Inside '*Angle Measurement & Collision Check*' block

In Sub-Figure 4.2a, an *arctan*2 block is used instead of a standard *arctan* block to avoid angle wrapping when calculating ψ_{LOS}. The collision detection in Sub-Figure 4.2b checks the length of the LOS (R) and stops the simulation once R is the same distance as the front length dimension of the car (l_f). At this distance, it is assumed that the front of the car will have made contact with the target.

4.2. Simulation Results

The simulation environment was used to set up a number of tests to validate the effectiveness of the system and to observe the behaviour of the system under different conditions.

4.2.1. Monte Carlo Simulation

To validate the effectiveness of this system, it was decided to use a Monte Carlo simulation [16]. This test randomizes the starting point and heading direction of the main car without changing the path of the target. The conditions for each Monte Carlo iteration are:

Target

- Starting point at (0,0) on the inertial axes (X,Y).
- Drives directly to the right with a constant heading and constant velocity (3m/s).

Main Car

- Initial X position is randomized between -10 and -5.
- Initial Y position is randomized between -5 and 5.
- Initial heading (ψ_{car}) is randomized between -30° and 30°.
- Velocity remains constant (5m/s).

The values chosen for the initial test conditions were based on a possible real-world test - where the main car and target cannot be too far apart; and the main car cannot be initially rotated too far, otherwise the camera will not see the target.

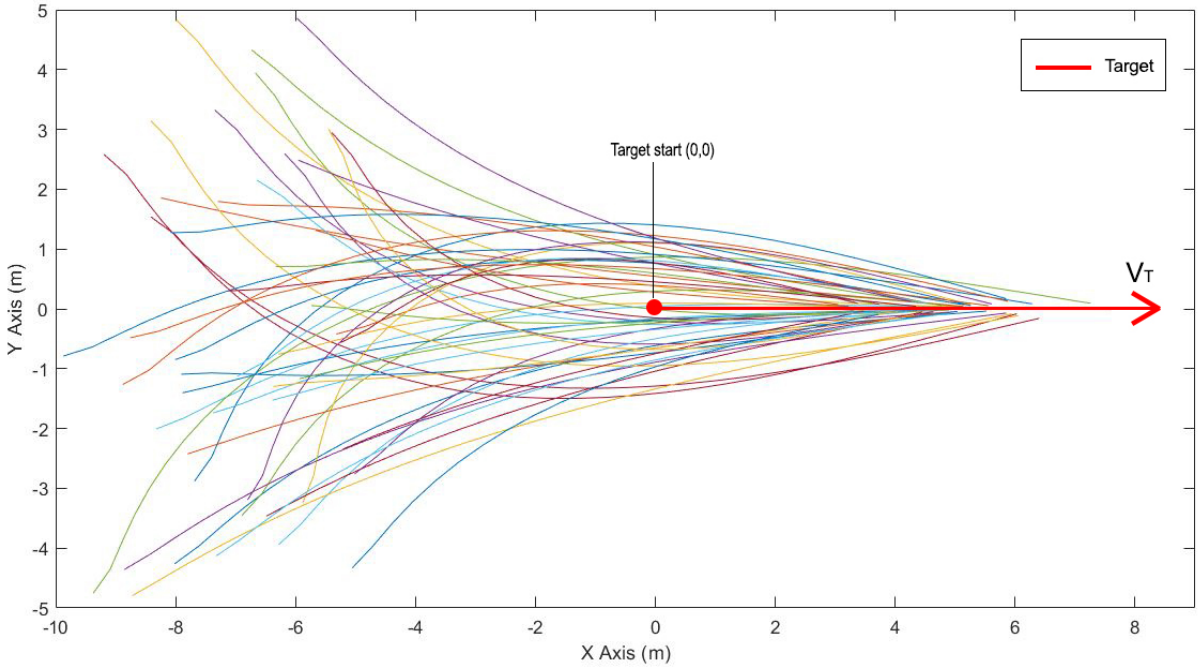


Figure 4.3: Monte Carlo Simulation

Figure 4.3 shows the results of 50 Monte Carlo iterations. The red line represents the path of the target, and the various different coloured lines represent the paths of the main car for each Monte Carlo iteration. It was decided to run 50 simulations because the output became too difficult to interpret when the number of plotted vehicle paths was increased.

From the plot in Figure 4.3, the main car can be seen starting in a random position with a random heading each time before beginning its path towards the target. The camera immediately starts aligning its LOS with the moving target which induces a rotation on the main car and sets the vehicle off on a curved path as it is turning. Once an appropriate lead angle is built up between the main car and the target, the car's path straightens and intercepts the moving target.

It is important to note that the main car was able to successfully intercept the target every time for each of the 50 Monte Carlo iterations.

4.2.2. Individual Tests

To observe the behaviour of the system, a few individual tests were created to assess how the system reacts under specific conditions. The different conditions which are of interest are split into two categories; normal test conditions, and abnormal test conditions. The normal test conditions are scenarios where the proportional navigation system is expected to work, and the abnormal test conditions are scenarios where this guidance system is not generally expected to be as successful [3]. The conditions for each test are:

Normal conditions

- Intercepting a stationary target.
- Intercepting a non-maneuvering target moving at a constant velocity.

Abnormal conditions

- Intercepting a maneuvering target moving at a constant velocity.
- Intercepting a non-maneuvering target accelerating in a random direction.

Intercepting a Stationary Target

Two test conditions were simulated for this scenario to observe the system behaviour. Both tests were performed exactly the same, except the main car's initial conditions where randomized for each test. The results for each test are shown in Figures 4.4 and 4.5.

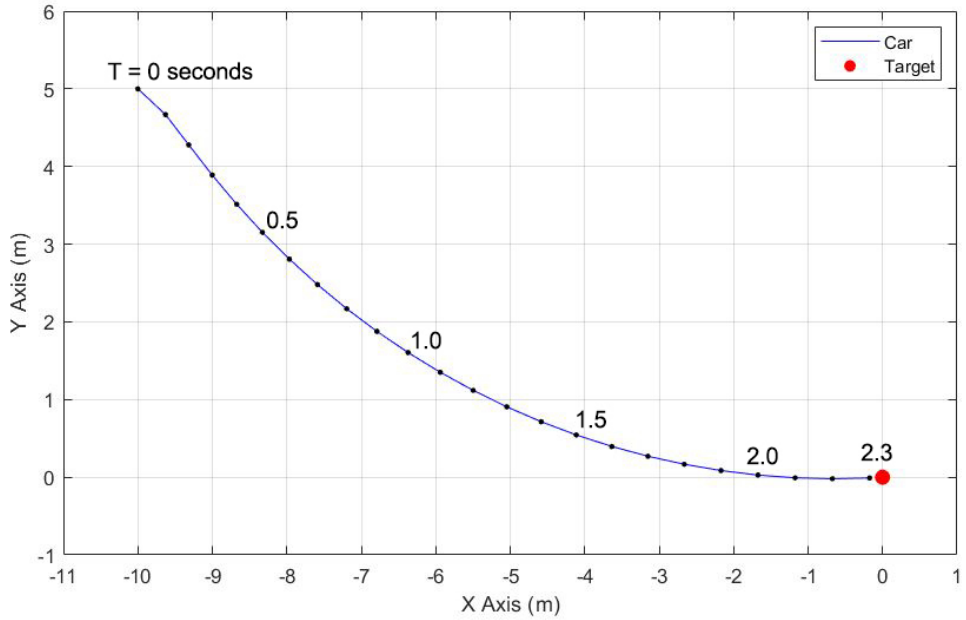


Figure 4.4: Stationary target interception - Test 1

The output of *Test 1* (Figure 4.4) shows the main car turning for most of the simulation before reaching the stationary target. The main vehicle seems to straighten only at the last moment before reaching the target; therefore, most of the simulation time was spent trying to build up a lead angle between the moving car and the stationary target.

The output of *Test 2* (Figure 4.5) shows a more expected vehicle path where the main car turns for just under 0.2s before heading on a roughly straight path to intercept the target at 2.2s (the gap between the final path point and the target represents the distance l_f which triggered the simulation stop condition).

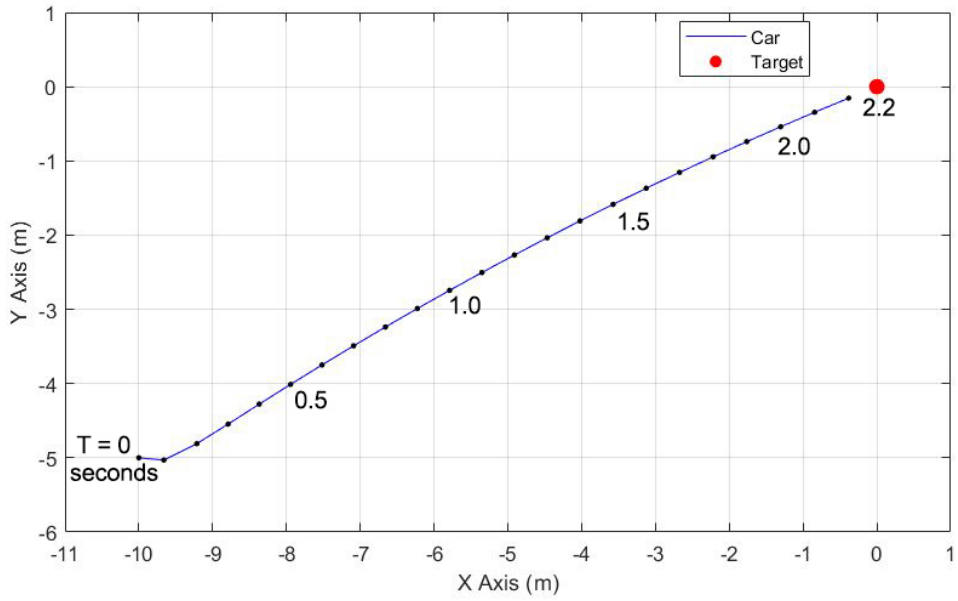


Figure 4.5: Stationary target interception - Test 2

Intercepting a Non-Maneuvering Target Moving at a Constant Velocity

Two test conditions were again simulated for this new scenario. For both tests, the main car was initiated with the same initial conditions. The only change made between the two tests was the heading direction of the target. The first test initializes the target's heading in the same way as the Monte Carlo simulation, and the second test initializes the target with a random heading. The results for each test are shown in Figures 4.6 and 4.7.

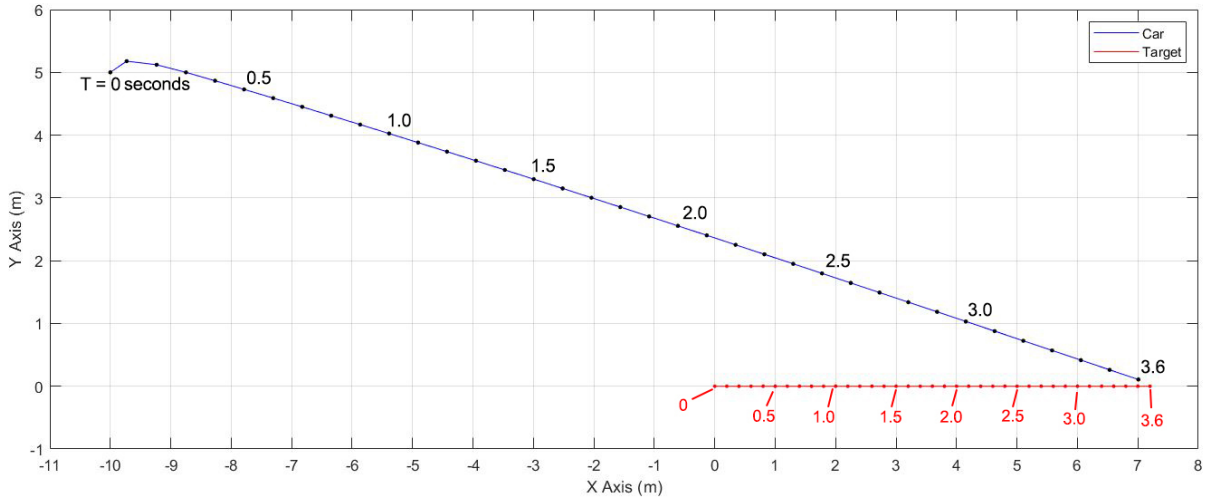


Figure 4.6: Moving target interception - Test 1

The output of *Test 1* (Figure 4.6) indicates that the main vehicle was quickly able to find a straight path of interception within about 0.4s. The vehicle then follows a straight path to successfully intercept its target after 3.6s.

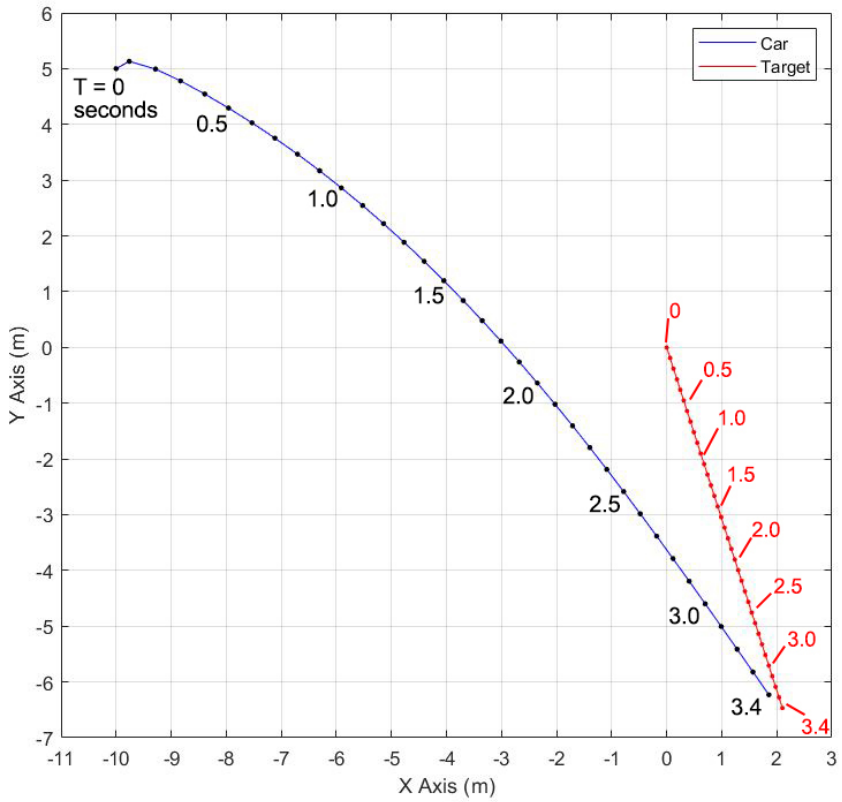


Figure 4.7: Moving target interception - Test 2

The output of *Test 2* (Figure 4.7) shows the main car initially curving towards the target to build a lead angle before straightening out at about 2.4s to successfully intercept the target after 3.4s.

Intercepting a Maneuvering Target Moving at a Constant Velocity

The aim of the proportional navigation is to direct a vehicle to intercept a target by building up a lead angle with respect to the LOS. It is stated that this lead angle will put the vehicle on a collision course *if the target is not maneuvering* and both the vehicle and the target are moving at a constant velocity [3]. The following set of simulations are designed to test the ability of the system under the abnormal condition of a maneuvering target.

A maneuvering target could be a target making sudden heading changes or a target with a constant heading change; therefore, two reasonably different tests were simulated under this abnormal condition to assess the system behaviour. The first test simulates a target which makes two sudden changes of direction. The second test simulates a target which has a constant steering angle and is therefore constantly changing direction. The results for each test are shown in Figures 4.8 and 4.9.

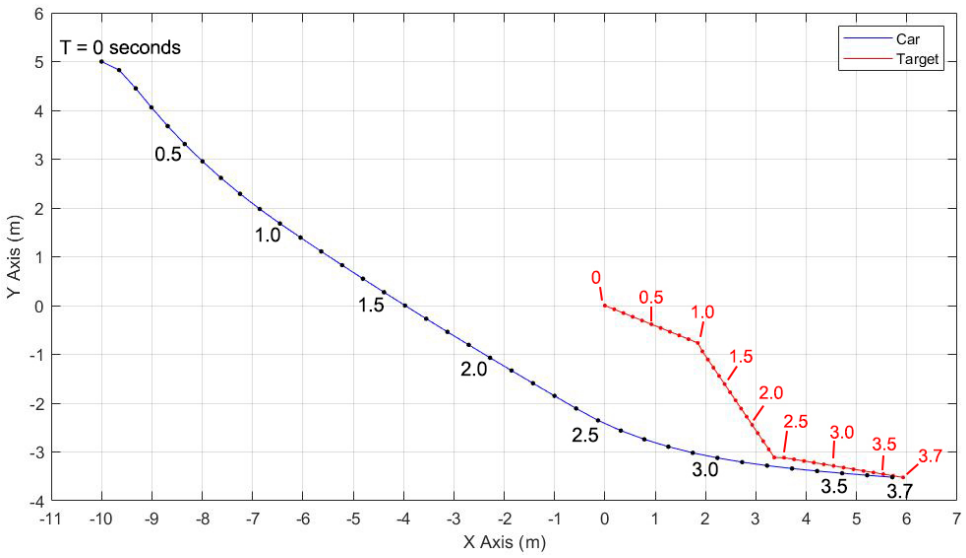


Figure 4.8: Maneuvering target interception - Test 1

The output of *Test 1* (Figure 4.8) shows the main car making changes in its travel path to compensate for the maneuvering target. The car seems to follow a slightly curved path before the target changes its course at 1s. The car then straightens out and starts curving again at about 2.6s before finding the final straight interceptions path for the last 0.5s.

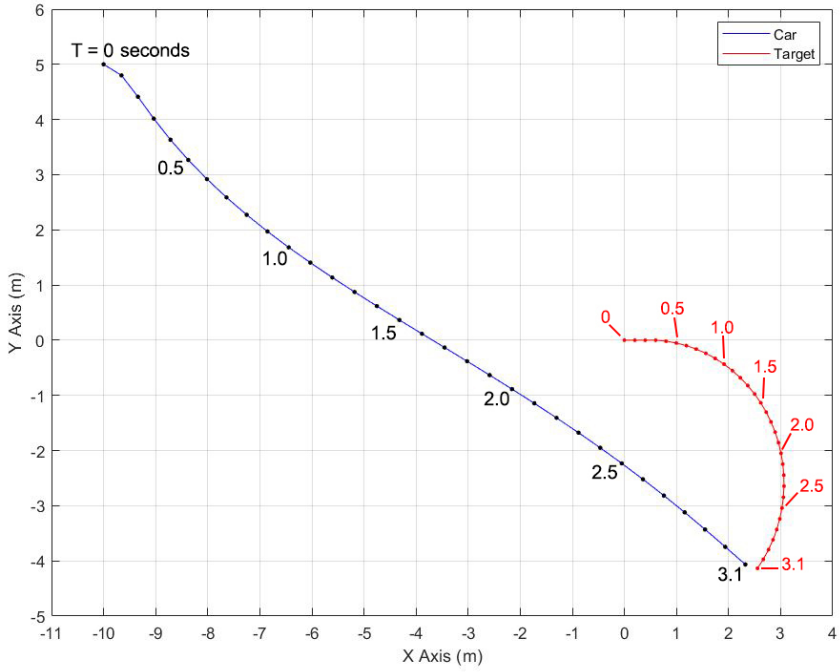


Figure 4.9: Maneuvering target interception - Test 2

The output of *Test 2* (Figure 4.9) shows the main car subtly curving as it tries to adjust for the turning target. Between 1.3s and 2.2s the path seems to straighten before making a final curved adjustment to intercept the target at 3.1s.

Intercepting a Non-Maneuvering Target Accelerating in a Random Direction

Recall that for proportional navigation, the lead angle will put the vehicle on a collision course if the target is not maneuvering and both the vehicle and the target are moving at a *constant velocity* [3]. This final test scenario will assess the behaviour of the system when the target has a changing velocity.

The single experiment tested for this scenario involves a target starting at a speed of 1m/s and accelerating at a rate of 1m/s^2. The results are shown in Figure 4.10.

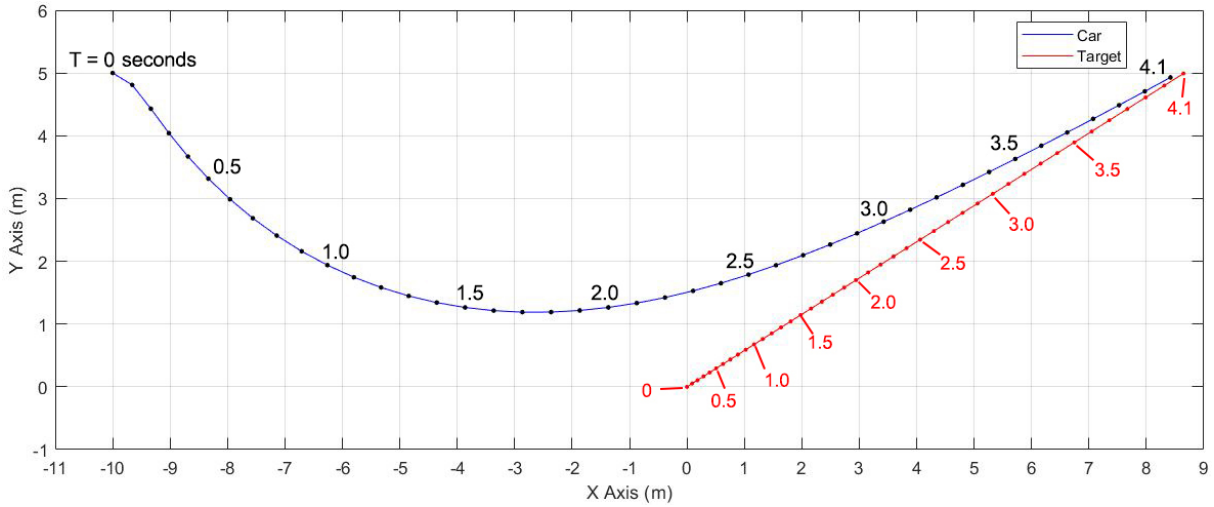


Figure 4.10: Accelerating target interception

The output of this single test (Figure 4.10) shows the car constantly turning to compensate for the accelerating target, making sharp adjustments to its turning rate as the target accelerates. Interestingly, at about 3s, the main car still seems to successfully find a straight interception path to reach its target at 4.1s.

Chapter 5

Physical System Overview

This chapter breaks down the complete physical system into its separate components. Each component is described and its contribution to the system is explained. The choice for each component is also motivated. This overview provides the reader with a clear idea of how the physical system was assembled. The specifications, prices, and suppliers for each part are found in Appendix D.

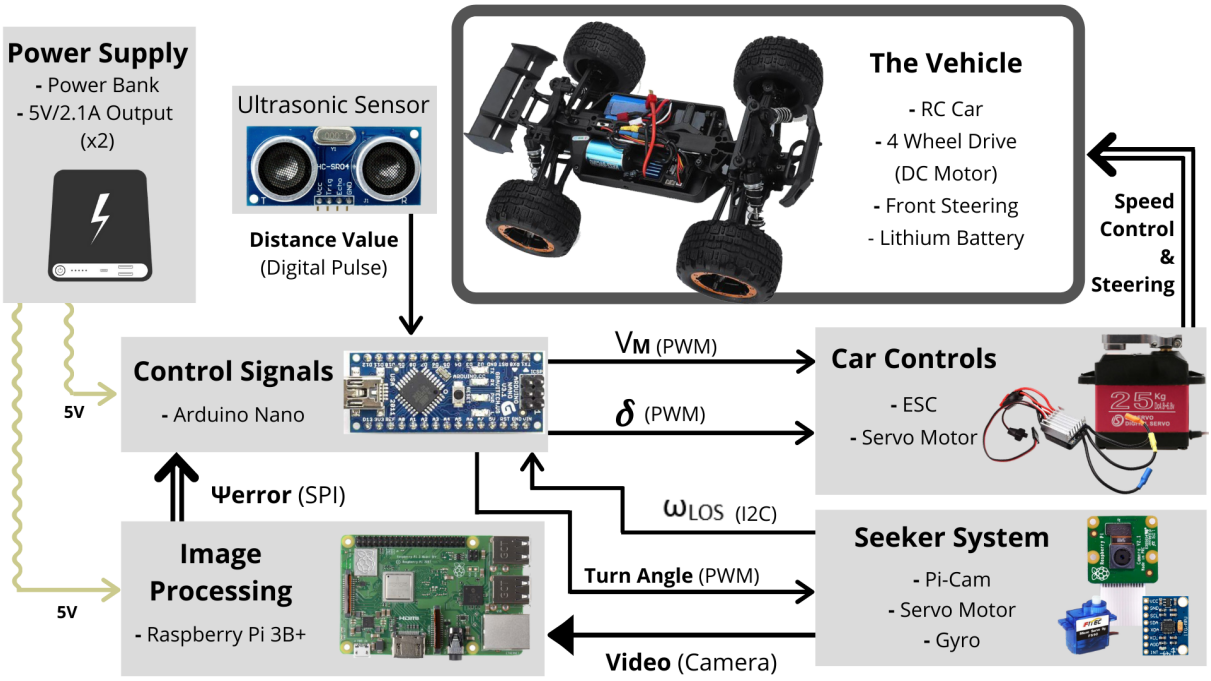


Figure 5.1: Physical system diagram

5.1. The Vehicle

The vehicle of choice for the practical demonstration is the *WLToys Storm-High Speed Buggy* which is a 1/12 size RC car model with four-wheel drive. This RC car has a 540 brush motor and a 20kg.cm steering servo which are both controlled by an ESC receiving server (three-in-one circuit). It is powered by a 7.4V, 1500mAh lithium battery and comes with a remote control.

This RC car was chosen for its size, build quality, maximum speed capability, price, and availability. The RC car needs to be large enough to hold all the extra components of the physical system and durable enough to handle any impact when intercepting a target. The internal components of the car are easily accessible which allows one to replace any components necessary. The RC car also needs to be fast enough to be able to intercept a moving target. The final and most constraining decision factors were the price and availability. RC cars are generally quite expensive and are mostly imported from other countries. This RC car was the best combination of value for money and availability that could be found. The RC car specifications are provided in Appendix D.

5.2. Car Controls

The ESC (Electronic Speed Controller) built into the *Storm Buggy* is unfortunately integrated with the receiver for the provided remote control. This means that the ESC is not directly controllable and can only be controlled via the remote. The steering servo built into the car is a 5-wire servo which is difficult to control with an external microcontroller. It was decided to replace these two components so that the car can be easily controlled with a separate microcontroller.

The *Storm Buggy's* ESC was replaced with a *Hobbywing QUICRUN 1060 Brushed ESC*. This ESC is designed to work with brushed DC motors that are either a size 540 or 550. This means that the chosen ESC will work perfectly with the *Storm Buggy's* 540 brush motor. The final deciding factor for this ESC was again price and availability. The ESC is powered by the RC car's lithium battery and the speed is controlled by sending a PWM signal to the ESC.

The RC car's 25kg.cm steering servo was replaced with the *DS3218* servo motor which has a torque of 25kg.cm and a 3-wire control interface. This servo was chosen as a replacement because it has the same torque and physical dimensions as the original steering servo and is easily controlled with a PWM signal from a microcontroller.

5.3. Seeker System

The vision-based seeker system requires an input, controller and sensor. A Raspberry Pi Camera (Type D) will be responsible for recording the real-time video input. A Raspberry Pi Camera (Pi-Cam) is designed specifically to be used with an R-Pi board, therefore this camera was chosen for its compatibility with the R-Pi board. The Type D Pi-Cam is a good balance between video quality and price. It has a 5MP camera and is less than half the price of the next Pi-Cam upgrade, which comes with an 8MP camera.

The camera will need to be rotated to follow the target, therefore a servo motor is also needed for the seeker system. The Pi-Cam is extremely light, therefore a simple and affordable 1.5kg.cm *FS90* servo that requires a 5V input was chosen. The servo will be controlled with a PWM signal and will be powered by an external power supply.

The turning rate of the seeker system will need to be measured to control the turning rate of the vehicle. This measurement will be done with an *MPU-6050* triple-axis gyroscope (gyro), which also includes a triple-axis accelerometer. This sensor was chosen based on price, availability, and for its user-friendly interface. The chosen gyro has a full-scale range of up to $\pm 2000°/sec$ [17], which is more than sufficient for this application.

5.4. Image Processing

A Raspberry Pi 3B+ will be responsible for the video processing. A Raspberry Pi (R-Pi) was chosen because of the real-time computer vision software available for these boards. The scenario that this physical system will attempt to demonstrate requires quick decision making based on large amounts of real-time video data; therefore, the faster the R-Pi can process the incoming video, the better. However, choosing the latest R-Pi model comes at a high price, so the 3B+ model was chosen to stay within the project budget. The 3B+ is the second fastest R-Pi model available today.

5.5. Control Signals

The components added to the RC car can be controlled using different input signals. These control signals need to be generated by an embedded system based on the theoretical control systems designed in Chapter 3. This embedded system is also required to interface with all the added components using different communication methods.

An Arduino Nano microcontroller was chosen to meet these control requirements. The Arduino Nano can interface with the gyro using I2C and can receive data from the R-Pi via SPI (see Chapter 6). It is also able to control the ESC, servos, and ultrasonic sensor using PWM or digital signals. All the control signals can be calculated using C++ code uploaded onto the board, which is powered by an external 5V source. Arduino is an open-source hardware and software company that has a significant amount of online resources to help beginner hobbyists with their projects [18]. This means that Arduino users have access to thousands of lines of source code from various online resources which makes Arduino the best choice for 'plug-and-play' scenarios. This was the main deciding factor for choosing an Arduino microcontroller. The Arduino Nano board was chosen

because it was the smallest and most affordable Arduino microcontroller available with all the necessary I/O required for the SPI and I2C protocols, and all the PWM and digital signals.

It was also decided to add an *HC-SR04 Ultrasonic Ranging Module* to the car. This ultrasonic sensor is used to measure the distance between the RC car and any object placed in front of the car to check when a collision occurs. This specific ultrasonic sensor was chosen because one was readily available and did not need to be purchased.

5.6. Power Supply

The components added to the RC model car require a power supply separate from the car's lithium batteries (except the added ESC and steering servo). A power bank was selected as an external power supply because power banks generally have stable outputs with over-current protection built into it. Power banks generally have a 5V output, which is well suited for this system since all the extra components require either a 5V or 3.3V power supply.

The current draw of the physical system is a crucial factor when choosing a power supply. Table 5.1 shows the maximum expected current draw from each additional component:

Component	**Current Draw**
Arduino Nano	19mA [19]
FS90 Micro Servo Motor	100mA [15]
HC-SR04 Ultrasonic Ranging Module	15mA [20]
MPU-6050 Gyroscope	3.6mA [17]
Raspberry Pi 3B+ (with Pi-Cam)	1A [21]
Total	**≈ 1.14A**

Table 5.1: Maximum expected current draw from system

The *Romoss Sense4 Mini 10000mAh* power bank was chosen to meet the requirements of the external power supply. The chosen power bank has two 5V outputs which have a total current output of 2.1A. This power bank is more than enough current for the additional components and conveniently provides two separate power supply lines. The Arduino, R-Pi (with attached Pi-Cam), servo motors, and ultrasonic sensor will be directly connected to this power supply and the gyro will be powered via a 3.3V pin from the Arduino. The chosen power bank was very affordable and is also small enough to easily fit on the RC car.

Chapter 6

Hardware Design

This chapter aims to provide more detail to the hardware design choices made and explains the interface between each component. The chapter clarifies the basic functionality of each component and motivates how they contribute to the physical implementation of the designed system.

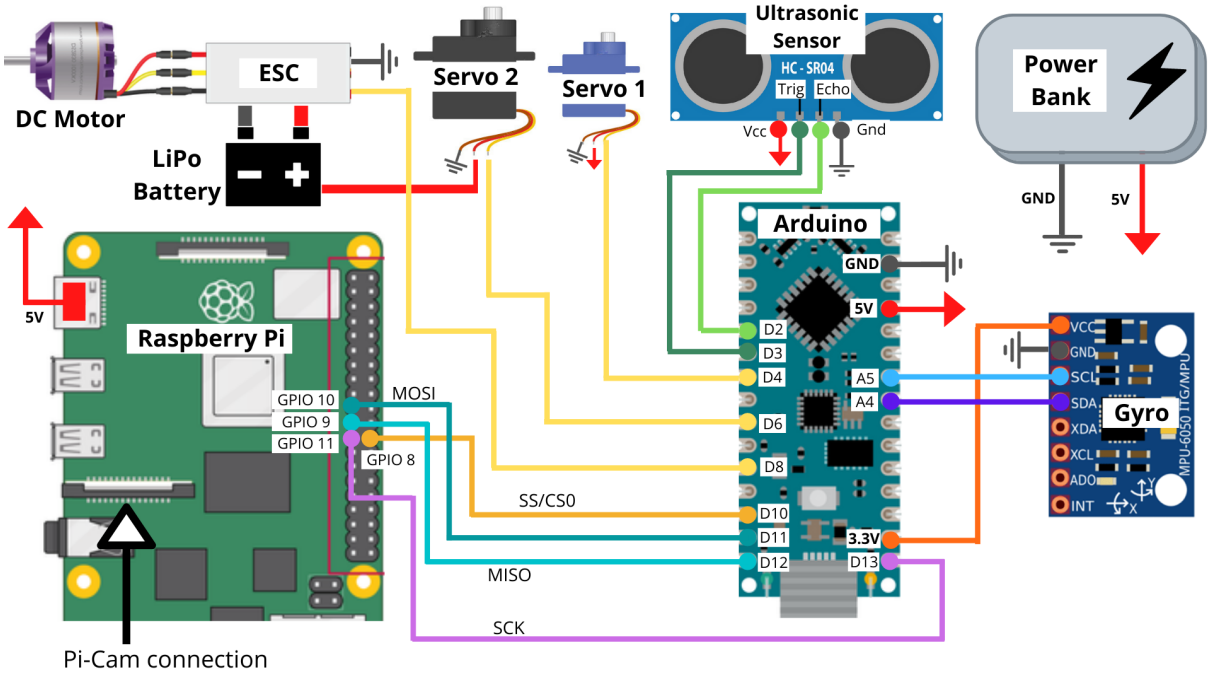


Figure 6.1: Full system pinout diagram

The components are grouped into four categories for this section: *Power Supply*, *Raspberry Pi Interface*, *Arduino Interface*, and *3D Printed Components*. Figure 6.1 provides a full pinout diagram of the entire system which is added to the RC model car. Each pin was chosen based on the Arduino Nano's datasheet [19]. This section will continually refer to Figure 6.1 as each connection and communication method is explained.

6.1. Power Supply

The power bank chosen for this system has two 5V outputs which together provide 2.1A of current. One of these outputs is directly connected to the micro-USB port of the R-Pi. This connection sufficiently powers the entire R-Pi board with the Pi-Cam attached to it.

It was originally planned to power the Arduino in the same way through its mini-B USB port with the other power bank output, but this unfortunately prevents the use of the second power bank output for other components. The Arduino has a 5V output pin, but each pin on the Arduino has a maximum current output of 40mA [19] - which is not enough to power the camera's servo [15]. For this reason, it was decided to solder the second power output to an external stripboard to power the Arduino and camera servo. This allows the servo to draw its necessary current without any limitations caused by the Arduino's pins. This also allows the Arduino, servos and ESC to share a common ground - which is necessary for the PWM control signals sent to the ESC and servos. The gyroscope requires a 3.3V input with a maximum current draw of 3.6mA; therefore, the Arduino's 3.3V pin is perfectly capable of powering this gyro.

As seen in Figure 6.1, the ESC and steering servo are powered by the car's lithium-polymer (LiPo) battery and not the power bank. The DC motor is also powered by the LiPo battery. This was chosen based on the original power connections of the RC car. The ESC and steering servo's ground lines are connected to the shared ground to allow the Ardiuno to send PWM signals to them both.

6.2. Raspberry Pi Interface

The interface between the R-Pi and the Pi-Cam is extremely simple. All that is required is to connect the Pi-Cam cable to the dedicated port shown in Figure 6.1. This connection provides the Pi-Cam with power and allows the Pi-Cam to stream video to the R-Pi.

There are three possible communication protocols which can be used to send data from the R-Pi to the Arduino: UART, SPI, or I2C. The data transfer between these two boards needs to happen as fast as possible since the camera has to follow a target in a moving environment with as much accuracy as possible. Of these three choices, SPI is the fastest protocol. It is also simple to implement and can transmit continuously without interruption [22]. SPI is therefore best suited for the communication requirements and was chosen as the communication protocol between the R-Pi and the Arduino.

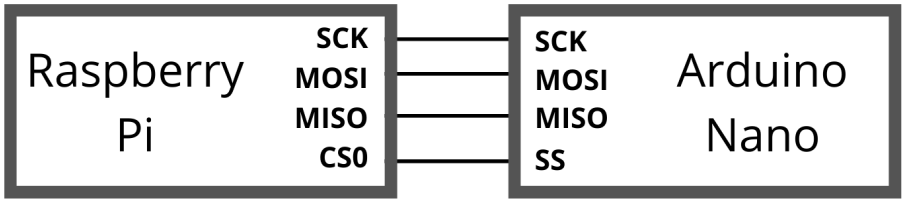


Figure 6.2: SPI connections

SPI data transfer is synchronised with a clock and works on a Master-Slave relationship. The Master device controls the clock and sends or requests data from Slave device [23]. In this system, the R-Pi will act as the Master and will control the clock and send data to the Arduino. Figure 6.2 shows the connections for SPI communication between the two boards and Figure 6.1 shows the pins used for these connections. The four connections are:

- **SCK**: Serial Clock (synchronizes communication between the two boards).
- **MOSI**: Master Output, Slave Input (output from master).
- **MISO**: Master Input, Slave Output (output from slave).
- **SS/CS0**: Slave Select (selects the appropriate slave device) / Chip Select 0 (R-Pi has two slave select lines available: chip 0 and chip 1).

6.3. Arduino Interface

The Arduino Nano is the 'brain' of the system and is responsible for all the control signals sent to each component and for processing all measured data received from each sensor.

Gyroscope

The *MPU-6050* gyro makes use of the I2C communication protocol to transmit its measured values. I2C is a lot simpler than SPI and only requires two wires to transmit data between devices [24]. These wires are:

- **SDA**: Serial Data (allows the master and slave to send and receive data).
- **SCL**: Serial Clock (carries the clock signal).

I2C also has synchronised data transmission based on a shared clock between a Master and a Slave [24]. In this scenario, the Arduino is the Master device and the gyro is the Slave device. The data is transmitted one byte at a time - according to set data frames - to address registers in each Slave device [24]. The simple two pin connection for the gyro to the Arduino is shown in Figure 6.1. The gyro also draws power from the Arduino and is therefore also connected to the 3.3V and ground pins of the Arduino. The default address register accessed by the Arduino during I2C communication is the 0x68 address register of the gyro. This address register is selected by powering the Vcc pin on the gyro instead of the AD0 pin [17] (as seen on Figure 6.1).

Servo Motors and ESC

Both servos and the ESC require a PWM signal to control them. The servos turn to a specific position based on the duty cycle of the PWM signal, and the ESC turns the DC motor at a certain speed based on the duty cycle of the PWM signal. Most microcontrollers require timers to be set up with prescaler values and clock dividers to generate PWM signals with the desired frequency and duty cycle. This is not the case for Arduino microcontrollers which uses simple functions in their code to generate PWM signals. This functionality contributes to the 'plug-and-play' ability of the Arduino Nano.

The two servos are directly connected to the external stripboard to receive the required current from the power bank's 5V supply. The third wire on each servo is connected to a pin on the Arduino to received their PWM control signal. The ESC is powered by the RC car's LiPo battery, therefore it only requires a single pin connection to the Arduino to receive its PWM control signal.

Ultrasonic Sensor

The sensor measures distance by sending out a burst of ultrasound at 40kHZ which is then echoed back to the sensor after bouncing off any object in front. The sensor then outputs a signal that represents this received echo value. To initiate the ultrasound burst, a digital pulse of 10μs is sent to the sensor's 'trigger' pin. The signal on the sensor's 'echo' pin is then read and converted to get a distance value. These two pins connect to two digital pins on the Arduino (see Figure 6.1) and the sensor is powered by the 5V supply.

6.4. 3D Printed Parts

The original RC car body was not designed to house any extra components added to the system. The additional components, therefore, require suitable structures to securely hold them in place on the RC car. The Pi-Cam also has no structure to help attach it to the servo motor that will be rotating it.

To solve these issues, 3D structures were designed to house the extra components in such a way that they can be easily attached to the car's main structure and to provide the camera with a holder which attaches to the servo. This will prevent them from falling off as the car moves around and it also stabilizes the components during a collision with the target. The 3D parts were designed in *Autodesk Inventor* and were 3D printed.

These designs and their applications are shown in Appendix E.

Chapter 7

Software Design

This chapter specifies the software objectives and explains how the software was designed to meet these objectives. The coding environment of each board is described and the software design of each board is provided separately. The structure of the code which controls the entire system is represented by the code flow diagram in Figure 7.1 and the descriptions in this chapter can be referred back to this figure. All code used in this project can be found in Appendix G.

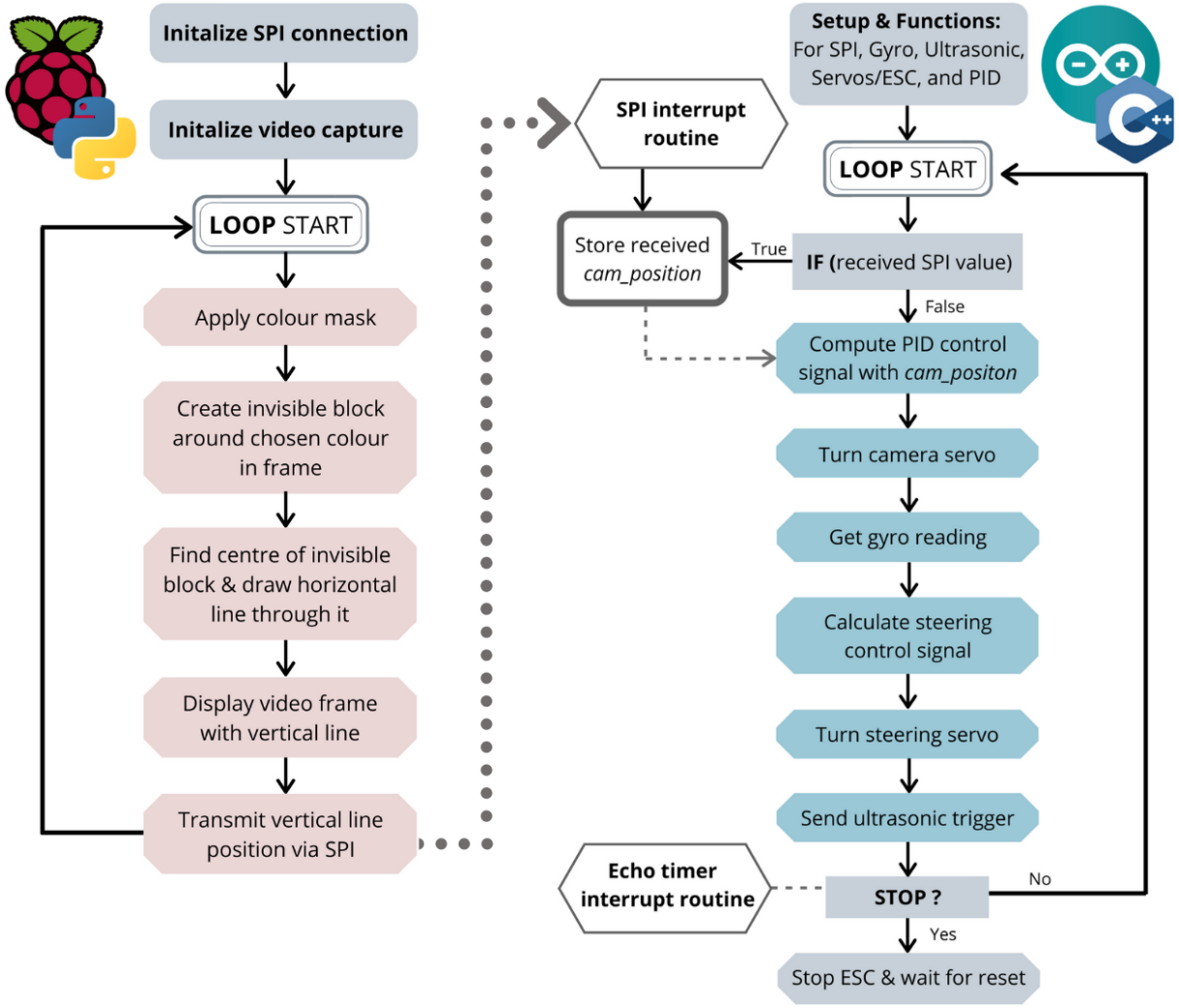


Figure 7.1: Code flow diagram

7.1. Raspberry Pi Software

All Raspberry Pi boards run a Linux based operating system (*Raspberry Pi OS*) and support a variety of coding languages. It was decided to use Python to code the R-Pi software and this decision was based on experience previously gained from coding with Python, and for the specific library packages Python has available to it.

The R-Pi software objectives are mainly based on the need for image processing of real-time camera footage to detect an object. The simplest method available to detect an object under the conditions of this project is colour detection. OpenCV is a library of programming functions which is specifically designed for real-time computer vision applications and also has colour detection functionality. This library is open-source with a large user community that provides code online from a range of vision-based projects. This library was used in the R-Pi software design as well as the Numpy library for handling data arrays and other mathematical operations. The Spidev library was also needed for SPI communication.

The objectives of the R-Pi software are:

- Stream video input and process incoming image frames.
- Identify a specified colour in a captured video frame.
- Get the horizontal pixel-distance from the colour-object to the centre of the frame.
- Transmit necessary values to Arduino Nano using SPI.

First, the R-Pi's camera module is enabled and a *picamera* package is installed to use the attached Pi-Cam. The Pi-Cam's video is then captured and processed frame-by-frame using OpenCV code [25]. A specified colour mask can then be created to overlay onto the captured frame which causes the software to only process a certain range of colours that passes through the applied mask. A simple colour detection script was found which allows the user to adjust a few sliders until the output of the video capture only shows the desired colour [26]. This code was used to select an appropriate mask for the colour detection of the software.

With a colour mask applied to each frame, OpenCV's *findContours* function finds a curve that joins all the continuous points of the same colour intensity along an object's boundary. OpenCV can then be used to create an invisible block around this boundary and the centre of this block can be calculated. This code functions as the object detection based on a specified colour. A line can be drawn onscreen to indicate the centre of the identified

object on the frame. The value of the object's position is then sent to the Arduino via SPI as *cam_position*. The code used for this colour-based object detection was found and adapted to suit the needs of the controller [27]. Figure 7.2 shows the onscreen output of the code.

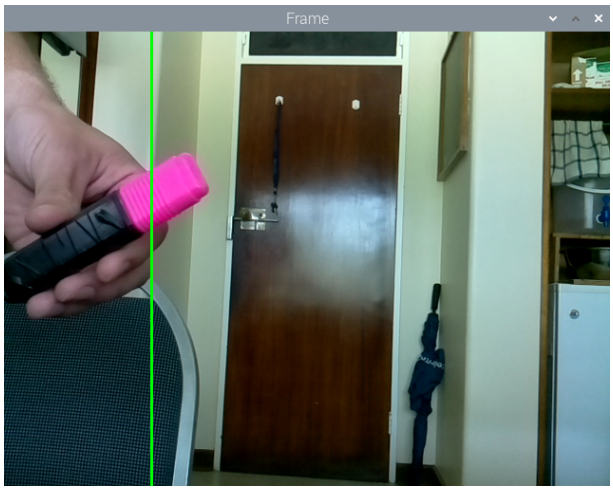


Figure 7.2: Onscreen output for pink object detection

The pixel width of each captured frame is set to a width of 180, causing each x-position on the frame to represent a number from 0 to 180. This is done to represent an angle from 0 to 180 degrees which can be sent to the camera's servo motor. ψ_{LOS} is represented by the centre position of the frame and is permanently set to a value of 90. ψ_{cam} is set to the position of the detected object's centre in the frame and the ψ_{error} angle is then represented by the difference between the centre position and the object position.

The Spidev package is very simple to implement for SPI communication. The R-Pi's SPI communication just needs to be enabled and then the SPI bus is opened at a certain transmission rate. The Slave device's system clock needs to be at least 4 times this SPI transmission rate [23]. The Arduino Nano has a system clock speed of 16MHz [19], which means the transfer rate can be set at a maximum of 4MHz. The transfer rate was set to 3MHz to allow a fast transfer rate without the risk of exceeding the limits of the communication protocol. After the SPI bus has been set up, any data from the R-Pi can be transmitted with a single line of code for a pink object.

7.2. Arduino Nano Software

All Arduino boards come with a dedicated Arduino IDE which allow users to code in either C or C++. Most of the source code which was needed for this project was coded in C++, therefore it was decided to use C++ for the entire Arduino software design.

The main objectives of the Arduino software are to interface with and control the additional system components based on functions which emulate the theoretical controllers. The software is also required to receive SPI inputs from the R-Pi. The design of the Arduino software is explained separately for each component in this section.

SPI Communication

To allow the microcontroller to receive SPI inputs, the Arduino's *SPI* library needs to be included in the software design. This library provides all the necessary functions for SPI communication. A basic implementation of how this library is used to receive SPI inputs was found and adapted for this project [28]. This code simply initializes the SPI communication and then sets one of the Arduino's pins for the necessary data transfer. An SPI interrupt-routine function is then created to store any received values continuously without interfering with the flow of the code's main loop. When a value is transmitted to the Arduino, this value is stored and a flag is set to indicate that data has been received.

I2C Communication and Gyroscope Readings

An all-inclusive library was found to control the I2C software communication with the gyroscope [29]. This library was created by Adafruit Industries and it includes the Arduino's *Wire* library needed for I2C. This library was used to create and initialize a *gyro_sensor* struct variable. A *getGyro* function was then created to call values from this struct variable which are directly read from the sensor via I2C. For this function to work effectively, the software was designed to call the *getGyro* function only every 100ms. This prevents the gyroscope's data buffers from overflowing and the I2C communication from causing a time-out in the software.

Servo and ESC Control

The Arduino's *Servo* library is used to generate PWM signals for the two servo motors and the ESC. With this library, the control of the servos and ESC is done by simply attaching each PWM output to the appropriate pin and then writing a value from 0 to 180 to that pin. This value sets the duty cycle of the generated PWM signal and it represents the angle in degrees at which to turn a servo motor. Consequently, the theoretical controller values are converted from radians to degrees to successfully implement the control system in the code. For the ESC, the value written to the pin it is attached to corresponds to a specific speed at which the DC motor will turn.

Implementation of Controllers

The main objective of the Arduino software design is to implement each controller in software. The vehicle turning rate controller is implemented by simply using Equations 3.5 and 3.11 in the code and converting the output from radians to degrees to get the required δ value to steer the car with the added servo motor. The input turning rate to this controller is the measured angular rate from the camera's gyroscope. Since the camera is attached to the car, this gyro reading provides a LOS turning rate relative the inertial axes (X, Y).

The camera controller was implemented using an Ardunio *PID* library; which accepts a given value, calculates the error signal from a specified setpoint value, then provides the optimal output for the controller. Only integral control was used for this controller, therefore the *PID* library's source and header file had to be altered to implement the necessary controller functionality. The alterations to the code included discretizing the camera controller using the bilinear transform and implementing this result in the code. The discretized formula is shown in Equation 7.1.

$$control[n] = \frac{K.T_{sampling}}{2}(error[n] + error[n-1]) + control[n-1] \tag{7.1}$$

The discretized camera controller is implemented by calling a *pid.compute* function with *cam_position* as an argument (received from R-Pi) to calculate the turning angle needed for the camera's servo motor to align the centre of the object with the camera's centre. The camera gain K was converted to a value suitable for degrees (multiplied by $\pi/180$) to produce **a new K value of 0.3**.

Stop Condition

Once all the control objectives are met, the final requirement for the Arduino is to stop the system once a certain condition has been met. This condition occurs when the RC car collides with its target and it is tested by checking the distance reading of the ultrasonic sensor. The code for this sensor starts by initializing the 'trigger' and 'echo' pins as an output and an input respectively. A 10us digital pulse is then sent every 60ms (recommended by datasheet [20]) in the main loop to initiate the ultrasonic burst. An interrupt routine measures the time between each level change of the echo pin to calculate the duration of the echo signal without stalling the main loop. The duration value is converted using a formula from the datasheet [20] to get the distance value in centimetres. This value is compared with a reference distance and the system is stopped if the received value is smaller or equal to the reference value.

Chapter 8

Practical Results

This chapter describes the practical tests done to validate each physical controller. The complete integrated system is tested and the results are provided. The chapter shows how the physical demonstrator was able to meet all the physical project objectives.

8.1. Camera Controller

The camera controller was tested by measuring how fast the camera system responds to a detected object. As described in Chapter 7, the practical camera controller tries to reduce the difference between the object's centre position and the video frame's centre position to zero. The video frame's centre is a constant value of 90, therefore the controller tries to set the object's centre (i.e. *cam_position*) to 90.

The test was done by initially placing an object of a specified colour at the edge of the camera's viewing range (*cam_position* = 180), then recording the *cam_position* variable over time. A controller gain of K = 0.3 was used in this test (motivated in Chapter 7) with a sampling time of 0.01s. The results are shown in Figure 8.1.

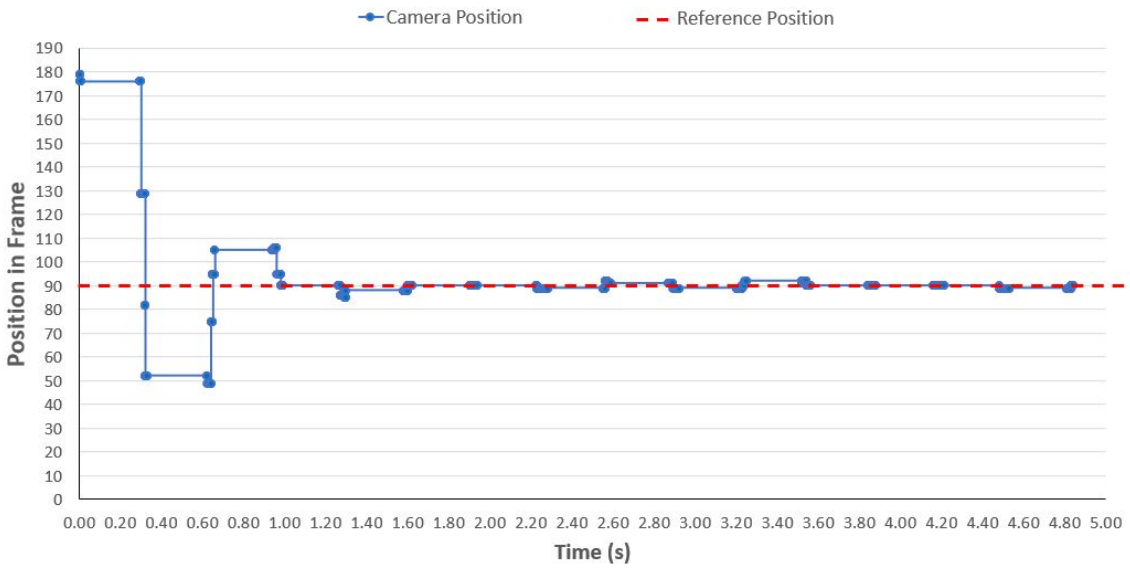


Figure 8.1: Response of camera controller

The measured output shows an underdamped response with a 5% settling time of about 1s and a rise time of approximately 0.3s. These measured values yield a natural frequency of 6 rad/s and a damping ratio of 0.5. The observed results vary from the theoretical controller which has an overdamped response (1.2 damping ratio) with a 5% settling time of about 90ms and a rise time of 65ms (Figure 3.7). The differences between these two responses are discussed in Chapter 9. Regardless of the unexpected characteristics in the measured response, the test was still able to successfully show that the practical camera controller can align the camera with a detected object.

8.2. Turning Rate Controller

The turning rate controller was tested by commanding a specified turning rate and measuring the gyroscope reading. This test was done with a vehicle speed of about 5m/s. The gyro readings were recorded in the same way as the previous practical test and the results are shown in Figure 8.2.

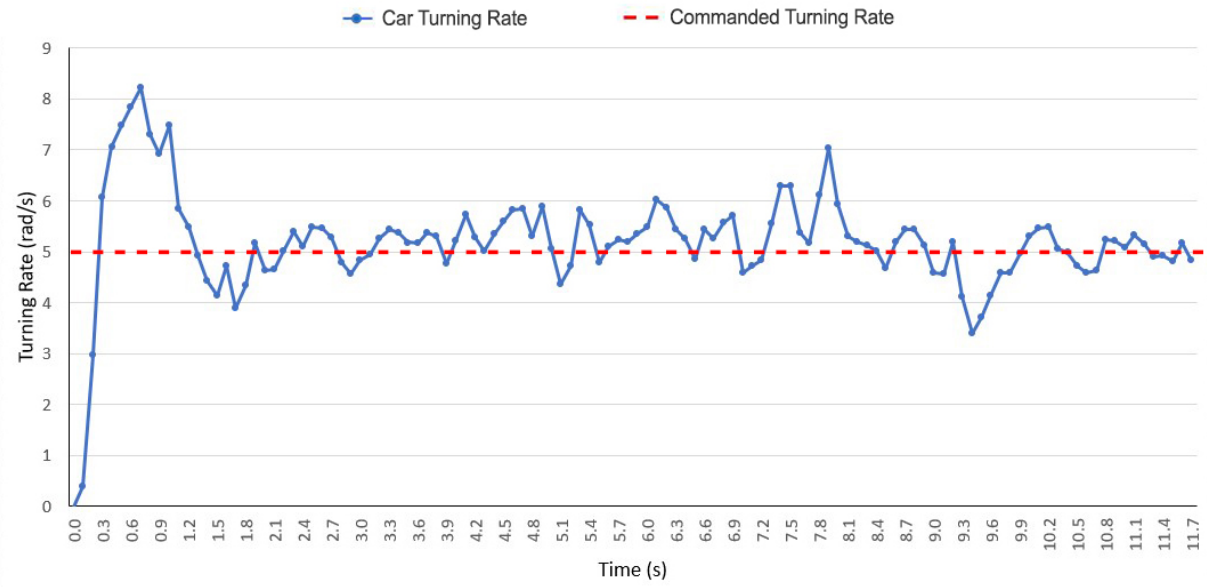


Figure 8.2: Response of turning rate controller

The output readings of the gyroscope are quite noisy and never settle at a constant value. The response has an initial spike of about 8 rad/s before averaging out around a turning rate of 5 rad/s after approximately 1.5s. Small and irregular spikes are observed throughout the response which is caused by the noise on the gyro sensor. The observed turning rate of the physical demonstrator appeared to be constant despite the irregularity in the measured results. Therefore, it is assumed that this constant turning rate was approximately 5 rad/s according to these measurements. This test successfully validates the controller's ability to induce a desired turning rate on the vehicle.

To achieve this desired turning rate, the proportional gain (N) in the turning rate controller had to be increased to a much higher value than originally designed. This change in the proportional gain greatly impacted the final integrated tests.

8.3. Integrated Tests

The entire system was assembled and tested as a whole. The complete build of the physical demonstrator is shown in Appendix F. The initial build was far too delicate for colliding with a target, therefore, the system was rebuilt with the original car frame to strengthen it. The target used in the integrated tests was a pink, spray-painted basketball which was rolled in front of the RC car.

The integrated tests were performed with different values of N. The choice of each N value is stated and motivated for each test and the results thereof are described. Video recordings were taken of each test to document the results. The link to these videos are provided below, with some extra footage which documents the functionality of the controllers:

https://drive.google.com/drive/folders/1MEz-HrGYrHiVw1pMtdoo6MSMEuTXe_TG?usp=sharing

For the sake of this report, frame-by-frame screenshots were taken of each video to aid in the description of the integrated test results. The frame-by-frame images are shown in Appendix C with appropriate descriptions for each frame.

According to the tests done on the turning rate controller, the proportional gain was initially set to a maximum value of 6 for the first integrated test with an RC car speed of approximately 5m/s. This gain proved to be far too large and resulted in driving the car straight into a wall on the first test. This crash broke the back shaft supporting the car's frame, disconnected the steering servo from the front wheels, and broke one of the gears inside the camera's servo which caused the camera to occasionally slip as it rotated. These structural issues limited the number of integrated tests, yet a few successful tests were still carried out despite the reoccurring problems caused by the crash.

The result of this first integrated test seems to explain the initial spike observed in the turning rate controller's response (Figure 8.2). The turning rate controller appears to command an instantaneous turning rate which does not apply over time. This deduction was taken into account for the further integrated tests.

The second integrated test was done with an N value of 4 and an RC car speed 3m/s. The results of this test were vastly better than the first test. The car managed to track and follow the target before prematurely attempting to intercept it. The car then missed the target, but was able to turn around and find the target after it had stopped moving. Interestingly, the path taken by the RC car to intercept the stationary target seemed to mimic the path shown in *Test 1* of the stationary target individual test (Figure 4.4).

To prevent the car from prematurely intercepting its target, the value of N was decreased to a value of 3 for the third integrated test. This test proved to be successful. The car was able to track and intercept the moving target.

A final integrated test was done with an N value of 2. The result of this test showed the car following a 'pure-pursuit' trajectory to its target. Pure-pursuit occurs when the turning rate of the vehicle is not fast enough, causing the car to follow the target until a collision occurs instead of building up a lead angle to intercept the target [3].

One of the greatest challenges faced throughout the integrated tests was a scenario where the car maneuvered too quickly, causing the target to move out of the camera's angle of view. This caused the seeker system to lose its target and prevented the physical demonstrator from tracking and intercepting the target. To reduce the likelihood of this scenario happening, the camera gain (K) was increased from 0.3 to 0.35.

Chapter 9

Conclusions and Recommendations

This final chapter summarizes the project and discusses conclusions based on the observed results from the theoretical and practical tests. Recommendations for future work on this project topic are also provided.

9.1. Summary and Conclusions

This project aimed to design a system which applies the guidance law of proportional navigation to autonomously control an RC model car to track and intercept a moving target. The system was initially implemented using mathematical models and simulations before creating a physical demonstrator of the designed system.

The results of the simulations (Chapter 4) prove that the designed proportional navigation system is successful in autonomously guiding a theoretical vehicle to intercept a moving target. The Monte Carlo simulation results (Figure 4.3) show that the system is able to do this regardless of the initial conditions - which were based on physical limitations of a real-world scenario. The control system even exceeded expectations by showing that it is capable of intercepting targets under abnormal test conditions (Figures 4.8 - 4.10).

Various vehicle paths plotted by the Monte Carlo simulation do not behave entirely as expected, even though target-interception was successful. An optimal vehicle path is one where the vehicle initially follows a curved path before heading straight towards the point of interception with the target. Some of the vehicle paths in the Monte Carlo simulation did not behave like this and instead curved for most of its journey towards the target. This same behaviour is seen in Figure 4.4 of the first individual test simulating a stationary target.

This behaviour is caused by the known physical constraints designed into the system. Proportional navigation was originally designed for missiles, which can manoeuvre freely in the air with few limitations on its movement. Missiles are also generally quite far away from their targets because their seeker system can hone in on targets which are

kilometres away. This provides ample time for the missile to finish its curved path to generate an appropriate lead angle for interception. However, an RC car is unfortunately limited by the maximum steering angle of its wheels as well as its vision-based seeker system. The limited steering angle prevents the car from making any extreme manoeuvres towards its target, and the vision-based seeker system is only able to hone in on targets which are a few meters away. The steering limitation increases the time it takes the RC car to create a lead angle with respect to the target, and the seeker limitation decreases the time the RC car has to create this lead angle. Unfortunately, these limitations can not be avoided due to the nature of how cars are built and how they move. Regardless of these limitations, the system still meets the main objective of being able to guide and control an RC car to autonomously intercept a target.

The results of the practical test (Chapter 8) show that the physical demonstrator is also able to autonomously track and intercept a moving target. This was done using practical gain values similar to the theoretically design gain values ($N_{designed} = 2$, $N_{actual} = 3$; $K_{designed} = 0.3$, $K_{actual} = 0.35$). These results validate the implementation of the controllers and provide a visual, real-world representation of the designed system.

The design of the camera controller was based on the outputs and inputs of the seeker system and not on a physical model representing the size and mass of the actual camera. This meant that any external or inertial forces were ignored. This over-simplification of the physical camera is most likely to be the cause of the unexpected response shown in Figure 8.1. This was, however, not a problem since the aim of the controller was to be functional, not accurate.

The commanded steering angle (δ) required to induce a desired vehicle turning rate is dependent on the proportional constant (N) and the vehicle's velocity (see Figure 3.5). Due to issues in the physical demonstrator's structure, the RC car's velocity was limited to 3m/s instead of the 5m/s velocity used in the design of the system. Had the physical demonstrator been able to operate at the designed velocity, the actual value of N would have most likely been closer to the designed N value of 2. This supports the success of the system design in the implementation of the physical demonstrator.

The results of this report show that the project was able to successfully meet all the project objectives defined in Chapter 1 - with a few inefficiencies in the system caused by physical limitations.

9.2. Recommendations for Future Work

Additional in-depth simulations could be created to fully test the capabilities of the system. The described abnormal conditions in Sub-Section 4.2.2 could be combined to test the limits of the system more extensively. Signal noise could also be added to the simulated models and controllers to improve the validity of the simulations. It is recommended for future work to use a mathematical vehicle model with greater detail as many of the assumptions made for the kinematic bicycle model (Section 3.1) omit forces and effects on the vehicle which could have greatly altered the simulation results.

The compromises in the structural integrity of the physical demonstrator caused by the crash in the initial integrated test (Section 8.3) posed many challenges in the testing of the physical system and prevented any further tests. It is recommended to create a built system which is sturdy enough to handle head-on collisions which will allow the system is keep performing successfully even after a crash. The limitations created by these structural failures meant that the vehicle speed had to be decreased from 5m/s to 3m/s. Without a compromised physical structure, a speed of 5m/s would be possible which could potentially result in requiring a proportional gain (N) value of 2 to command the necessary turning rate for the car to intercept its target.

It is recommended to test the physical system using another RC car as the target. This will allow for integrated tests with a target that can maneuver and accelerate. These tests will validate whether or not the practical system can track and intercept a target under the abnormal test conditions (described in Sub-Section 4.2.2) as successfully as the simulated system.

The greatest limitation to the entire design was the seeker system. Vision-based seekers have a limited angle of view as well as a limited range of view. Designing and building a seeker system which does not have these limitations would be a great benefit to the overall design.

The design of this system can be taken much further. Proportional navigation guides real missiles, therefore this system could be improved to guide real cars. This would require a vehicle model with more mathematical complexity and detail. A more complex vehicle model would require a more complex control system which would interface with a car engine and gears instead of a DC motor and servo motors. The seeker system should also be upgraded to match the capabilities of a missile seeker system.

Bibliography

[1] P. Polack, F. Altche, B. d'Andrea Novel, and A. de La Fortelle, "The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?" in *2017 IEEE Intelligent Vehicles Symposium (IV)*. Los Angeles, CA, USA: IEEE, Jun. 2017, pp. 812–818. [Online]. Available: http://ieeexplore.ieee.org/document/7995816/

[2] "Proportional navigation," Sep. 2020, page Version ID: 976441173. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Proportional_navigation&oldid=976441173

[3] J. H. Blakelock, *Automatic control of aircraft and missiles*, 2nd ed. New York: Wiley, 1991.

[4] K. Bimbraw, "Autonomous Cars: Past, Present and Future - A Review of the Developments in the Last Century, the Present Scenario and the Expected Future of Autonomous Vehicle Technology:," in *Proceedings of the 12th International Conference on Informatics in Control, Automation and Robotics*. Colmar, Alsace, France: SCITEPRESS - Science and and Technology Publications, 2015, pp. 191–198. [Online]. Available: http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0005540501910198

[5] "Self-Driving Cars 101 | Union of Concerned Scientists." [Online]. Available: https://www.ucsusa.org/resources/self-driving-cars-101

[6] "Self Driving RC Car," Jul. 2015. [Online]. Available: https://zhengludwig.wordpress.com/projects/self-driving-rc-car/

[7] J. Liu, P. Jayakumar, J. L. Stein, and T. Ersal, "A Multi-Stage Optimization Formulation for MPC-Based Obstacle Avoidance in Autonomous Vehicles Using a LIDAR Sensor," in *ASME 2014 Dynamic Systems and Control Conference, Volume 2*. San Antonio, Texas, USA: American Society of Mechanical Engineers, Oct. 2014, p. V002T30A006. [Online]. Available: https://asmedigitalcollection.asme.org/DSCC/proceedings/DSCC2014/46193/San%20Antonio,%20Texas,%20USA/228978

[8] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli, "Kinematic and dynamic vehicle models for autonomous driving control design," in *2015 IEEE Intelligent Vehicles*

Symposium (IV). Seoul, South Korea: IEEE, Jun. 2015, pp. 1094–1099. [Online]. Available: http://ieeexplore.ieee.org/document/7225830/

[9] T. Romijn, W. Hendrix, and M. Donkers, "Modeling and Control of a Radio-Controlled Model Racing Car," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 9162–9167, Jul. 2017. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2405896317323352

[10] P. Engström, D. Fredriksson, O. Olesen, D. Olsson, S. Sandberg, and A. H. F. M. H. F. M. S. El Din, "High performance autonomous racing with rc cars - designing models and controllers to implement a small scale system of multiple autonomous cars," p. 1–81, 2018.

[11] U. Shukla and P. Mahapatra, "The proportional navigation dilemma-pure or true?" *IEEE Transactions on Aerospace and Electronic Systems*, vol. 26, no. 2, pp. 382–392, Mar. 1990. [Online]. Available: http://ieeexplore.ieee.org/document/53445/

[12] A. D. Luca, "Visual servoing," Sep 2020. [Online]. Available: http://www.diag.uniroma1.it/deluca/rob2_en/17_VisualServoing.pdf

[13] C. Ravizzoli, "Identification and control of an rc car for drifting purposes," Master's thesis, Politecnico di Milan, 2016-2017.

[14] "RPi Camera (D), Raspberry Pi Camera Module, Fixed-focus." [Online]. Available: https://www.waveshare.com/rpi-camera-d.htm

[15] *1.5kg.cm Analog Servo product Specification.*

[16] Palisade, "What is Monte Carlo Simulation?" [Online]. Available: https://www.palisade.com/risk/monte_carlo_simulation.asp

[17] *MPU-6000/MPU-6050 Product Specification*, InvenSense, August 2013, revision 3.4.

[18] "Arduino - Home." [Online]. Available: https://www.arduino.cc/

[19] "Arduino Nano | Arduino Official Store." [Online]. Available: https://store.arduino.cc/usa/arduino-nano

[20] *Ultrasonic Ranging Module HC - SR04*, Elec Freaks.

[21] "Power Consumption Benchmarks | Raspberry Pi Dramble." [Online]. Available: https://www.pidramble.com/wiki/benchmarks/power-consumption

[22] "UART vs I2C vs SPI – Communication Protocols and Uses," Sep. 2019. [Online]. Available: /blog/2019/09/25/uart-vs-i2c-vs-spi-communication-protocols-and-uses/

[23] "Raspberry Pi to Arduino SPI Communication." [Online]. Available: http://robotics.hobbizine.com/raspiduino.html

[24] S. C. |. D. Electronics | 49, "Basics of the I2C Communication Protocol," Feb. 2016. [Online]. Available: https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/

[25] "Accessing the Raspberry Pi Camera with OpenCV and Python," Mar. 2015. [Online]. Available: https://www.pyimagesearch.com/2015/03/30/accessing-the-raspberry-pi-camera-with-opencv-and-python/

[26] "Realtime Color Detection (Webcam) [8] | OpenCV Python Tutorials for Beginners 2020 - YouTube." [Online]. Available: https://www.youtube.com/watch?v=Tj4zEX_pdUg

[27] "Control webcam with servo motor and raspberry pi - Opencv with Python," Jun. 2019. [Online]. Available: https://pysource.com/2019/06/05/control-webcam-with-servo-motor-and-raspberry-pi-opencv-with-python/

[28] "Arduino SPI Tutorial: Communication between two Arduino as Master and Slave." [Online]. Available: https://circuitdigest.com/microcontroller-projects/arduino-spi-communication-tutorial

[29] "adafruit/Adafruit_mpu6050," Oct. 2020, original-date: 2019-09-16T22:39:43Z. [Online]. Available: https://github.com/adafruit/Adafruit_MPU6050

Appendix A

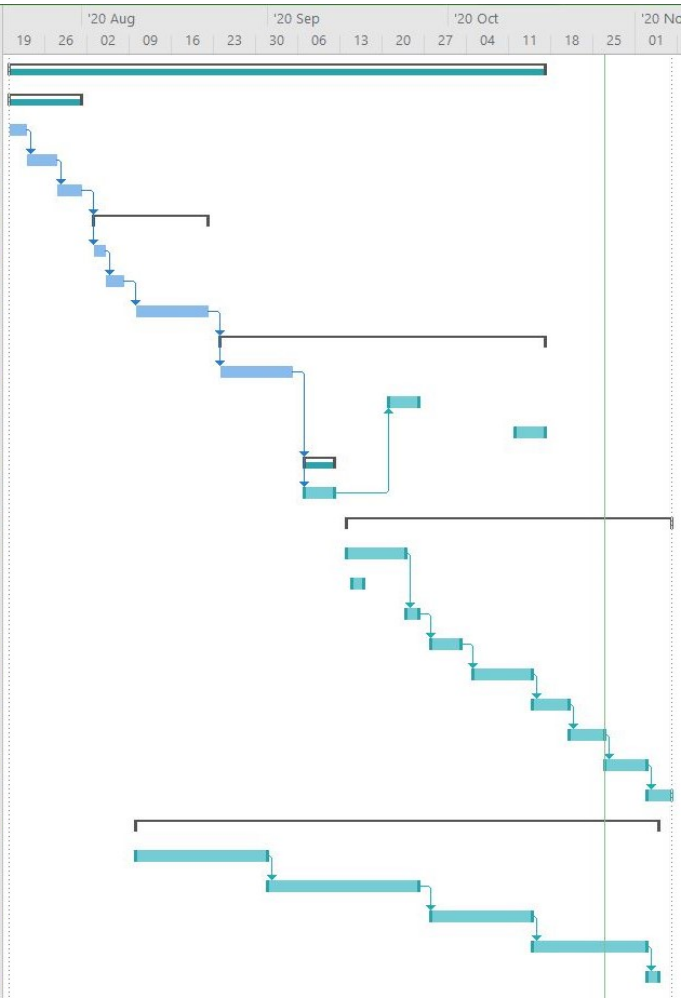
Project Planning Schedule

An agile project management approach was used for the scheduling of this project. The project was composed of several sprints which were generally allocated to each week in the schedule. These sprints had their own objectives and they helped to adapt each milestone as the project progressed.

A full project schedule is shown in Figure A.1 below.

Figure A.1: Project schedule and Gantt chart

Task Name	Durati	Start	Finish
Theoretical Progress	**65 days**	**20/07/20**	**20/10/16**
Preperation	**10 days**	**20/07/20**	**20/07/31**
Review content on vehicle modelling and missile guidance	3 days	20/07/20	20/07/22
Practice Simulink modelling on AA883 exercises	3 days	20/07/23	20/07/27
Create airplane model in Simulink to better modelling skills	4 days	20/07/28	20/07/31
Car Model	**15 days**	**20/08/03**	**20/08/21**
Collect and review literature on car dynamics	2 days	20/08/03	20/08/04
Choose car model	3 days	20/08/05	20/08/07
Model car dynamics and controls in Simulink	10 days	20/08/10	20/08/21
Simulation	**40 days**	**20/08/24**	**20/10/16**
Simulate collision scenario with target car	10 days	20/08/24	20/09/04
Simulate with added camera controller	5 days	20/09/21	20/09/25
Fix bugs and compile results	5 days	20/10/12	20/10/16
Seeker Model	**5 days**	**20/09/07**	**20/09/11**
Design and implement camera controller	5 days	20/09/07	20/09/11
Report Progress	**40 days**	**20/09/14**	**20/11/06**
Write Literature Review	8 days	20/09/14	20/09/23
Compile report template	2 days	20/09/15	20/09/16
Introduction	2 days	20/09/24	20/09/25
System Design	5 days	20/09/28	20/10/02
Physical System Design	8 days	20/10/05	20/10/14
Simulation Results	4 days	20/10/15	20/10/20
First run-through edit and changes	4 days	20/10/21	20/10/26
Practical build results write up	5 days	20/10/27	20/11/02
Conclusion, Abstract, Appendices, etc.	4 days	20/11/03	20/11/06
Physical Demonstrator Progress	**63 days**	**20/08/10**	**20/11/04**
Choose parts and find suppliers	16 days	20/08/10	20/08/31
Finalize parts and order them	19 days	20/09/01	20/09/25
Wait for parts and fix order issues	13 days	20/09/28	20/10/14
Build and code	13 days	20/10/15	20/11/02
Run tests	2 days	20/11/03	20/11/04



Appendix B

ECSA Exit Level Outcomes Compliance

This project was able to achieve all the required ECSA Exit Level Outcomes (ELO). The description of each ELO and how this project complied with the respective ELO are found in Table B.1 below.

Table B.1: ECSA outcomes and project compliance

ECSA ELO	Project compliance
ELO 1. Problem solving: Identify, formulate, analyse and solve complex engineering problems creatively and innovatively.	The problem of creating a proportional navigation system for an autonomous car was identified. This problem was defined in separate statements and objectives were created to solve the problem. The project followed a systems approach by breaking the problem into smaller components to solve the identified problem. *(Sections 1, 3, 5, 6, 7)*
ELO 2. Application of scientific and engineering knowledge: Apply knowledge of mathematics, natural sciences, engineering fundamentals and an engineering speciality to solve complex engineering problems	A variety of engineering spheres of knowledge was applied in this project. This project included mathematical modelling, simulated tests, controller design, hardware design, electronic design and building, software design and programming, and signal processing. *(Sections 3 - 8)*
ELO 3. Engineering Design: Perform creative, procedural and non-procedural design and synthesis of components, systems, engineering works, products or processes.	Procedural design steps were followed to model the vehicle used in this project and to design the hardware of this system. The control system design involved non-procedural design methods for testing the physical limits of the RC car and the camera. These designs were synthesized in the form of simulated models, hardware components soldered together, 3D printing of parts, coding of software, and physical testing of the complete system. *(Sections 3 - 8)*

ELO 4. Investigations, experiments and data analysis: Demonstrate competence to design and conduct investigations and experiments.	A significant amount of research was performed to investigate possible designs for each component of the system. Simulated experiments were designed and executed to collect, graph, and analyze data. Physical experiments were set up with real-time reading of sensor values during test conditions to validate individual components of the system. Informal tests were also run to check the physical limitations of the RC car and camera. *(Sections 4, 6, 7, 8)*
ELO 5. Engineering methods, skills and tools, including Information Technology: Demonstrate competence to use appropriate engineering methods, skills and tools, including those based on information technology.	The engineering methods, tools and skills used in this project were: MATLAB and Simulink for mathematical modelling, simulation and controller design; feedback control system design; hardware design using embedded systems; 3D-part design using Inventor Professional; real-time image processing; Python coding for Raspberry Pi; C++ coding for Arduino Nano; Latex coding for report writing. *(Entire report)*
ELO 6. Professional and technical communication: Demonstrate competence to communicate effectively, both orally and in writing, with engineering audiences and the community at large.	This report acts as a technical document communicating the objectives and outcomes of this project in a writing format. The solution provided by this project will also be validated with verbal communication to an engineering audience during a presentation. The objectives and outcomes also had to be verbally simplified when explaining the project to friends, family, and colleagues. *(Written report, oral presentation, informal communication)*
ELO 8. Individual work: Demonstrate competence to work effectively as an individual.	All solutions and designs are my own. External resources were referenced where necessary and any external resources used had to be adapted to fit the project's requirements. All building, coding, adaptations, and writing were done by me alone. *(Entire report)*
ELO 9. Independent Learning Ability: Demonstrate competence to engage in independent learning through well-developed learning skills.	To complete this project, I had to learn the following new concepts: vehicle modelling in Simulink; simulation design in Simulink; implementing controllers in embedded systems; C++ coding; image processing; ordering and choosing parts based on budget and time constraints; theoretical and physical system design from scratch. *(Entire report)*

Appendix C

Pictures of Practical Tests

N = 4

Frame 1
The target is rolled in front of the RC car.

Frame 2
The car begins following behind the target.

Frame 3
The car builds up a lead angle from the left side of the target and initiates its path to interception.

Frame 4
The RC car prematurely intercepts the target and misses it.

Frame 5
The RC car is able to continue turning right, making a full revolution, until it has the target in its field of view again.

Frame 6
The car then makes its way towards the stationary target, approaching from the target's right hand side as it builds up a lead angle.

Frame 7
The car finally successfully collides with the stationary target.

N = 3

Frame 1
The target is rolled in front of the RC car.

Frame 2
The car begins following behind the target.

Frame 3
The car builds up a lead angle from the left side of the target and initiates its path to interception.

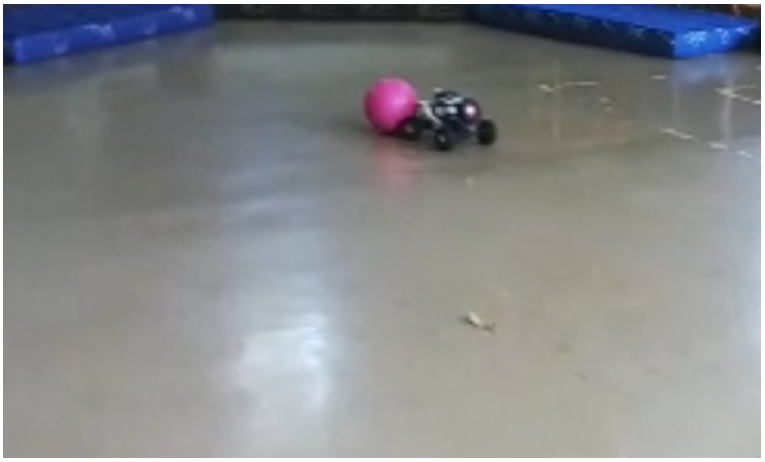
Frame 4
The car successfully intercepts its target from the left hand side.

N = 2

Frame 1
The target is rolled in front of the RC car.

Frame 2
The car begins following behind the target.

Frame 3
The car continues to follow the target from behind without building up a lead angle.



Frame 4
The car successfully collides with its target from behind.

Appendix D

Purchased Parts List

WL TOYS 12429 STORM RC CAR
Scale: 1/12th
Max Speed: 40km/h
Dimensions: 38.5 x 26 x 20.5cm
Supplier: RC EDGE
Price: R 1825.00

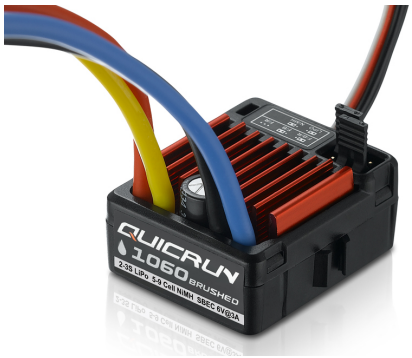
RASPBERRY PI 3B+
Processor: Quad core A53 (ARM v8) 64-bit SoC @ 1.4GHz
Memory: 1GB LPDDR2 SDRAM
Power: USB connector for 5.1V / 2.5A DC
Supplier: Micro Robotics
Price: R802.70

ARDUINO NANO
Microcontroller: ATmega328
Clock Speed: 16 MHz
Power: 7-12 V, 19mA Consumption
Supplier: Micro Robotics
Price: R424.35

ROMOSS SENSE4 MINI
Capacity: 10000mAh (37Wh)
Output: 5V/1A & 5V/2.1A (2.1A together)
Supplier: Takealot
Price: R279.00

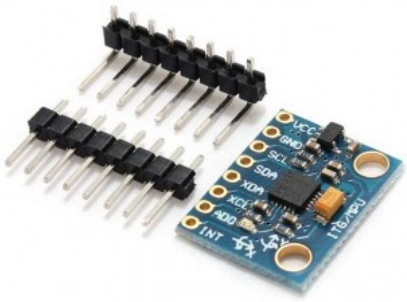
DS3225 LARGE SERVO
Torque: 25kg/cm @ 6.8V
Operating Angle: 270deg
Operating Power: 4.8-7.2 V, 180 mA
Supplier: Micro Robotics
Price: R258.75

FS90 MICRO SERVO
Torque: 1.5kg/cm @ 6V
Speed @ 6V: 0.10 sec/60
Operating Voltage: 4.8-6 V
Supplier: Micro Robotics
Price: R56.35



HOBBYWING QUICRUN 1060 BRUSHED ESC
Cont./Peak Current: 60A/360A
Motor Limit: 540 or 550 Size Motor
Input: 2-3S LiPo/6-9 Cells NiMH
Supplier: RC EDGE
Price: R407.00

RASPBERRY PI CAMERA (D)
Camera: 5 MP
Angle of View (diag): 66deg
Supplier: Micro Robotics
Price: R227.70



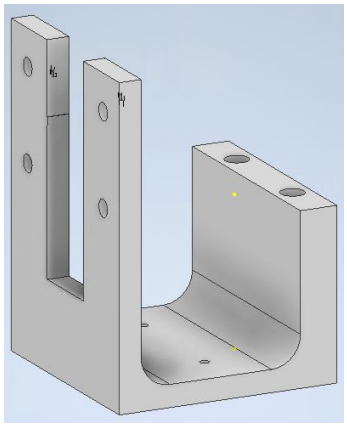
MPU-6050 6DOF MODULE
Operating Voltage: 3.3V DC Maximum
Degrees of Freedom: 6 (Acceleration and Tilt)
Communication: I2C
Supplier: DIY Electronics
Price: R 42.95

Total Price = R4323.80 (excluding delivery)

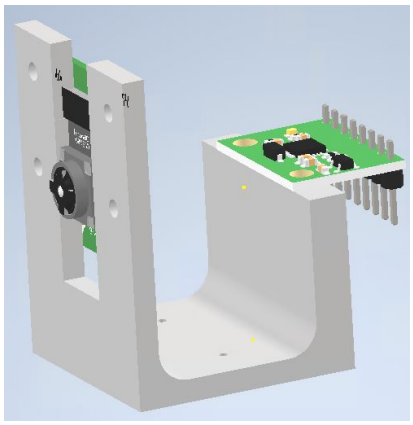
Appendix E

3D Printed Parts

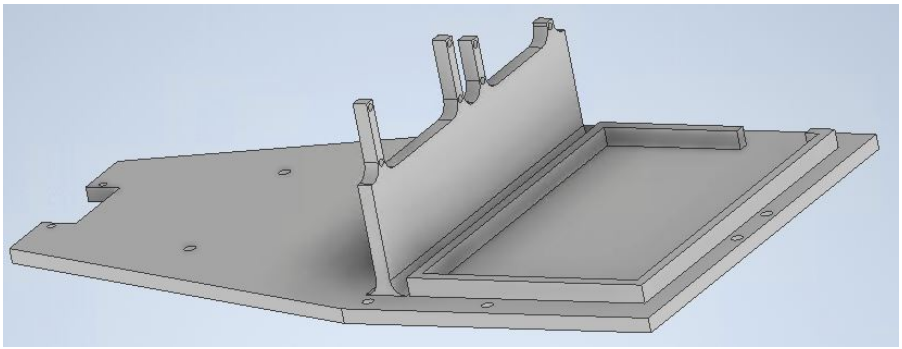
Figure E.1: 3D CAD designs



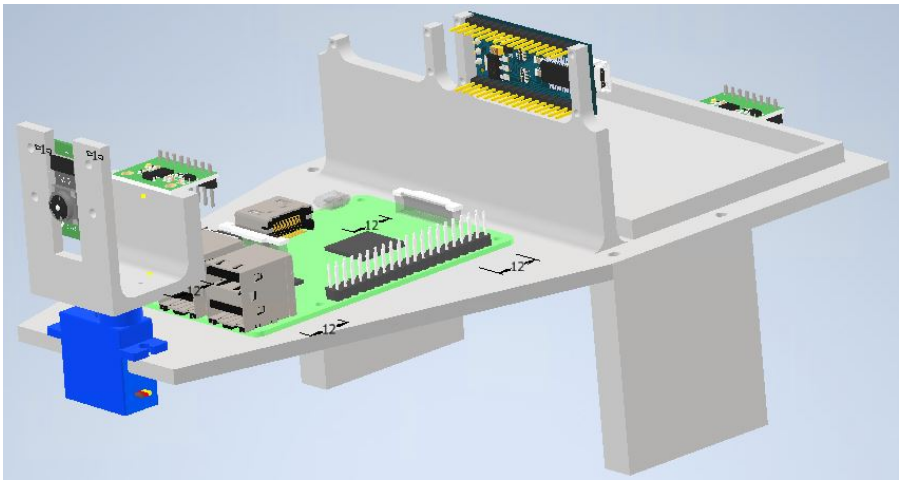
(a) Empty camera holder



(b) Camera and gyro attached



(c) 3D printed base



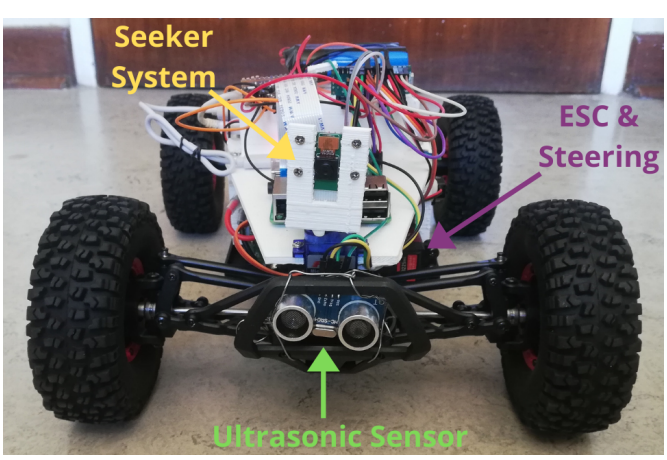
(d) Full 3D printed assembly

Appendix F

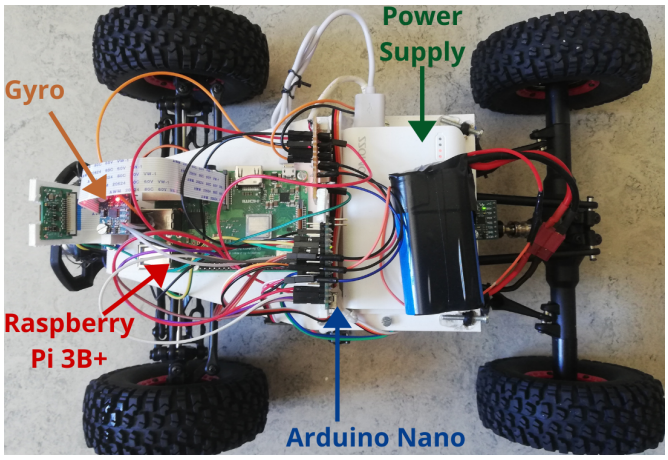
Full Build of Practical Demonstrator

F.1. Version 1

Figure F.1: Full build - *Version 1*



(a) Front view



(b) Top view

F.2. Version 2

Figure F.2: Full build - *Version 2*

(a) Front view

(b) Side view

Appendix G

Project Code

Figure G.1: Colour Detection (Python)

```python
import cv2
import numpy as np

cap = cv2.VideoCapture(-1)

cap.set(3, 640)
cap.set(4, 480)

def empty(a):
    pass

cv2.namedWindow("HSV")
cv2.resizeWindow("HSV",640,240)
cv2.createTrackbar("HUE Min","HSV",0,179,empty)
cv2.createTrackbar("HUE Max","HSV",179,179,empty)
cv2.createTrackbar("SAT Min","HSV",0,255,empty)
cv2.createTrackbar("SAT Max","HSV",255,255,empty)
cv2.createTrackbar("VALUE Min","HSV",0,255,empty)
cv2.createTrackbar("VALUE Max","HSV",255,255,empty)


while True:

    _, img = cap.read()
    imgHsv = cv2.cvtColor(img,cv2.COLOR_BGR2HSV)

    h_min = cv2.getTrackbarPos("HUE Min","HSV")
    h_max = cv2.getTrackbarPos("HUE Max","HSV")
    s_min = cv2.getTrackbarPos("SAT Min","HSV")
    s_max = cv2.getTrackbarPos("SAT Max","HSV")
    v_min = cv2.getTrackbarPos("VALUE Min","HSV")
    v_max = cv2.getTrackbarPos("VALUE Max","HSV")

    lower = np.array([h_min,s_min,v_min])
    upper = np.array([h_max,s_max,v_max])
    mask = cv2.inRange(imgHsv,lower,upper)
    result = cv2.bitwise_and(img,img,mask = mask)

    cv2.imshow('Original',img)
    cv2.imshow('Mask',mask)
    cv2.imshow('Result',result)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

Figure G.2: Monte Carlo Simulation (MATLAB)

```
%%% Dimensions %%%
l_f = 0.13;
l_r = 0.13;

%%% Control %%%
ESC = 5;
V_T = 2;
N = 2;
K = 17.2;

%%% Initial Conditions %%%
X_M0_min = -10;
X_M0_max = -5;

Y_M0_min = -5;
Y_M0_max = 5;

psi_M0_min = -pi/6;
psi_M0_max = pi/6;

n = 50;

for i = 1:n

    % Randomize Main Car initial conditions
    X_M0 = X_M0_min+rand*(X_M0_max-X_M0_min);
    Y_M0 = Y_M0_min+rand*(Y_M0_max-Y_M0_min);
    psi_M0 = psi_M0_min+rand*(psi_M0_max-psi_M0_min);

    psi_T0 = 0;

    sim("car.slx");

    X_M = ans.yout.getElement("X_M").Values.Data;
    Y_M = ans.yout.getElement("Y_M").Values.Data;
    X_T = ans.yout.getElement("X_T").Values.Data;
    Y_T = ans.yout.getElement("Y_T").Values.Data;

    plot(X_M,Y_M,X_T,Y_T)

    hold on

end

hold off

xlabel('X Axis (m)')
ylabel('Y Axis (m)')
```

Figure G.3: Individual Simulation Example (MATLAB)

```matlab
%%% Dimensions %%%
l_f = 0.13;
l_r = 0.13;

%%% Control %%%
ESC = 5;
V_T = 2;
N = 2;
K = 17.2;

%%% Initial Conditions %%%
X_M0_min = -10;
X_M0_max = -5;

Y_M0_min = -5;
Y_M0_max = 5;

psi_M0_min = -pi/6;
psi_M0_max = pi/6;

% Randomize Main Vehicle initial conditions
X_M0 = X_M0_min+rand*(X_M0_max-X_M0_min);
Y_M0 = Y_M0_min+rand*(Y_M0_max-Y_M0_min);
psi_M0 = psi_M0_min+rand*(psi_M0_max-psi_M0_min);

%% THESE VALUES DEPEND ON TEST CONDITIONS %%
V_T = 2;
psi_T0 = 0;
%%  %%  %%  %%

sim("car.slx");

X_M = ans.yout.getElement("X_M").Values.Data;
Y_M = ans.yout.getElement("Y_M").Values.Data;
X_T = ans.yout.getElement("X_T").Values.Data;
Y_T = ans.yout.getElement("Y_T").Values.Data;

t_M = ans.yout.getElement("X_M").Values.Time;
t_T = ans.yout.getElement("X_T").Values.Time;

plot(X_M,Y_M,'blue')

hold on

plot(X_T,Y_T,'red')

scatter3(X_M,Y_M,t_M,8,'filled','black')
scatter3(X_T,Y_T,t_T,6,'filled','red')

grid on

hold off

xlabel('X Axis (m)')
ylabel('Y Axis (m)')
legend('Car','Target')
```

Figure G.4: Video Processing and Object Detection (Python)

```python
from picamera.array import PiRGBArray
from picamera import PiCamera
import cv2
import numpy as np
import spidev
import time

spi = spidev.SpiDev() #Enable SPI
bus =0 # Bus address on RPI
device =0

spi.open(bus,device) # Open SPI connection to external device
spi.max_speed_hz = 3000000 #Set maximum speed for SPI transfer
spi.mode =0 #set SPI

camera = PiCamera()
camera.resolution = (640, 480)
camera.framerate = 32
rawCapture = PiRGBArray(camera, size=(640, 480))
# allow the camera to warmup
time.sleep(0.1)

cam_position = [int(0)]

print('Started!')
spi.writebytes([200])

for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True):

    image = frame.array

    rows, cols, _ = image.shape

    x_medium = int(cols / 2)
    centre = int(cols / 2)
    position = 90 # degrees

    # Create colour mask
    hsv_frame = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    h_min = 155
    h_max = 179
    s_min = 50
    s_max = 255
    v_min = 145
    v_max = 255

    lower = np.array([h_min,s_min,v_min])
    upper = np.array([h_max,s_max,v_max])

    # Apply colour mask and detect object
    mask = cv2.inRange(hsv_frame,lower,upper)
    contours, _ = cv2.findContours(mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    contours = sorted(contours, key=lambda x:cv2.contourArea(x), reverse=True)

    for cnt in contours:
        (x, y, w, h) = cv2.boundingRect(cnt)

        x_medium = int((x + x + w) / 2)
        break

    cv2.line(image, (x_medium, 0), (x_medium, 480), (0, 255, 0), 2)

    #cv2.imshow("Frame", image)
    key = cv2.waitKey(1) & 0xFF
    # clear the stream in preparation for the next frame
    rawCapture.truncate(0)
    # if the `q` key was pressed, break from the loop
    if key == ord("q"):
        break

    cam_position = [int(x_medium*(9/32))]

    spi.writebytes(cam_position)

cv2.destroyAllWindows()
```

Figure G.5: Main Code - Uploaded to Arduino Nano (C++)

```
#include <SPI.h>
#include <PIDController.h>
#include <Servo.h>
#include <math.h>
#include <Adafruit_MPU6050.h>

PIDController pid;
Servo cam_servo, car_servo, ESC;

volatile bool received;
volatile bool trigger = true;
volatile byte spi_value;
volatile long echoTime1, echoTime2;

const byte trigPin = 3;
const byte echoPin = 2;

int cam_position;
int cam_control = 90;
int steer_angle = 90;

bool PIDstart = false;

double T = 0.001;
double Ki = 0.35;
int N = 3;

float W_LOS = 0;

Adafruit_MPU6050 mpu;
Adafruit_Sensor *mpu_gyro;

long gyroTime;

long distance = 100;

boolean start = false;
```

```
//////////////////////
//      SETUP       //
//////////////////////

void setup () {

////// SPI //////
  pinMode(MISO,OUTPUT);
  SPCR |= _BV(SPE);
  SPI.attachInterrupt();

  received = false;

////// PID /////
  pid.begin();
  pid.setpoint(90);

  pid.tune(Ki, T);     // Tune the PID, arguments
  pid.limit(0, 180);     // Limit the PID output

////// ESC & Servos //////
  cam_servo.attach(4);
  cam_servo.write(90);

  car_servo.attach(6);
  car_servo.write(90);

  ESC.attach(8);

////// Gyro //////

if (!mpu.begin()) {
    Serial.println("Failed to find MPU6050 chip");
    while (1) {
      delay(10);
    }
  }
  Serial.println("MPU6050 Found!");
  mpu_gyro = mpu.getGyroSensor();

////// Ultrasonic Sensor /////
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  attachInterrupt(digitalPinToInterrupt(echoPin), echoTimer, CHANGE);

  Serial.println("");
  Serial.println("Ready!");
}
```

```
//////////////////////////
//     FUNCTIONS        //
//////////////////////////

// SPI interrupt routine function
ISR (SPI_STC_vect) {
  spi_value = SPDR;                // Value received from master
  received = true;
}

// Ultrasonic sensor interrupt routine function
void echoTimer(){

  if(digitalRead(echoPin) == HIGH){
    echoTime1 = micros();
  }

  if(digitalRead(echoPin) == LOW){
    echoTime2 = micros();
    trigger = true;
  }
}

// Ultrasonic sensor signal trigger
long send_trigger() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  trigger = false;
}

// Gyroscope read values function
float getGyro(){

  sensors_event_t gyro;
  mpu_gyro->getEvent(&gyro);

  return gyro.gyro.z;
}
```

```
////////////////////////////
//       MAIN LOOP        //
////////////////////////////

void loop () {

  if(received) {                // Receive value from R-Pi
    cam_position = spi_value;
    received = false;
  }

  if(cam_position == 200) {     // Start main loop functions based on R-Pi command
    cam_position = 0;
    start = true;
    ESC.write(100);
    delay(1000);
    ESC.write(110);
    gyroTime = millis();
  }

  if(start) {       // Officially start main loop

    // Camera controller
    cam_control = pid.compute(cam_position);    // Returns the optimal output
    cam_servo.write(cam_control);

    if((millis() - gyroTime) >= 100){

      W_LOS = getGyro();
      W_LOS = round(W_LOS * 10) / 10.0;

    }

    // Turning rate controler (scaled for servo angle in degrees)
    steer_angle = (int)(((((l_f+l_r)/V_M)*(N*W_LOS))*(180/PI))+90);

    // Limit steer angle based on physical demonstrator's limits
    if(steer_angle > 150){
      steer_angle = 150;
    }

    if(steer_angle < 30){
      steer_angle = 30;
    }

    car_servo.write(steer_angle);

    // STOP condition

    if((trigger)&&((micros()-echoTime2) >= 20000)){
      send_trigger();
    }

    distance = ((echoTime2-echoTime1)*0.034)/2; // From datasheet

    if((distance <= 20)&&(distance > 0)) {

      ESC.write(0);             // Stop moving car
      while(1){};               // Pause all operations
    }
  }
}
```