Autonomous Driving using Reinforcement Learning

Project Team

Afrah Hamdan

Amir Doufish

Nujood Shawar

Supervisor

Dr. Hashim Al-Tamimi

Submitted to the College of Engineering
in partial fulfillment of the requirements for the degree of
Bachelor degree in Mechatronics Engineering

Palestine Polytechnic University

Dec, 2018

Palestine Polytechnic University

Collage of Engineering

Mechanical Engineering Department

Hebron – Palestine

Autonomous Driving using Reinforcement Learning

Project Team:

Afrah Mosa Hamdan

Amir"m,Nabeh" Doufish

Nujood Kazem Shawar

Submitted to the Collage of Engineering

In partial fulfillment of the requirements for the

Bachelor degree in Mechatronics Engineering.

Supervisor Signature

[illegible]

Testing Committee Signature

[illegible] [illegible]

Chair of the Department Signature

[illegible]

Dec 2018

Palestine Polytechnic University

Collage of Engineering

Mechanical Engineering Department

Hebron – Palestine

Autonomous Driving using Reinforcement Learning

Project Team:

Afrah Mosa Hamdan

Amir"m,Nabeh" Doufish

Nujood Kazem Shawar

Submitted to the Collage of Engineering

In partial fulfillment of the requirements for the

Bachelor degree in Mechatronics Engineering.

Supervisor Signature

……………………………………….

Testing Committee Signature

…………………………………   ……………………………….

Chair of the Department Signature

……………………………………….

Dec 2018

شكر وتقدير

لله, لأنْ أكرمنا بأن أوصلنا لما نحن ها هنا عليه يومنا هذا, لمن لم ينسنا يوماً من كرمه ولطفه , لمن آنسنا وعّلمنا مما لم نكن نعلم من ذي قبل وألهمنا من لدنه رشدا وما تكفي الكلمات شكرك يا رباه. شكرُنا دائمٌ كذاك لمن سهر تلكمُ الليالي لاجلنا ولمن أفنى فؤاده ليرسم بنا املاً ما انفكَّ عن اليقين به ... لذوينا الاحياء منهم والاموات في ارماسهم. رضي الله عنكم بكل ذرة خير ذريتموها بنا وها أنتم أُولاء ترونها تزهر بعد هذه السنين. أساتذتنا, زملاءناَ وكل من آزرنا بقلبه وكلمته وعلمه, كل من قد حفر طيب وجهه داخل لُباب افئدتنا مذ اول يوم لنا في هذا المقام.

لايامكم تلك ...

لقلوبكم تلك ...

نقول أنْ حاشا للقلب نسياها ...

فبارك الله بأعماركم واعمالكم ...

Acknowledgement

We would like to express our gratitude to our supervisor, Dr. Hashim Al-Tamimi, for his support. To Eng.Wael Al-Takruri for his remarkable help and to Eng. Mahmoud Al-Sharabati for his technical support. Our Deep thanks is also going for all of our teachers whom were the reason after God that we reached to that point of accomplishment. Thanks are also for our parents, classmates and friends for their cooperation and encouragement.

Abstract

The aim of this project is to apply reinforcement learning (RL) to a car-like robot. Through RL the car should learn to drive a long some desired path. The learning should be done by itself through changing the steering angle and the speed of the car. The used methodology is an integration between the agent -controller- and its environment, so the agent has a feedback from the environment that can be received by a camera to detect the orientation of the car and a center point to drive a round, with that feedback the agent would be able to adjust its algorithm or policy followed. In our case we have chosen Q-Learning method as an approach of reinforcement learning to be our agent algorithm.

الملخص

يهدف المشروع الى ابراز التكامل والتعاضد الذي يجمع ما بين عالم السيارات والذكاء الاصطناعي , متمثلا ذلك بتطبيق احدى خوارزميات تعلم الآلة (التعلم بالتعزيز) على سيارة لتقوم بالقيادة ذاتيا وبدون تدخل الانسان ضمن مسار مرغوب لها , إذ يجب ان تحافظ على دورية تعديل حركتها لضمان ديمومة تحقيقها للمسار المطلوب في حال تم تغيير اي شيء على طبيعه البيئة المحيطة فيها.

ما تقوم به الخوارزميه تحديدا هو اكتشاف البيئة التي تجري عمليه تعليم السيارة داخل نطاقها وذلك بأخذ تغذية راجعة من هذه البيئة, في هذا المشروع تم اعتماد نظام مراقبة متمثل بكاميرا لتحديد موقع السيارة وزاوية دورانها بالنسبة لنقطة المركز باستمرار, كما وتم اخذ تغذية راجعة من زاوية دوران الاعجل الامامية ايضا , ليلي ذلك ادراج وتحليل تلك البيانات اجمعها في خوارزمية التعلم المعتمده وهي Q-Learning والتي بدورها تقوم بالتوصل بعد مرحلة من التجارب والتحديثات المستمرة للبيانات إلى الافعال المثلى التي يجب ان تقوم بها السيارة لكي تحقق الهدف المراد ذاتيا.

Table of contents

List of Figures

1

CHAPTER 1
Introduction

1.1 Introduction

In the last years the domain of mobile robotics is shifting from user-teleoperated systems to autonomous or semi-autonomous systems. An autonomous robot is a robot that implements tasks with a high degree of independence so that it can perform the tasks entrusted to it without human intervention. This independency or self-learning came from many methods and algorithm which is in our project derived from machine learning as a branch of artificial intelligence field.

Reinforcement leaning algorithm (RL) as a learning approaches, which allows a software agent to automatically determine the ideal action that the robot has to do in order to maximize its performance based on a feedback comes back from the robot sensor's, hence the agent keep updating its decisions to accomplish the desired behavior.

The advantage of this sort of learning that it needs no previous knowledge about its working environment, the agent or the robot can learn about its environment through interacting with it. Which makes a good alternative in the case of lack of information or training data that required to be fed into the agent in order to achieve the learning process.

In this project we have designed an autonomous car to drive on a specific path using reinforcement learning a self-learning strategy.

1.2 Project objective

This project aims to get a remote controlled (RC) car to maintain a circular motion around a specific point autonomously. To achieve that the RC car must learn the best way to stay in that track using reinforcement learning, and to improve its learning process a vision system has been used to detect both of the position of the car and the desired center point, by using balls with specific colors on these spots, after so the camera sends these taken images to personal computer (PC) to process them, these processed data transferred later wirelessly into the microcontroller that has the learning algorithm running on it, as a result of image processing the appropriate decision is taken depends on the current reward value.

1.3 Project constraints

- Colored markers are used for defining the position of the obstacle and orientation of the car.
- The algorithm is responsible on changing both of the steering angle and the throttle commands of a car.
- Computer vision is used as feedback for the learning algorithm.
- The learning experiments of the car have to be in a limited area.

1.4 Literature review

The relevant contributions that have combined control theory and machine learning altogether are illustrated in that section.

1.4.1 Based on machine learning autonomous car using raspberry-pi

Wang Zheng presents a self-driving car that was designed to drive on a path as shown in Figure 1.1 by using supervised learning of a neural network. This project contains two important components raspberry-pi and raspberry-pi camera module. There are two modes: the first one is training mode which the camera would provide images needed to train the neural network, and the second one which is autonomous mode would provide the images to the trained model to predict the movements and direction of the car [1].

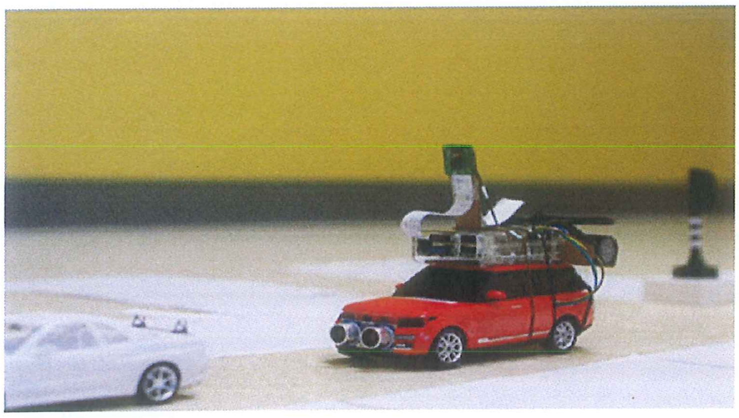


Figure 1.1: Self-driving car[1]

1.4.2 Reinforcement learning for a self-balancing motorcycle

Madhu Govindarajan presents a balancing motorcycle as shown in Figure 1.2, this motorcycle using RL to balance itself, which it balances for about 10 -15 seconds, but further training and changing the incentives of the reward function can train it to balance longer, the Q-learning used for avoiding any obstacles face it, moreover it can maneuver and drive along path [2].

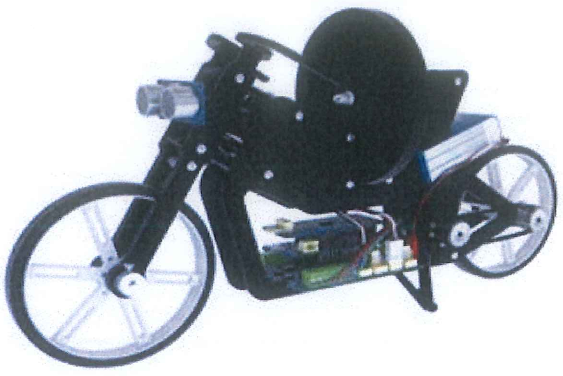


Figure 1.2: A self-balancing motorcycle[2]

1.4.3 Control of a quadrotor with reinforcement learning

As shown in Figure 1.3 the quadrotor is controlled with a neural network trained using RL. By RL, a network can be trained to directly map state to actuator command making any predefined control structure obsolete for training. The algorithm is used for complicated tasks is conservative but stable. Experiments show that policy network or the algorithm that the agent has developed can react to step response relatively accurately. This algorithm needs ten seconds to reach the stability after this test [3].

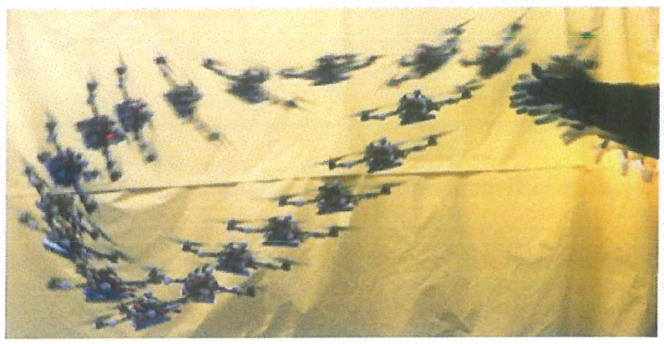


Figure 1.3: Throwing test of quadrotor[3]

1.4.4 Robot motor skill coordination with EM-based reinforcement learning

In Figure 1. 4 the experimental setup for the Pancake-Flipping task is shown. The robot obtains a new motor skill by learning the couplings across motor control variables. Firstly, the demonstrated skill is encoded in a modified version of dynamic movement primitives (DMP) form which used for the interaction connection information. Expectation-Maximization based RL is then used to modulate the mixture of dynamical systems initialized from the user's demonstration. The approach is evaluated on a torque-controlled 7-DOF Barrett WAM robotic arm. From the experiments, the robot learns two specific skills: a reaching task where the robot needs to adopt the learned movement to avoid an obstacle, and a dynamic pancake-flipping task [4].

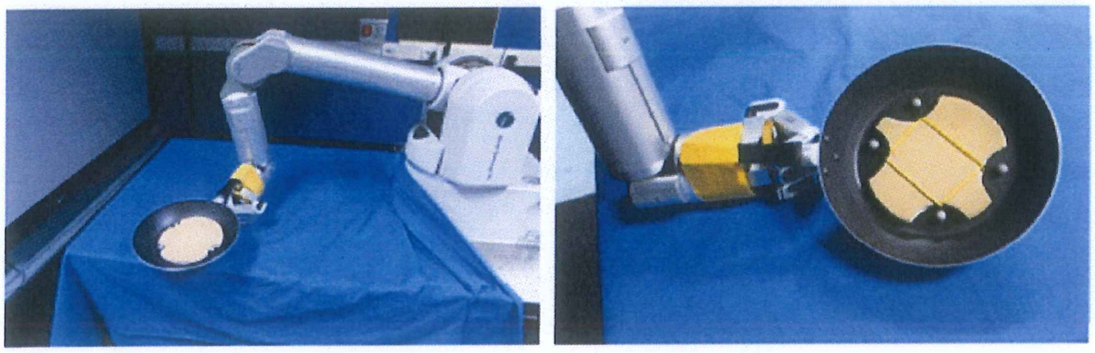


Figure 1.4: Pancake-flipping task[4]

2

CHAPTER 2
Background

2.1 Introduction

This chapter reviews the used reinforcement learning algorithm (Q- Learning method). with autonomous car to have a circular path.

2.2 Autonomous car

An autonomous car (self-driving car) is a vehicle that uses a set of sensors, cameras and the artificial intelligent, to reach specified destinations without human involvement. To be titled as fully autonomous, it must be able to travel without human orders to preplanned destination.

This project presents an autonomous car learns by reinforcement learning how to move in a circular motion around a center point as shown in Figure 2.1

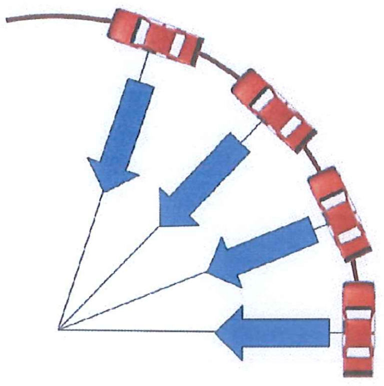


Figure 2.1: The desired motion of our autonomous car[5]

To achieve that, the car was supported with sensors such as, ultrasonic to prevent any collisions with obstacles or a wall, a camera used to keep updating the current position of the car in reference to its center point and finally to achieve that desired motion an RL algorithm has been employed as section 2.2 discusses.

2.3 Reinforcement learning

Reinforcement learning is the iterative process of an agent, learning to behave optimally in its environment by interacting with it. Which means that the way the agent learns to achieve a goal is by trying different actions in its environment and receiving positive or negative feedback also called reward. RL is used in a diverse area of disciplines, including robotics, automatic control, economics, and manufacturing.

In Figure 2.2, the process of reinforcement learning shows how the agent decides itself in the absence of previous training data to reach the goal through a process of experimentation and back rewards. whereas the agent receives a positive reinforcement reward signal when it makes an action that produces the desired result or gives a one near to, and a negative reward when it does not. Through this trial and error attempts, the agent can perform a better policy or trajectory to get the highest reward [6].

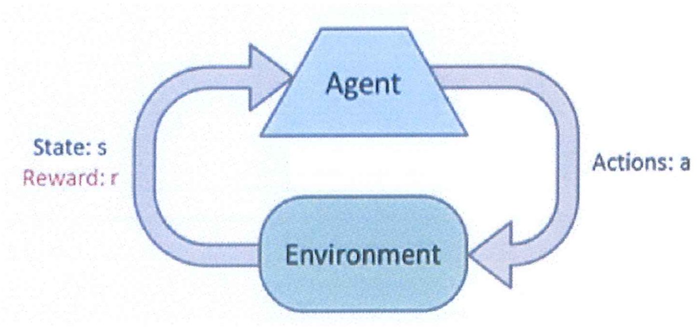


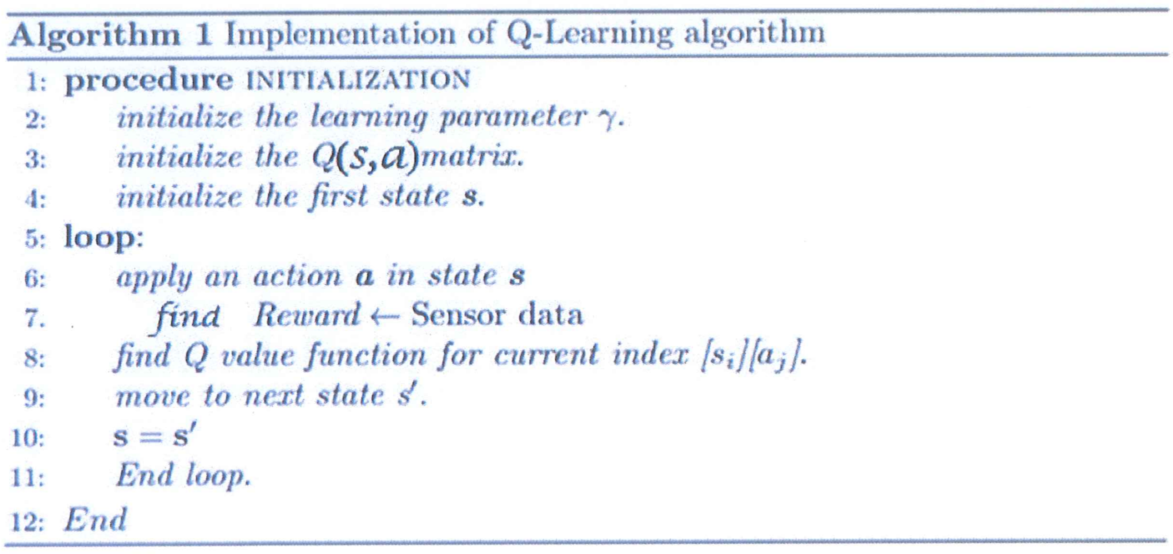
Figure 2.2: The process of reinforcement learning[6]

So, at each step (t) the agent:

- Executes an action **a**
- Receives a new state (current position) **s**
- Receives a scalar reward **r**

2.3.1 Q-Learning method

In this project, the algorithm of Q-Learning is chosen to find the optimal path without being explicitly told the dynamics of the system. at time step t, the car state is s, the action chosen is a, and the next state of the car ends up in after taking the action a. the Q value function represents the equation shown in Eq.2.1, whereas Q(s, a) is the quality or usefulness for taking this action in that current state, this is going to be saved as a matrix that the value function depend on the reward R(s,a). γ is being added to give a weight for either the current value of Q matrix or to trust its future values more through a verity between 0-1, where 0 means trust your current Q-matrix totally without give the future values any weight, and 1 means the whole weight goes to the future value without taking the current reward in account. The fundamental idea behind reward is that the learning agent is rewarded along the right path with increasing its value the agent knows that it doing well with taking this action in that current state and vice versa in that way the learning improves by time it will ended with a high reward value for achieving its goal [6].



Algorithm 1 Implementation of Q-Learning algorithm

```
1: procedure INITIALIZATION
2:     initialize the learning parameter $\gamma$.
3:     initialize the $Q(s,a)$matrix.
4:     initialize the first state s.
5: loop:
6:     apply an action a in state s
7.        find  Reward ← Sensor data
8:     find Q value function for current index $[s_i][a_j]$.
9:     move to next state $s'$.
10:    $s = s'$
11:    End loop.
12: End
```

Figure 2.3: Q-Learning algorithm[6]

$$Q(a, s) = R(s, a) + \gamma * max'_a[Q(s', a')] \tag{2.1}$$

3

CHAPTER 3
Conceptual design and component selection

3.1 Introduction

This section describes the architecture of this project, including the system components (subsystems), and the function of each component then demonstrates the relation between these elements that shown in Figure 3.1.

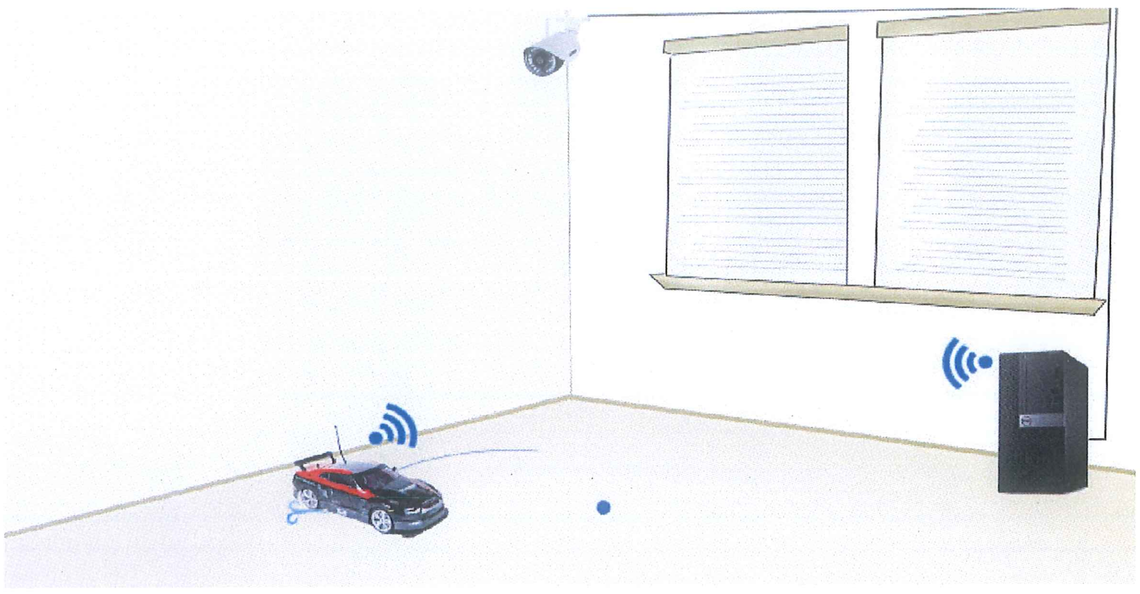


Figure 3.1: System architecture

As shown in Figure 3.1, the system consists of three integrated subsystems that perform altogether to achieve the goal, these subsystems are: a RC car, a vision system and a personal computer (PC), the first two subsystems are connected with each other through the PC. The RC car has a microcontroller for controlling purposes in addition to making a better exploration of the learning area, the PC has to be used in order to send the processed images that are taken by the vision system of the car position and orientation to the learning algorithm, that position can be known by distinguishing the detected balls that are located on specific locations on front and back sides of car, this captured images can be transmitting wirelessly during the learning period between the microcontroller and the PC system, through a wire between the camera and the computer.

It is worth mentioning that the microcontroller has the reinforcement algorithm where with each updating of the taken images the RL algorithm updates its Q-value matrix so that the learning improves with time with an indication of increasing or decreasing the values of Q-matrix [5].

3.2 Conceptual design

This section illustrates the overall process of *visual servoing* that states for taking the feedback information extracted from the vision sensor in order to control the angle of the car. Figure 3.2 shows the steps that the leaning car is going through to accomplish its motion.

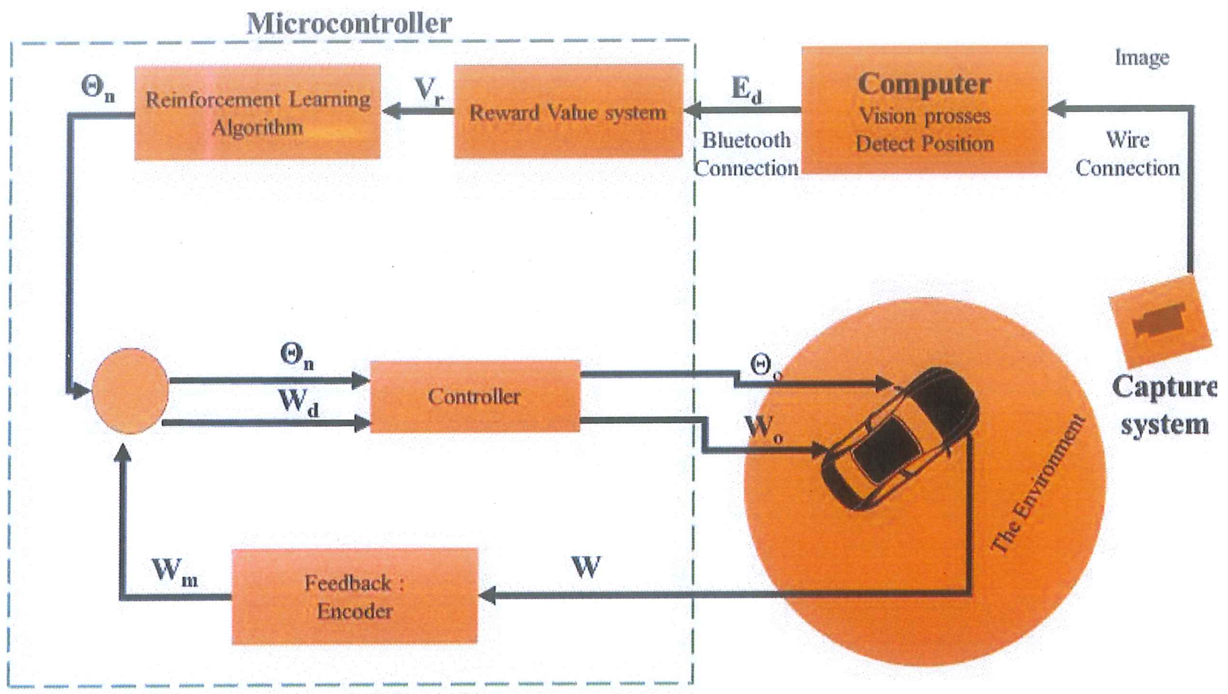


Figure 3.2: Conceptual design

In Figure 3.2, the RL algorithm makes a random action presented by change the steering angle of the car which yields a car motion in some spot, this new position is taken by the camera then sent into the PC to be processed there, after that it process the position and orientation can be known. The sequence is continued and take this information to convert it into a reward value or punishment so that the RL can decide if that taken action was good or not. This action in that state is saved in the history of learning process. After that the agent of the RL algorithm decides to take a new action and get a new value of reward. If the value of the reward variable tends to decrease by time, the RL algorithm realizes how bad it was taking that action or this sequence of actions and starts to avoid repeating this action in this specific state. In this way a better performance can be improved by time.

symbols indication:

- E_d: Error output from image process.0
- V_r: Reward Value , which get negative or positive according to the car position.
- Θ_n: New Angle, that came from the RL.
- Θ_o: Output angle, we don't measure because the servo has feedback system.
- W_d: desired angular velocity
- W_o: Output angular velocity
- W_m: Measured angular velocity
- W : angular velocity

3.3 Components interfacing with microcontroller

This section illustrates the connection between the microcontroller, PC computer and capture system which shown in Figure 3.3.

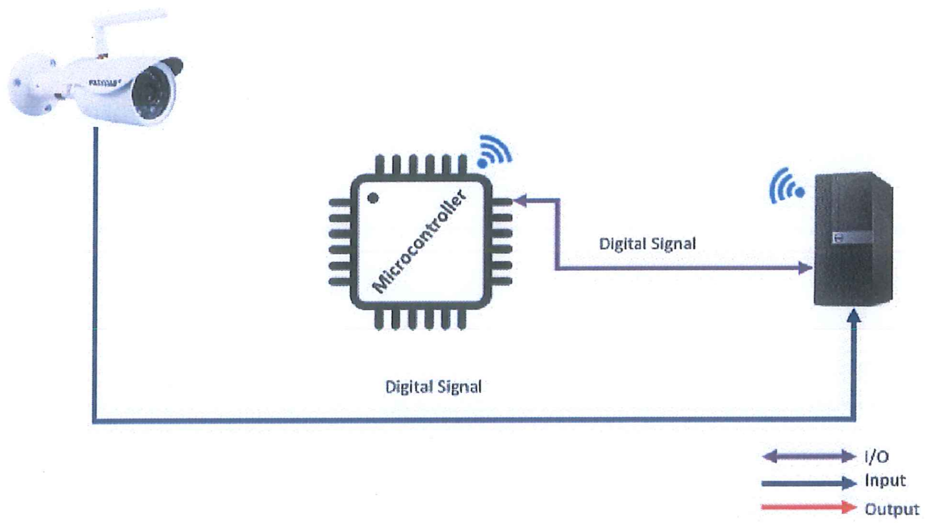


Figure 3.3: The main subsystems

Figure 3.3 shows the connection of the camera-PC subsystems, this connection can be achieved by a wiring cable, digital signal. The camera captures the image, after that the image is processed to detects the position of the RC car and sends the data to the PC. The PC sends the processing data back into the microcontroller that controls the car's steering angle.

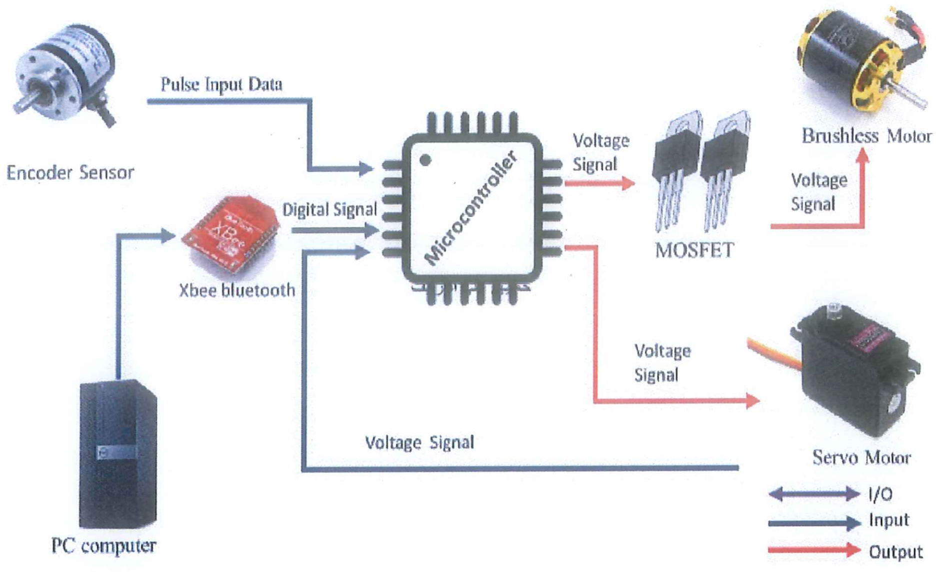


Figure 3.4: Components interfacing with microcontroller

As show in Figure 3.4, the connection of sensors, onboard Bluetooth with the PC, and actuators. The figure shows two inputs and two outputs, the input components are: the encoder sensor and Xbee bluetooth module that take the transferred data from the PC and send it into the microcontroller. The output components are the steering angle of the servo motor and the velocity of the DC motor which has two MOSFET to control its speed from the given Arduino voltage.

3.4 Q-Learning design

Q- Matrix, is the base of RL algorithm that we have chosen, which keeps updating its values in order to make sure that the optimal quality of the exploration process reached, the car should learn in terms of rewards and punishments. To get rewards it should move through states by choosing actions randomly as shown in Figure 3.5 the number of arrows that come out from each state represents the number of allowed actions, so the maximum number of actions is two, Also, the circles represent the states of the car for each state we have different positions for each motors and it moves from one state to another using action which is represented by arrows.

.

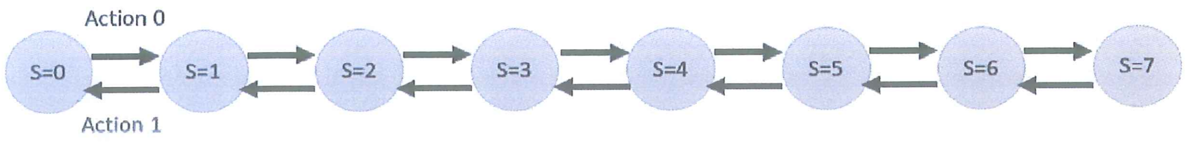


Figure 3.5: States and actions sequence of learning

3.5 Component selection

This section shows the selected components, the reason of selecting them, and other alternative components.

3.5.1 RC car

Since the car is the main component in that project, we have chosen (WL toys A959-B 2.4G 1/18 4WD Vehicle 70KM/h High Speed [7]) as shown in Figure 3.6 for the following reasons:

- This car is cheaper compared with other cars.
- Its size is suitable to carry out experiments in places such as a room in the house or laboratory, which means it does not need a specific space to operate in.
- It has large range of speed (0- 70 Km/h).
- For the dynamic factors issue it has a suspension that makes the system damped.
- The car consists of a single DC brushed motor and a four-wheel drive tires (4WD) these wheels are coupled mechanically together.
- The tires cause a small friction with the surface.

Figure 3.6: The chosen RC car[7]

3.5.1.1 RK-540PH brushed DC motor

The used brushed DC motor has nominal voltage six volt, at maximum efficiency its speed is 6088 revolution/minute, with current five ampere and output turqe is 22.2 N.m [8].

3.5.1.2 DC motor driver

Due to the large current that the motor consumes while it running from the source in Table 3.1, we need to attach a compatible driver between the microcontroller and the motor otherwise the Arduino would be damaged. The motor driver that came with the car cannot be used since the manufacture company erased all names of the integrated circuits (ICs). The candidate's drivers were:

1. **BT-2 (High current 30A DC motor driver)**

This driver in Figure 3.7 uses two special half bridge (BTS 7960 chip), operating current 30A, voltage 5-27V, motor forward and backward motion and a large size heat sink is mounted in.

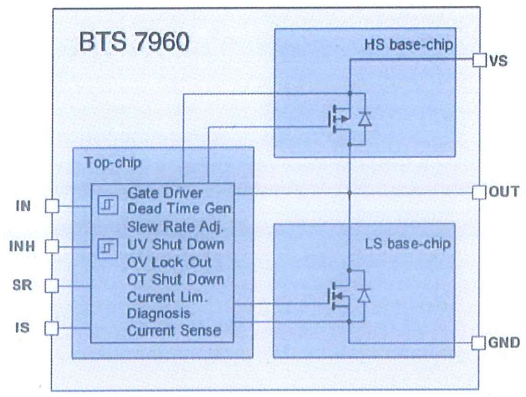


Figure 3.7: Block Diagram[9]

2. The IRF family MOSFET

This type of MOSFET has a large current operation, the IRF3205 can stand 110A and 55 V, and however after many experiments this type of MOSFET was not compatible with our motor so we have eliminated it.

3. The selected driver of the car

In that point we have decided to disassembly the car original driver that was came with it and use its MOSFET, this one was working fine during the operation process.

3.5.1.3 Battery

The used lithium battery that operates the car has come with the car itself, to ensure that it can run the motors and the Arduino properly in a specific time.

Now this is a calculation of the battery:

Total current of 2 serial pins with 40mA for each = 0.080 A

Total current of 6 Output pins with 20 mA for each=0.120 A

Total current of single DC motor = 0.005A

The current of single servo motor =0.100 A

The consuming power by the microcontroller is:

P=I*V

P=0.300 A * 5 V

P=1.5 Watt

The power of motors:

P= 5 A* 6 V

P=30 Watt

The total energy consumption in watt-hour for the car when it works for 1 hour

E=P*T = (1.5 +30) *1 = 31.5 Watt.hour

The total power supported by battery is:

P=I*V =1.5*7.4 =11.1 Watt

Total Energy =P*T =11.1 Watt.hour

Since the battery runs the motors up to 20 mins, hence:

E=31.5* 0.33= 10.39 Watt.hour

Which is less than the energy supported by the battery (that came with the RC car).

3.5.2 Microcontroller

The used controller as shown in Figure 3.8 is an Arduino Nano, due to its size in comparison with other Arduino boards family, this one was a compatible with other project components that we have chosen.

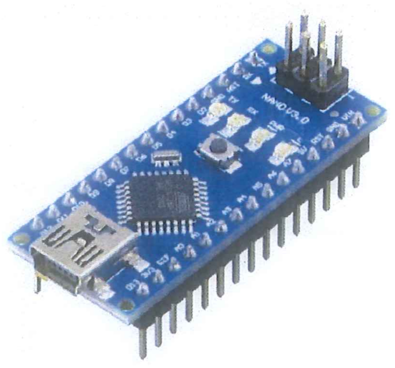


Figure 3.8: Arduino nano[10]

3.5.3 Communication

As mentioned in this chapter, the PC communicates wirelessly with the Arduino which is located on the car. Hence, three options show up here to communication part: Wi-Fi module, bluetooth and Xbee.

Wi-Fi sends data along large distance with a high speed; however, its weight may have an effect on our car. The weight is not that issue in the bluetooth option, however, it sends a smaller data along limited distance compared to Wi-Fi but the real issue with it is that, many times there might be a loss in the sending data which would damage the entire learning process. The Xbee did the trick here. It sends a good size of data without disconnection for about more than 91m and a compatible weight too [11].

3.5.4 Detection

Here we discuss the detection sensors to observe the car and other obstacles that are placed in the learning arena.

3.5.4.1 Ultrasonic and ladder

Ultrasonic and ladder sensors are close to each other, Ladder measures distance to a target using pulsed laser light and measuring the reflected pulses. Ultrasonic sends a high-frequency sound pulse then measures the time that the echo of the sound takes to reflect back.

These two sensors shown in Figure 3.9 will not be chosen since we do not have a rigid body to move around as a center however it is a point that has no observable height to be detected by these sensors.

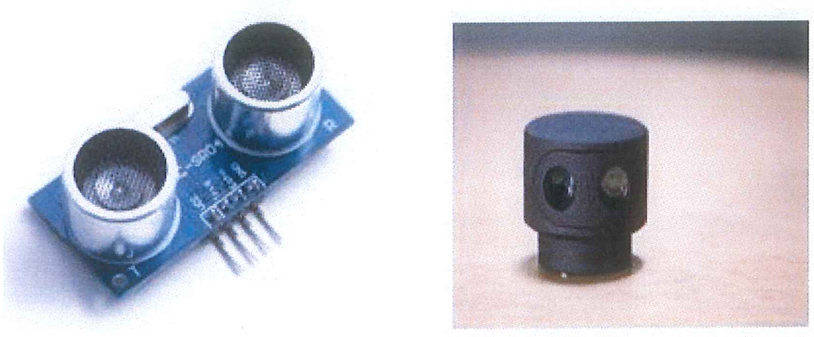


Figure 3.9: Ultrasonic and Ladder modules[12]

3.5.4.2 Infrared IR sensor

To protect the car from collisions with any obstacle or wall an IR sensor is used due to its good specification from high sensitivity to the objects and its accuracy then the low-cost property compared to ladder sensor. that shown in Figure3.10.

Figure 3.10:Infrared IR sensor[13]

3.5.4.3 Capture system

We have used a camera to measure the distance between the car and center point in addition to the orientation of the car too. The camera that we have chosen can cover the range of learning area that the car is moving on. This one is not mounted on the car so we do not need to care about its weight, only the resolution matters.

Two types of cameras have been used in Figure 3.10 and Figure 3.11 the first camera we used with an accuracy for about 720P HD was not shown the balls clearly due to its low resolution, however, the second one with a resolution of 1080P HD was a way better.

Figure 3.11: HD WEBCAM C270[14]

Figure 3.12: PRO WEBCAM[15]

3.5.5 Personal computer

For processing the camera images in a high speed, a PC was used to do this task, an i7 core processor with 8 RAM has been used in our PC, using a visual studio 2017 program to make the vision coding part, the processed images were generating quickly with a fast recognition time for any changing in the car position.

4

CHAPTER 4
Mathematical model and simulation

4.1 Introduction

This chapter demonstrates the robot kinematic model and is followed with the simulation, notes that we confirm a bicycle model to simplify the calculations.

4.2 Kinematic model

Kinematic analysis is one of several models that can represents the system mathematical model from its movement perspective, it is decoupled the equations from their dynamic such as torques and forces and its properties like mass and center of gravity to focuses on the velocity caused by these dynamics. Firstly, this section will start with an assumption of this system that the derivation will be based on later, then it will define with an appropriate way the definition of the vehicle's coordinate and its world coordinates as a framework to be used throughout the rest of this chapter and illustrate all other derivations grounded on kinematics aspect that will be used after that to design the controller of the vehicle.

At the beginning, it is used to assume a four-wheels vehicle such as this car-like vehicle with a bicycle approximation model, to simplify the system derivation either in kinematic or dynamic mode by taking the advantage of the similarity between the car sides, left and right. This bicycle model as shown in Figure 4.13 has a fixed rear wheel to the body and a front wheel rotate about its vertical axis in order to steer the vehicle.

The system has two frameworks that can be represented in, the first one$\{V\}$ is referred to the vehicle's frame which is called body coordinates, and the second frame $\{O\}$ represents the world coordinates [8]. From Figure 4.1, many variables in body coordinates can be demonstrated as below:

γ: is the steering angle of the front wheel, that makes the bicycle rotate about z-axis.

$\boldsymbol{\theta}$: represents the yaw angle or the heading of the whole vehicle.

L: is the vehicle's length, the longer this distance is, the wider the turning circle that the vehicle can make about some reference point.

R1 & R2: are the turning radius that rear and front wheels make respectively with the intersection of the dashed lines at the instantaneous center of rotation ICR, whereas the wheels cannot move along these dashed lines, and the vehicle make this point as its instantaneous reference to the steady state cornering that the car makes a path with a constant curvature surrounds it.

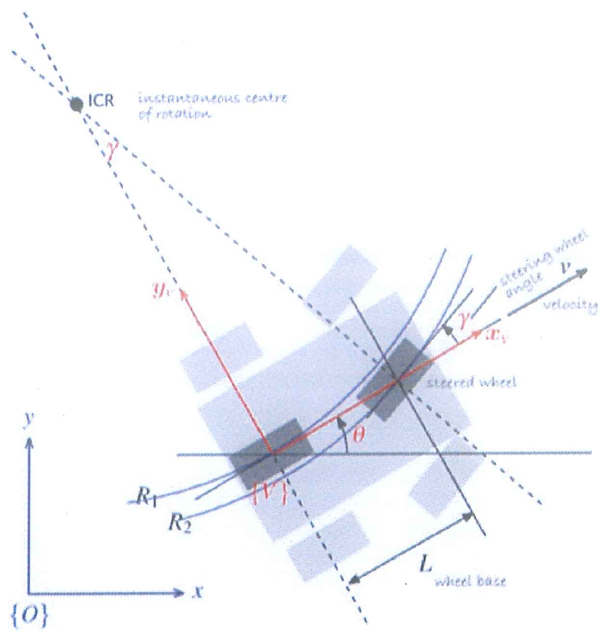


Figure 4.1: Kinematic bicycle model approximation of the vehicle

the velocities of the body frame are shown in Eqn (4.1), since mecanum wheels are not used here; the vehicle is roll only in the x-direction and has no slip slideway in the y-direction.

$$\dot{x}_B = v \tag{4.1}$$

$$\dot{y}_B = 0 \tag{4.2}$$

Since the motor is located in the rear part of the vehicle, hence the angular velocity of the vehicle is equal to:

$$\dot{\theta} = \frac{v}{R_1}, \text{ where } R_1 = \frac{L}{\tan\gamma} \tag{4.3}$$

It can be noticed that the turning circle would be increase by increasing the length of the vehicle. Also, the maximizing in the value of the steering angle would minimize the R_1value, but still constraints with steering mechanical limitations.

The equation of motion of the kinematic model for rear wheel can be represented by a combination of the velocities in the world frame $\{O\}$, and Eqn 4.3 to get:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\gamma} \end{pmatrix} = \begin{pmatrix} v\cos\theta & 0 \\ v\sin\theta & 0 \\ \frac{v}{L}\tan\gamma & 0 \\ 0 & \omega_r \end{pmatrix} \tag{4.4}$$

This equation represents the kinematic model that concerned with the velocities as shown in Eqn 4.4. $\dot{\theta}$ denoted to the turning rate or the angular velocity that can be measured by a gyroscope or a capture system. For the front wheel its kinematics model can be presented as in Eqn 4.5

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\gamma} \end{pmatrix} = \begin{pmatrix} v\cos\theta\ \cos\gamma & 0 \\ v\sin\theta\cos\gamma & 0 \\ \frac{v}{L}\tan\gamma & 0 \\ 0 & \omega_f \end{pmatrix} \tag{4.5}$$

4.3 Simulation

This section deals with the simulation of the vehicle from a kinematic perspective. Three cases have been highlighted here; the behavior of a bicycle model depending on some speed and orientation, move the robot to a point and lastly reinforce it to follow a desired trajectory which is in this project, is a circular one. Robotics toolbox has used to perform this simulation.

4.3.1 Bicycle model

The first case aims to detect the behavior of the bicycle while moving with some constant speed and steering angle at Figure 4.3 and Figure 4.4 for both front and rear wheels. The implementation of this model that has been derived in Eqn 4.4 which is shown in details at Figure 4.2, with two inputs entered into the system; the linear velocity v of the wheels and the steering angle γ [9]. In the model also, velocity, acceleration and steering angle limiters has been added to the system which came from the system constraints itself, as the mechanical limitations for the steering angle, likewise to the allowed speed of the car. The simulation done with 1 m/s speed and 0.3 rad, that is equal to 17.18 degrees, where the assumption made on maximum 18 degree that the wheel allows to rotate with.

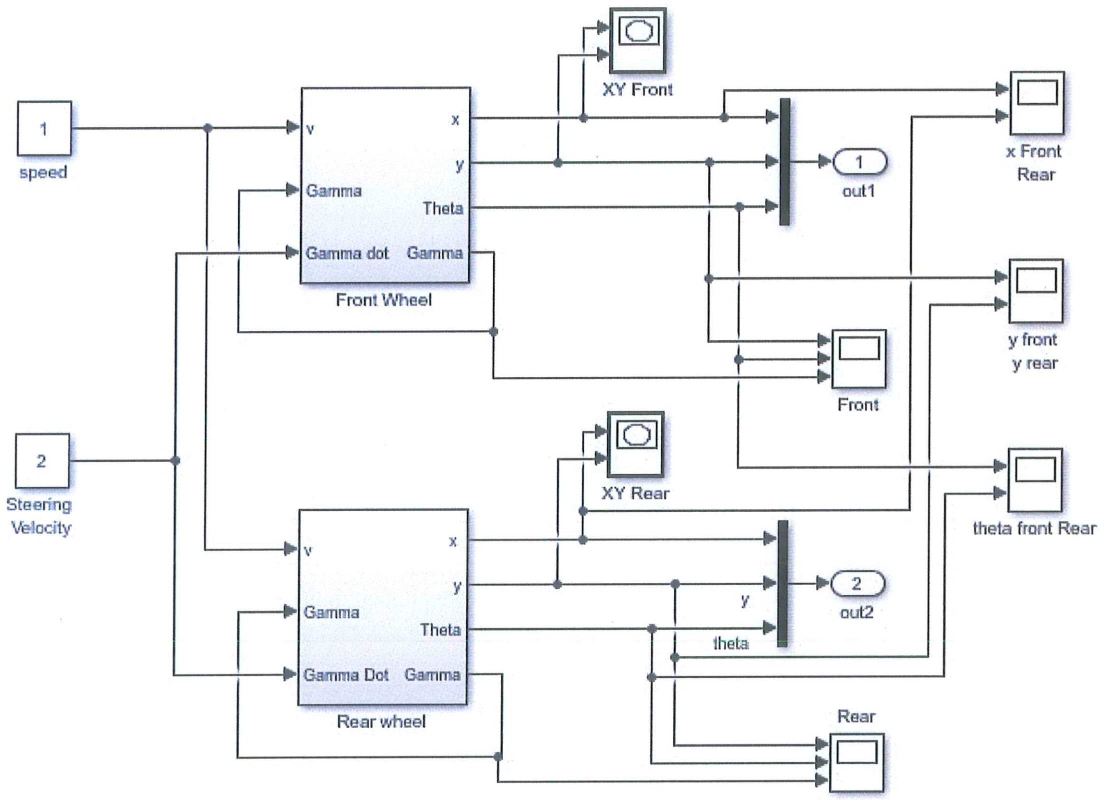


Figure 4.2: Bicycle model

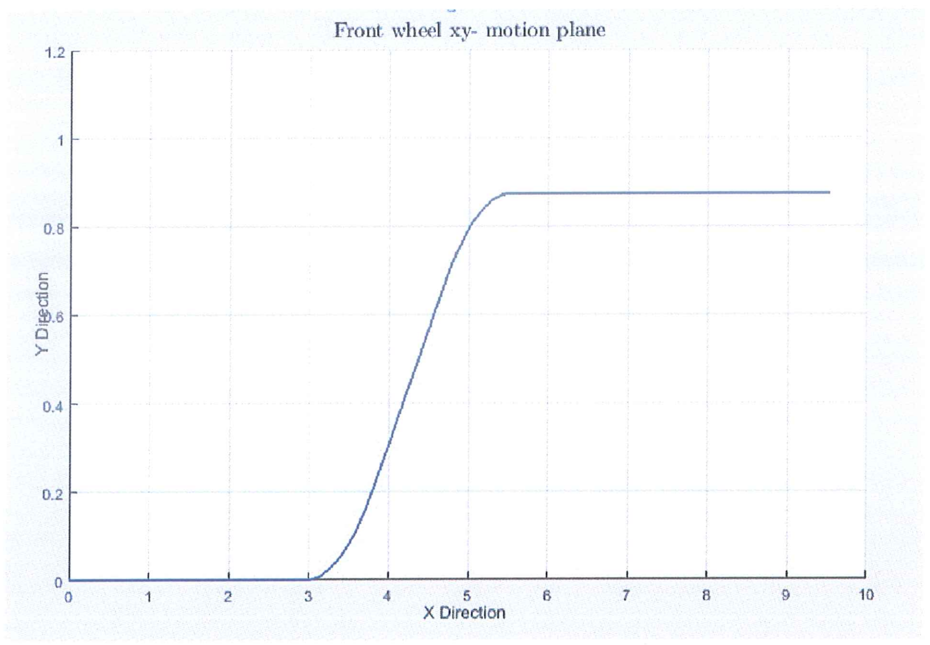


Figure 4.3: Front wheel motion in xy-plane

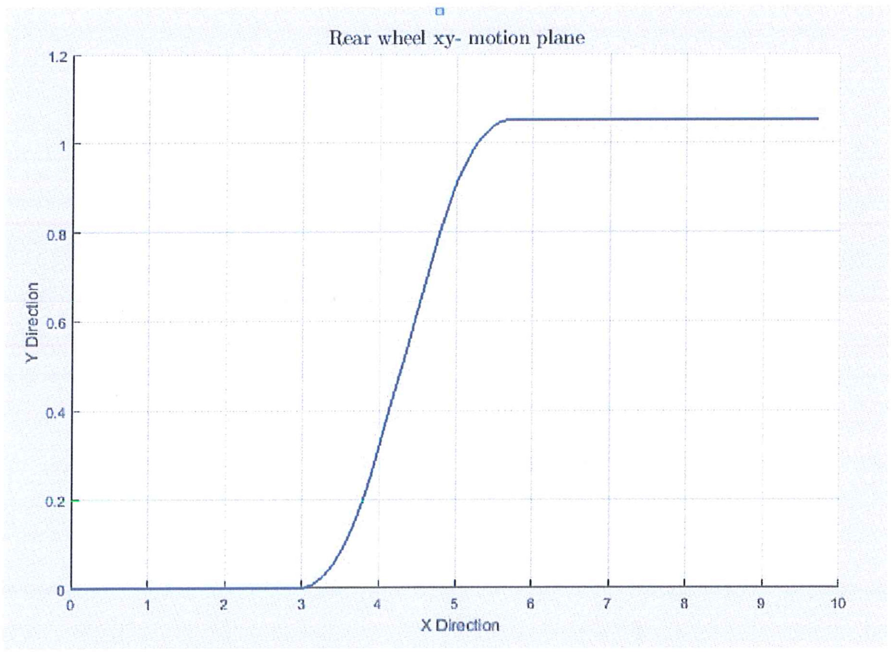


Figure 4.4: Rear wheel motion in xy-plane

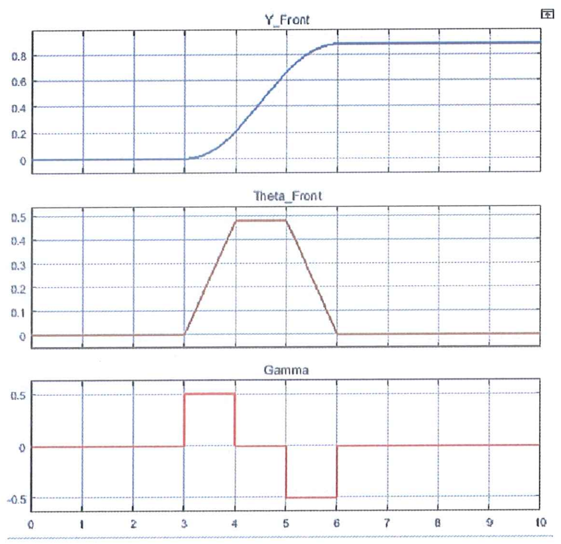


Figure 4.5: Front wheel response as a function of time at 1 m/s

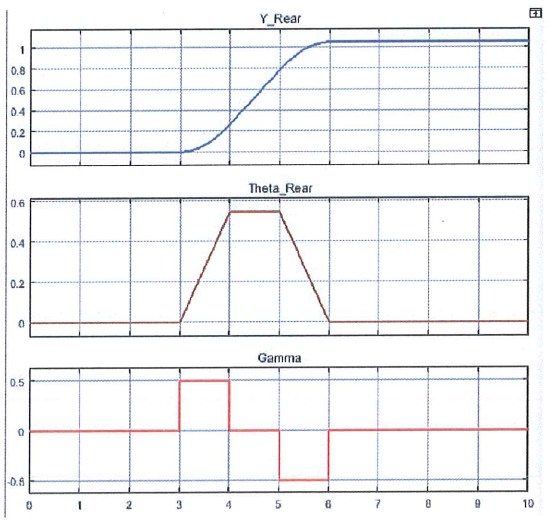


Figure 4.6: Rear wheel response as a function of time at 1 m/s

In Figure 4.5, the front wheel rotates with 0.49 rad, 1 m/s and reached 0.9 m in y-direction as shown in the first response, that transition action attached with changes in the yaw angle and the steering angle that shown in both second and third responses respectively in the same figure then goes back into its steady state. However, the positions from x and y were not indeed as desired, hence the next section would solve that issue.

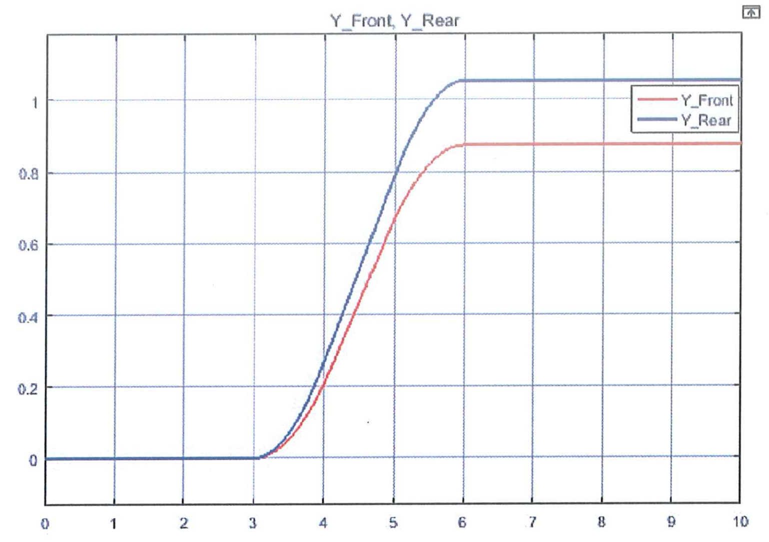


Figure 4.7: The distance in y direction for both front and rear wheel

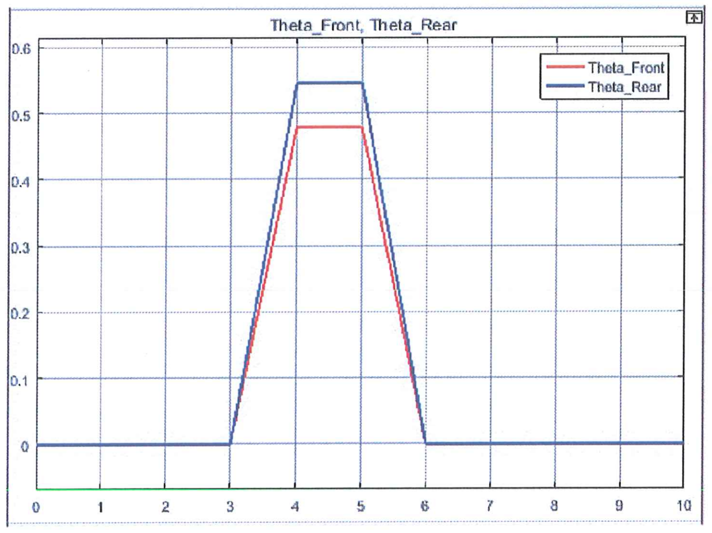


Figure 4.8: The sideslip angle for both front and rear wheel

4.3.2 Moving to a point

The consideration in that subsection is focused on how to move toward a goal point (x^*, y^*) in the plane, therefore two controllers here can be added to the system, one is used to control the speed and the other one to control the steering angle [8]. The speed controller is done by makes the robot's velocity proportional to its distance from the goal point.

$$v^* = k_{p_1}\sqrt{(x^* - x)^2 + (y^* - y)^2} \tag{4.6}$$

To steer toward the goal, the relative angle of the vehicle is,

$$\theta^* = \left(\frac{y^* - y}{x^* - x}\right) \tag{4.7}$$

Then be used to make a proportional controller for the steering angle, this controller makes the wheel turns toward its target.

$$\gamma = k_{p_2}(\theta^* \ominus \theta), \ \ k_{p_2} > 0 \tag{4.8}$$

$\ominus$: is a difference operator to get the angular difference between the target and the current value.

All these equations can be implemented by the model at Figure 4.9

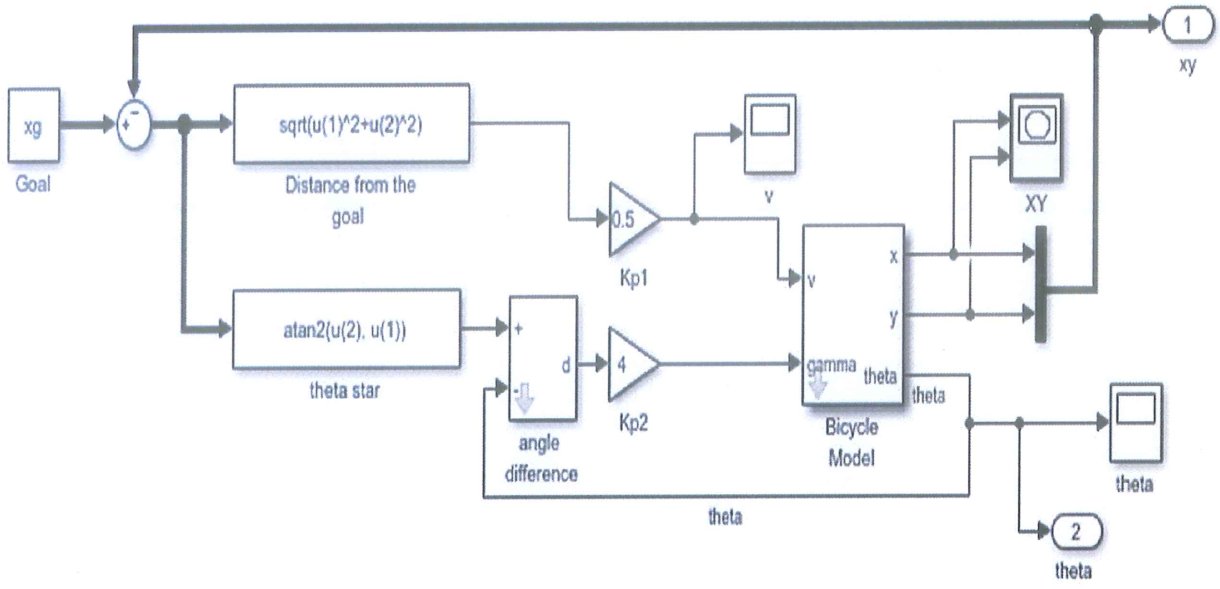


Figure 4.9: Proportional controlled bicycle model to move into a point

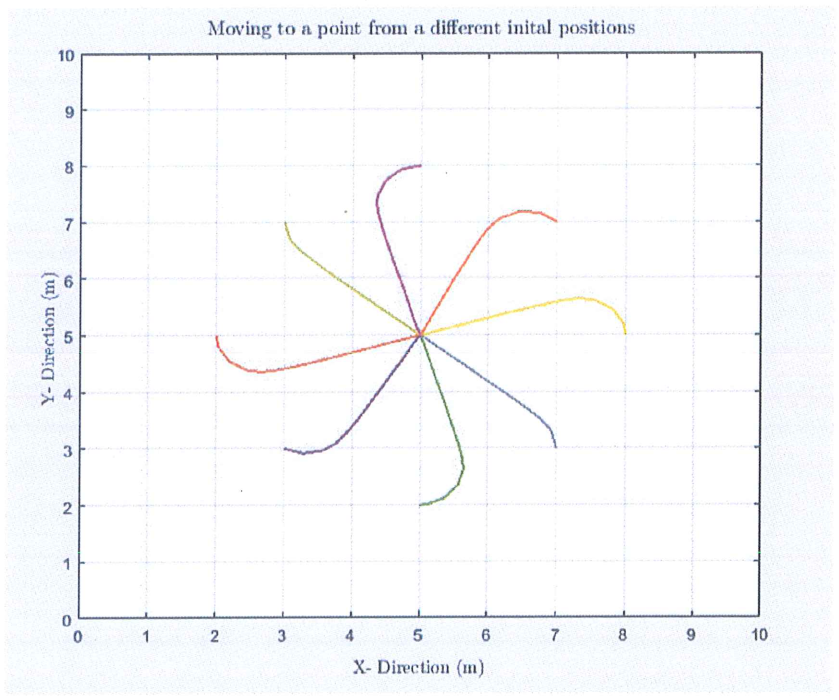


Figure 4.10: Response of moving toward a goal from different initial positions

The response in Figure 4.10 shows how the bicycle moved into a goal, which was in this case [5,5] from many starting poses (position and orientation) and how the controllers change the steering and velocity parameters to perform such paths to get the desired position. some of these initial poses makes the vehicle drive forward the goal as point [2,5] for instance, and another pose such as [8,5] made the vehicle drive backward to catch the target.

4.3.3 Following a circular trajectory

In this case, the model designed to be able to track some trajectory based on pure pursuit algorithm [12] this trajectory that the vehicles will follow might come from either real-time measurement from system sensors or a sequence of coordinates generated by a motion planner. In this case the goal $(x^*(t), y^*(t)\ )$ has updated along the path with a constant speed and keep the vehicle turn its steering angle with a proper angle to make sure that the path is correctly tracked.

This case is similar to the previous one, the difference is that the target now is moving so by maintains distance d^* behind the pursuit point, the formulated error will be [1].

$$\mathrm{e} = \sqrt{(x^* - x)^2 + (y^* - y)^2} - d^* \qquad \textbf{(4.9)}$$

To be a certain that the car follows the path without steady state error; PI controller has been used to control the velocity of the vehicle.

$$v^* = k_p \mathrm{e} + k_I \int \mathrm{e}\, \mathrm{d}t \qquad \textbf{(4.10)}$$

The second controller used to steer the vehicle toward the moving target, by insert Eqn 4.7 into the proportional controller of the steering angle.

$$\gamma = k_{p_2}(\theta^* \ominus \theta)\,, \qquad k_{p_2} > 0 \qquad \textbf{(4.11)}$$

The final model at Figure 4.11 demonstrates the equations above.

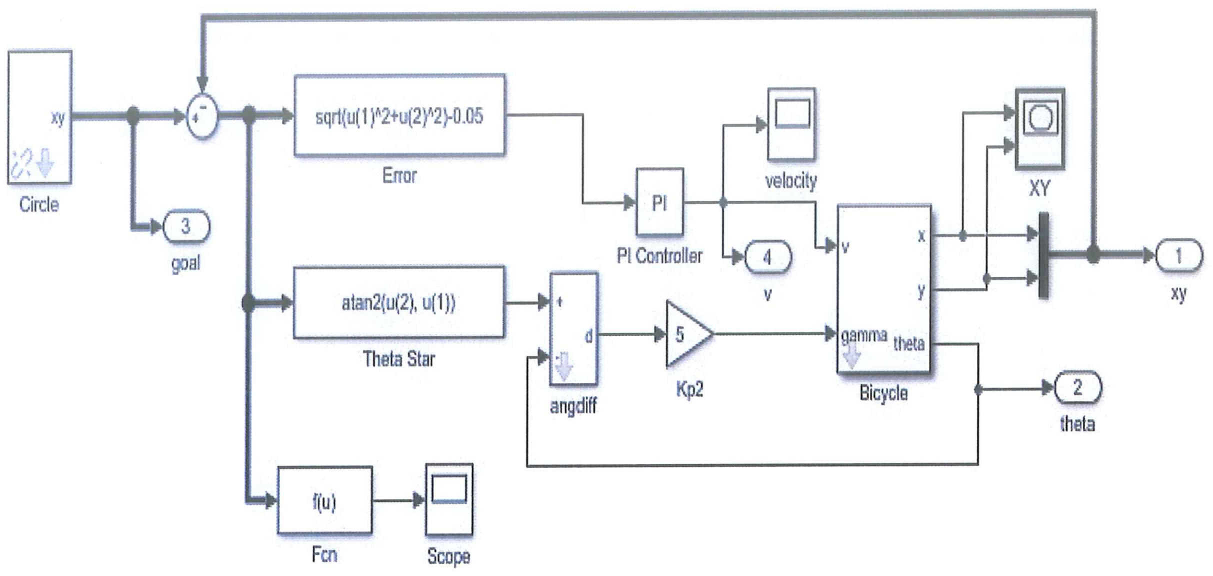


Figure 4.11: Bicycle model of following a circular trajectory using pursuit algorithm

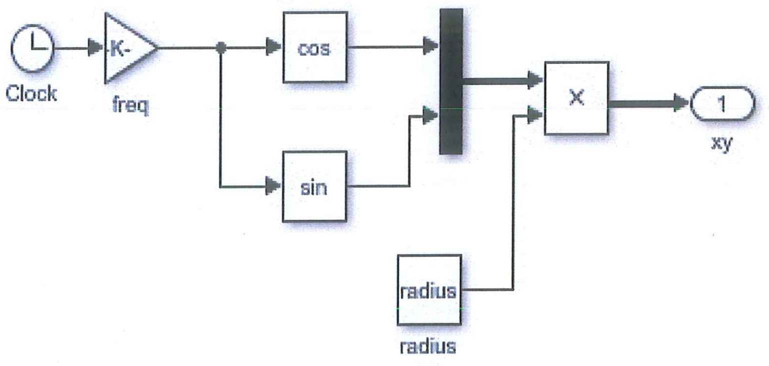


Figure 4.12: Circle model

By tuning k_p=1 and k_I=0.5, with 0.1 Hz frequency and 1 m radius of cornering; the response at Figures 4.13 has resulted.

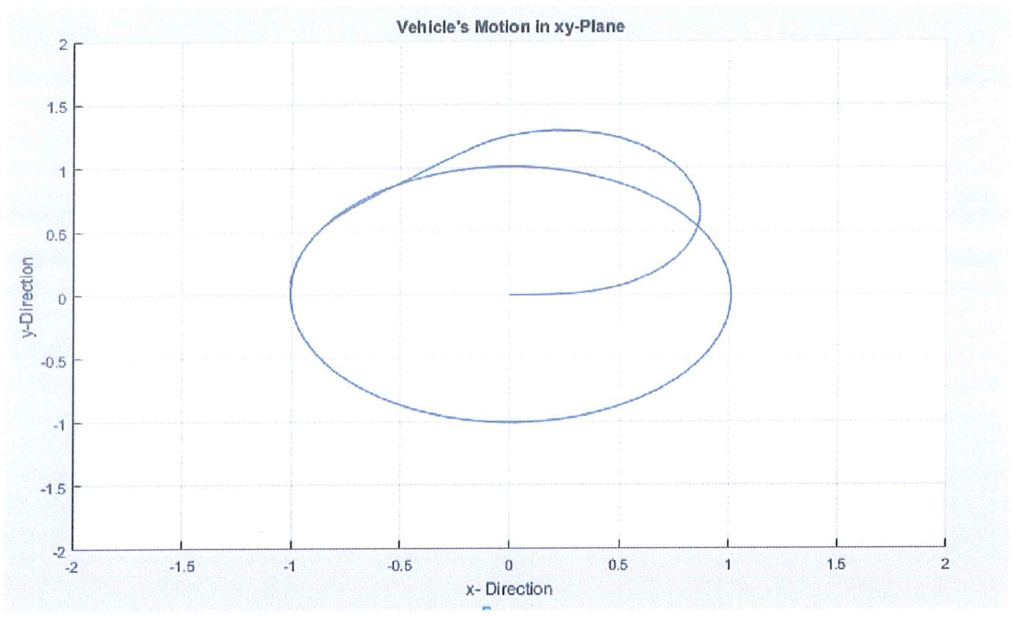


Figure 4.13: Following trajectory in xy-plane.

In Figure 4.13 the vehicle starts moving from an initial position then tend to regulate its path to accomplish the desired cornering radius which it was 1 meter.

5

CHAPTER 5
Implementation and validation

5.1 Introduction

5.2 Project architecture

5.1 Introduction

This chapter concerns with synergic employment of both the principles and the tools that have been chosen in the theoretical part to work upon in that project. Figure 5.1 shows the main categories from hardware and software that the accomplished work classified based on, through this chapter, the assembled hardware illustrated first, follows by the software architecture.

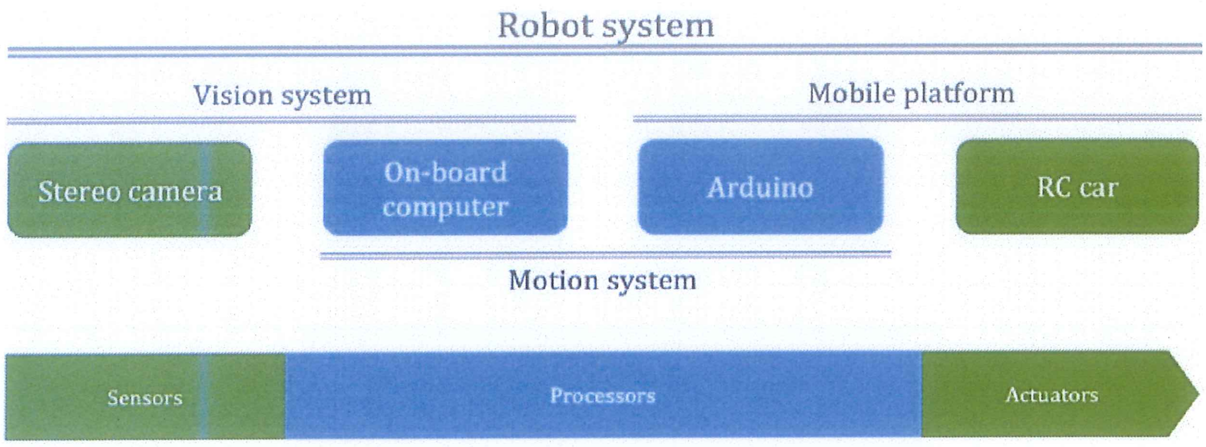


Figure 5.1: The project architecture

5.2 Project Architecture

This section shows the hardware parts of the car from actuators and drivers.

5.2.1 Hardware

The chosen car consists of two actuators, Direct current (DC) motor and servo, both for throttle and steering respectively. A lithium battery also attached into the car as well to insure the capability of motors to working at their rated values throughout the learning process on a real land. Figure 5.2 shows the gathering components altogether.

To control the vehicle an Arduino has been added, hence, we have provided each motor with its compatible driver in order to interfacing the motors with microcontroller, because of a primary requirement for the operation of the controller is low voltage and small amount of current. But the motors require a high voltage and current for its operation.

Figure 5.2: Car actuators[7]

The DC motor driver that represented in Figure 5.3 is used to interface the motor with the Arduino, the one provided with the car itself has no serial number or any name of its components, no other references as well, thus the circuit in Figure 5.3 has been made to accomplish this task.

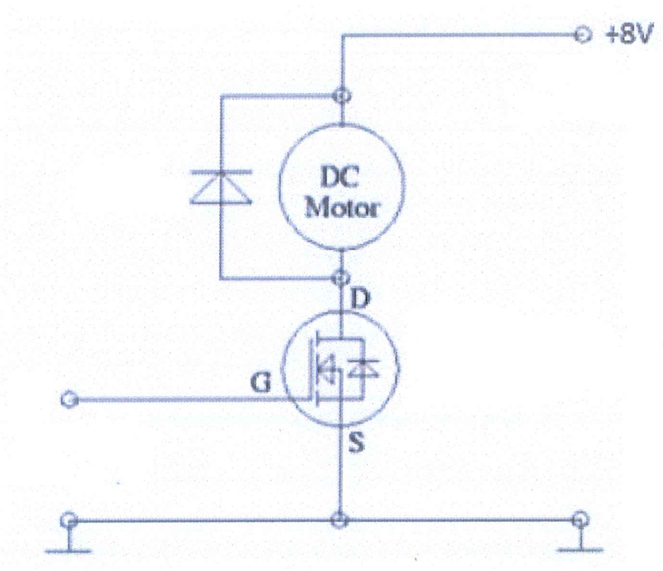


Figure 5.3: BT-2 high current 30A DC motor driver

For the servo motor, the provided one with the car consists of five wires, after making some measurements on the wires we have found that this motor has no control or feedback driver that has an amplifier integrated inside it to calculate the voltage difference between motor terminals which causes the motor to rotate in the desired direction. To solve that issue, we have taken another servo motor driver from Figure 5.4 and attached it into this motor, in that way we were able to take three wires Vcc, Gnd and a pulse wire. To connect them with the Arduino in order to control the servo properly. The overall connection shown in Figure 5.5.

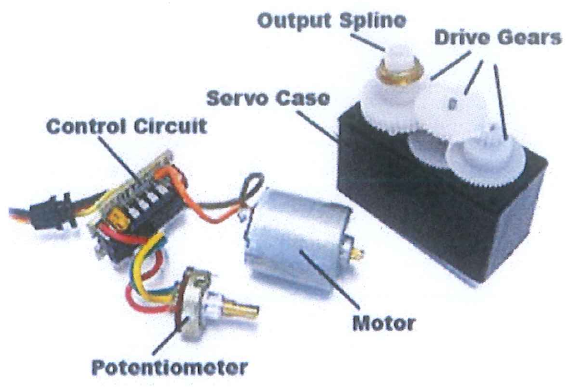


Figure 5.4: Servo structure.

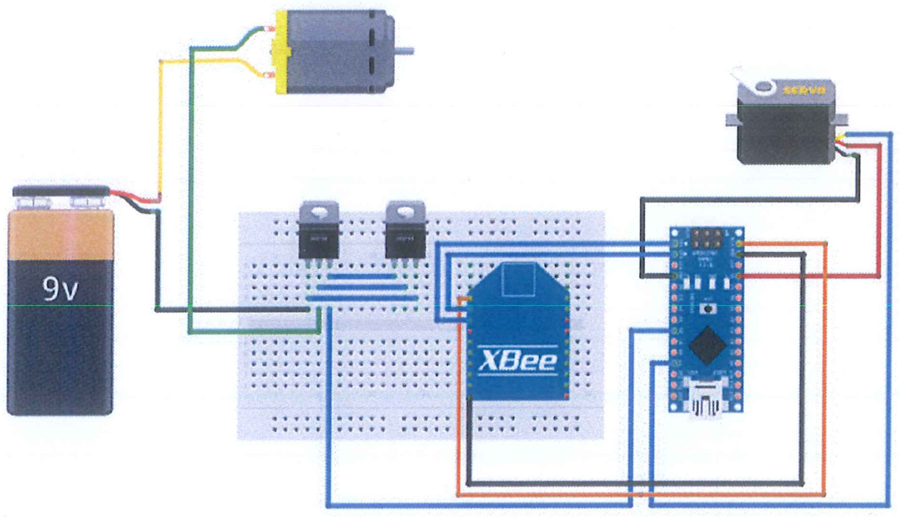


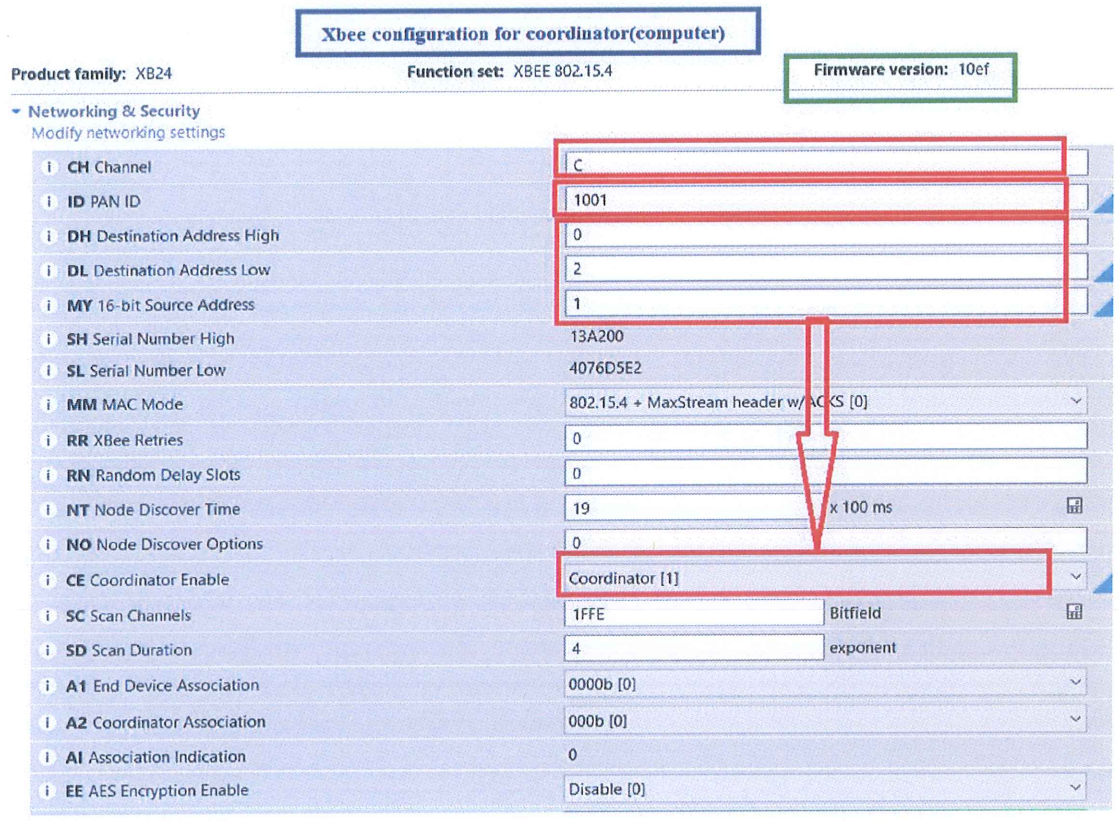
Figure 5.5: The component connection in the car

5.2.2 Software

This section shows the programs and the configurations that done in the practical work.

1. Xbee configuration

To communicate the PC with Arduino, we have used an Xbee. The configuration set from two sides the Arduino to receive and the PC to transmit, the transmission configuration shown in Figure 5.6, and the receiving part of configuration has been illustrated in Figure 5.7.



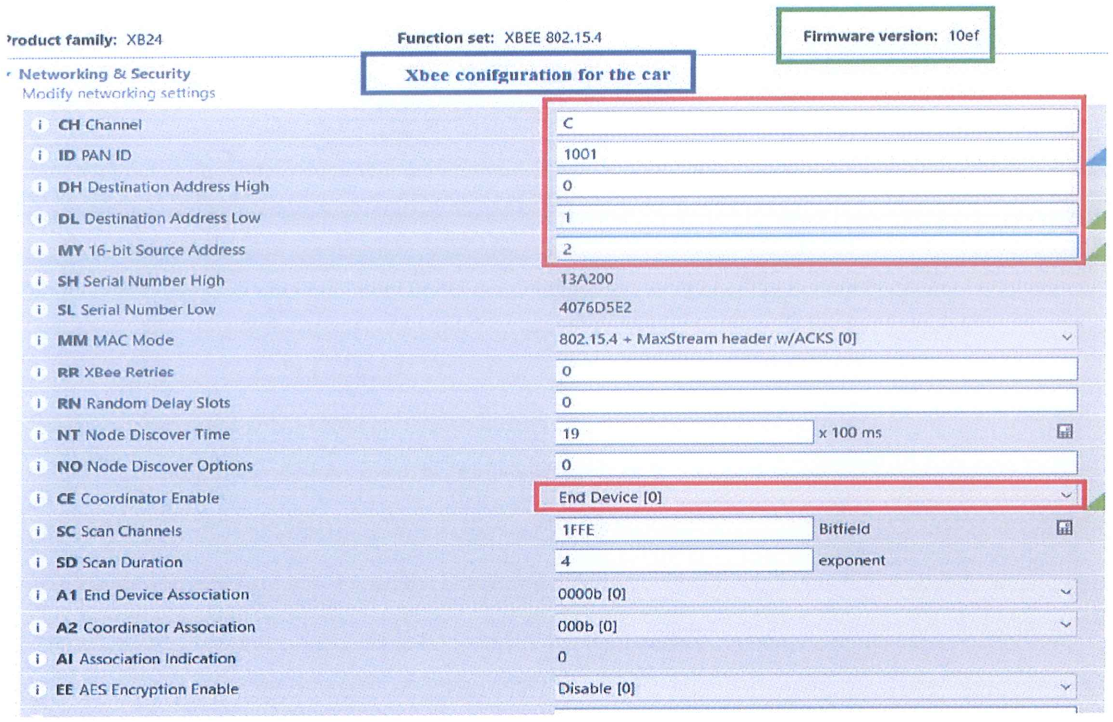
Xbee configuration for coordinator(computer)

Product family: XB24 **Function set:** XBEE 802.15.4 **Firmware version:** 10ef

Networking & Security
Modify networking settings

CH Channel	C
ID PAN ID	1001
DH Destination Address High	0
DL Destination Address Low	2
MY 16-bit Source Address	1
SH Serial Number High	13A200
SL Serial Number Low	4076D5E2
MM MAC Mode	802.15.4 + MaxStream header w/ACKS [0]
RR XBee Retries	0
RN Random Delay Slots	0
NT Node Discover Time	19 x 100 ms
NO Node Discover Options	0
CE Coordinator Enable	Coordinator [1]
SC Scan Channels	1FFE Bitfield
SD Scan Duration	4 exponent
A1 End Device Association	0000b [0]
A2 Coordinator Association	000b [0]
AI Association Indication	0
EE AES Encryption Enable	Disable [0]

Figure 5.6: Xbee configuration for coordinates -computer side.



roduct family: XB24   Function set: XBEE 802.15.4   Firmware version: 10ef

Xbee conifguration for the car

Networking & Security
Modify networking settings

CH Channel	C	
ID PAN ID	1001	
DH Destination Address High	0	
DL Destination Address Low	1	
MY 16-bit Source Address	2	
SH Serial Number High	13A200	
SL Serial Number Low	4076D5E2	
MM MAC Mode	802.15.4 + MaxStream header w/ACKS [0]	
RR XBee Retries	0	
RN Random Delay Slots	0	
NT Node Discover Time	19	x 100 ms
NO Node Discover Options	0	
CE Coordinator Enable	End Device [0]	
SC Scan Channels	1FFE	Bitfield
SD Scan Duration	4	exponent
A1 End Device Association	0000b [0]	
A2 Coordinator Association	000b [0]	
AI Association Indication	0	
EE AES Encryption Enable	Disable [0]	

Figure 5.7: Xbee configuration for Arduino side.

2. Car communication testing

This windows application in Figure 5.8 has been made firstly to check the speed of connection using the Xbee by taking orders from the PC application to send the commands into Arduino that is placed on the car in this test, it shows the options of moving direction of the servo in addition to the magnitude of that motion whether it was speed or steering angle.

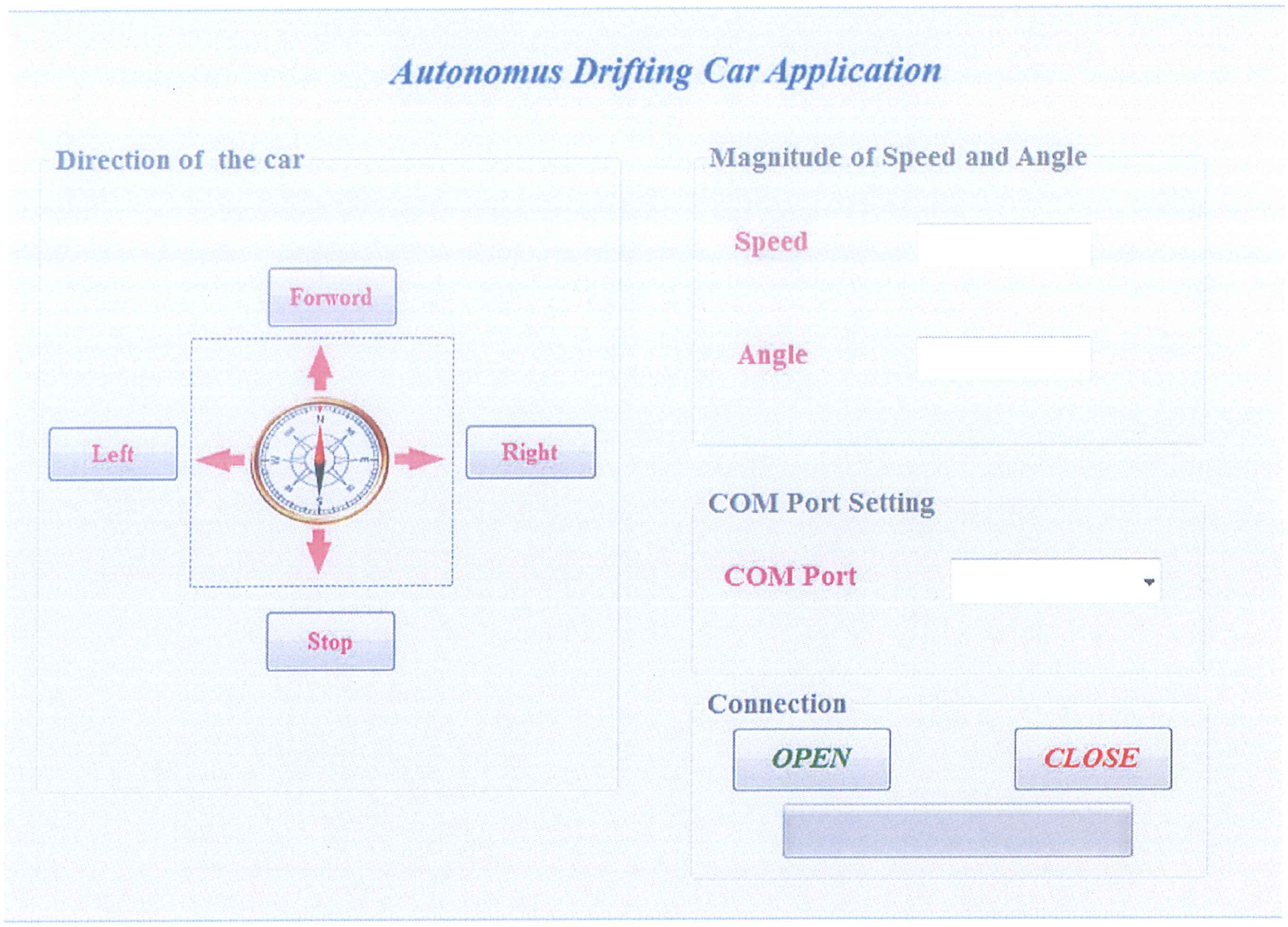


Figure 5.8: Testing application.

3. Computer vision

The chosen camera was fixed above 2.4 m from the area that the car moving on, to be able to detect the balls which are placed on the car that is shown in Figure 5.9, the taken photos are converted into their binary format to make the observation of balls counters much easier and less noisy, this process called *Thresholding* as shown in Figure 5.11.

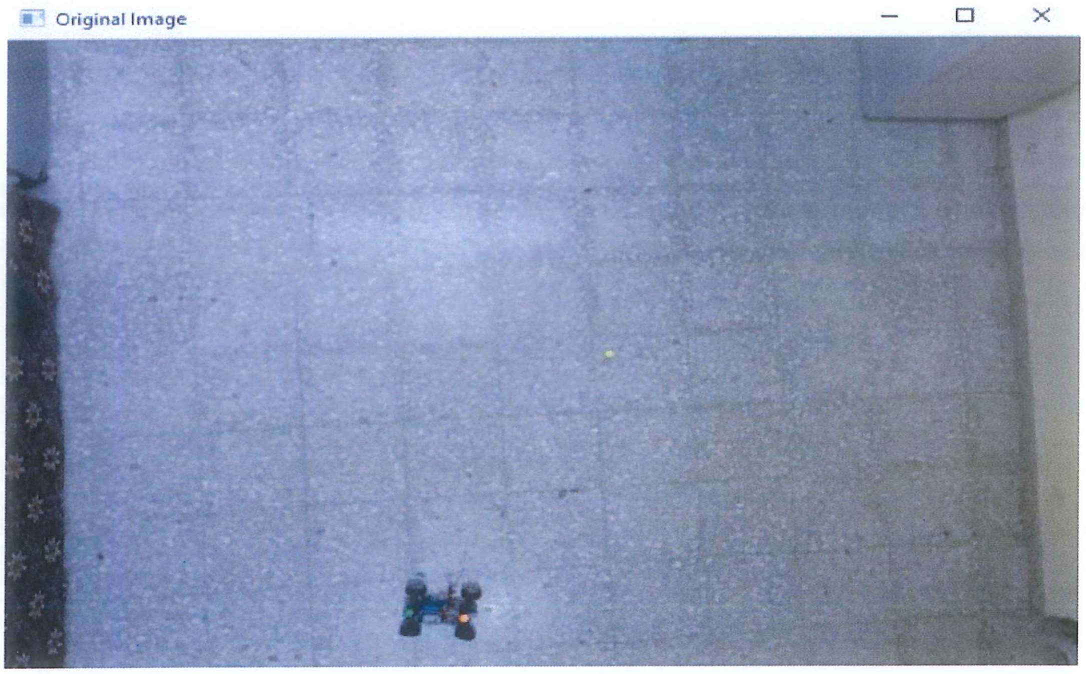


Figure 5.9: The original image taken.

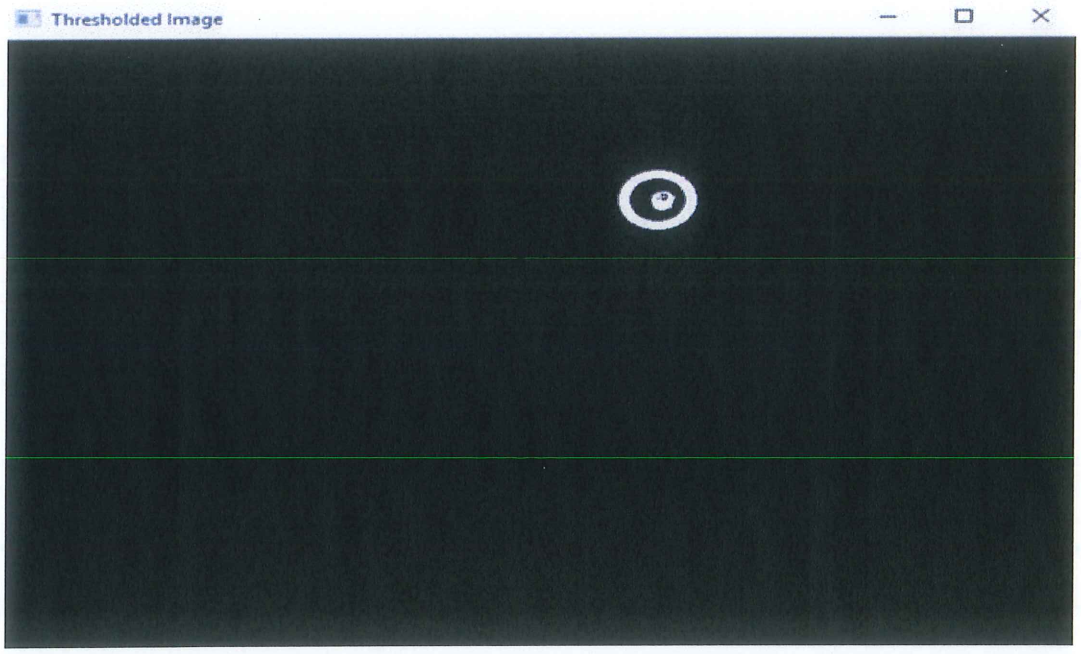


Figure 5.10: Detect the center point.

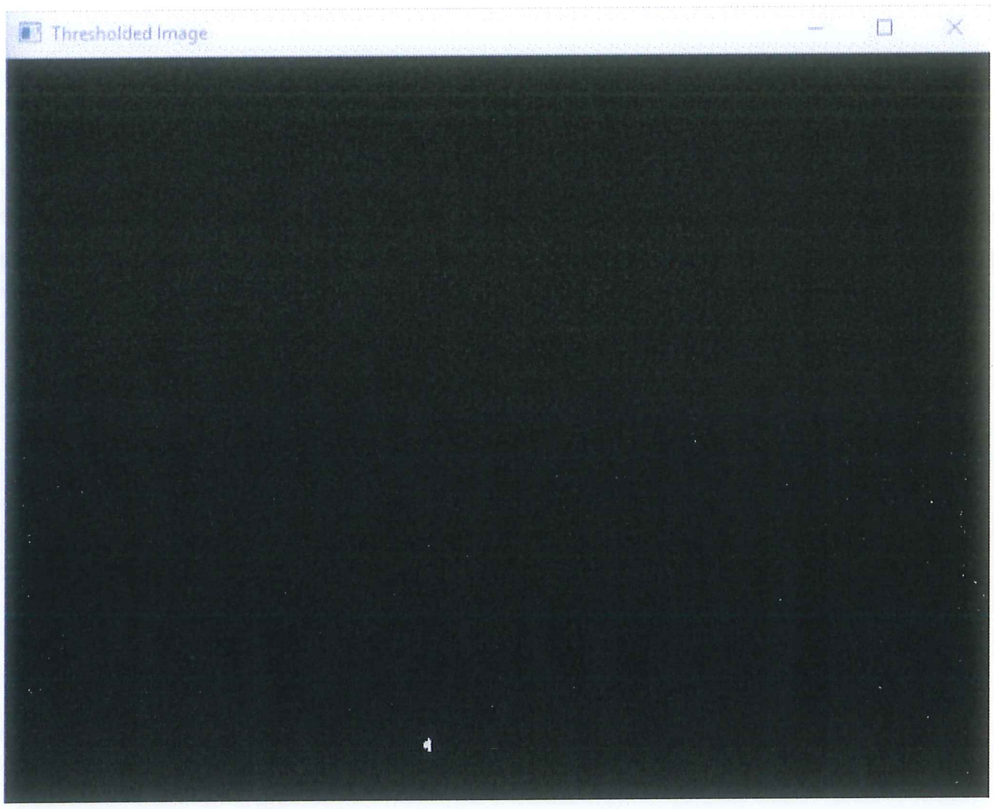
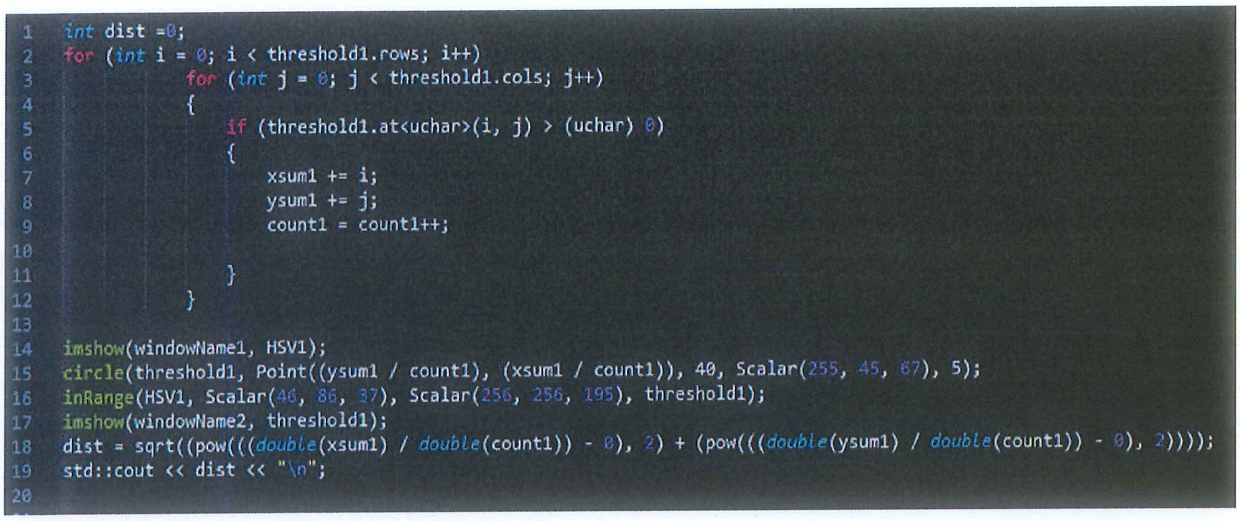


Figure 5.11: The orange ball- front ball

Once we are able to detect the object contour, the centroid can be found instantaneously as in Figure 5.10. Thus, we are able to detect the distance between any point on the car and the center point. That what the 18th line in Figure 5.12 shows clearly, after finding the centroid of the balls, two distances have to be calculated in order to determine the orientation of the car. The first distance is between the center point and the front ball and the second distance is calculated between the center and the back ball, now the difference between these two distances was sent to the microcontroller in Figure 5.13 then the Arduino use this value as a reward inside the Q-Learning algorithm.

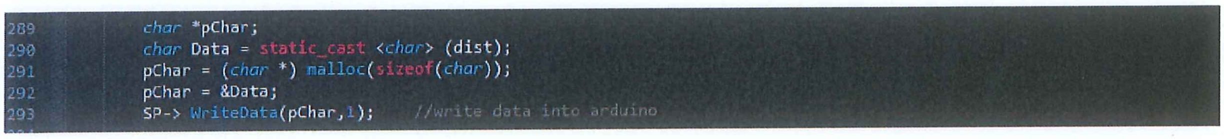


```
int dist =0;
for (int i = 0; i < threshold1.rows; i++)
            for (int j = 0; j < threshold1.cols; j++)
            {
                if (threshold1.at<uchar>(i, j) > (uchar) 0)
                {
                    xsum1 += i;
                    ysum1 += j;
                    count1 = count1++;

                }
            }

imshow(windowName1, HSV1);
circle(threshold1, Point((ysum1 / count1), (xsum1 / count1)), 40, Scalar(255, 45, 67), 5);
inRange(HSV1, Scalar(46, 86, 37), Scalar(256, 256, 195), threshold1);
imshow(windowName2, threshold1);
dist = sqrt((pow(((double(xsum1) / double(count1)) - 0), 2) + (pow(((double(ysum1) / double(count1)) - 0), 2))));
std::cout << dist << "\n";
```

Figure 5.12: Determine the centroid of the detected object.

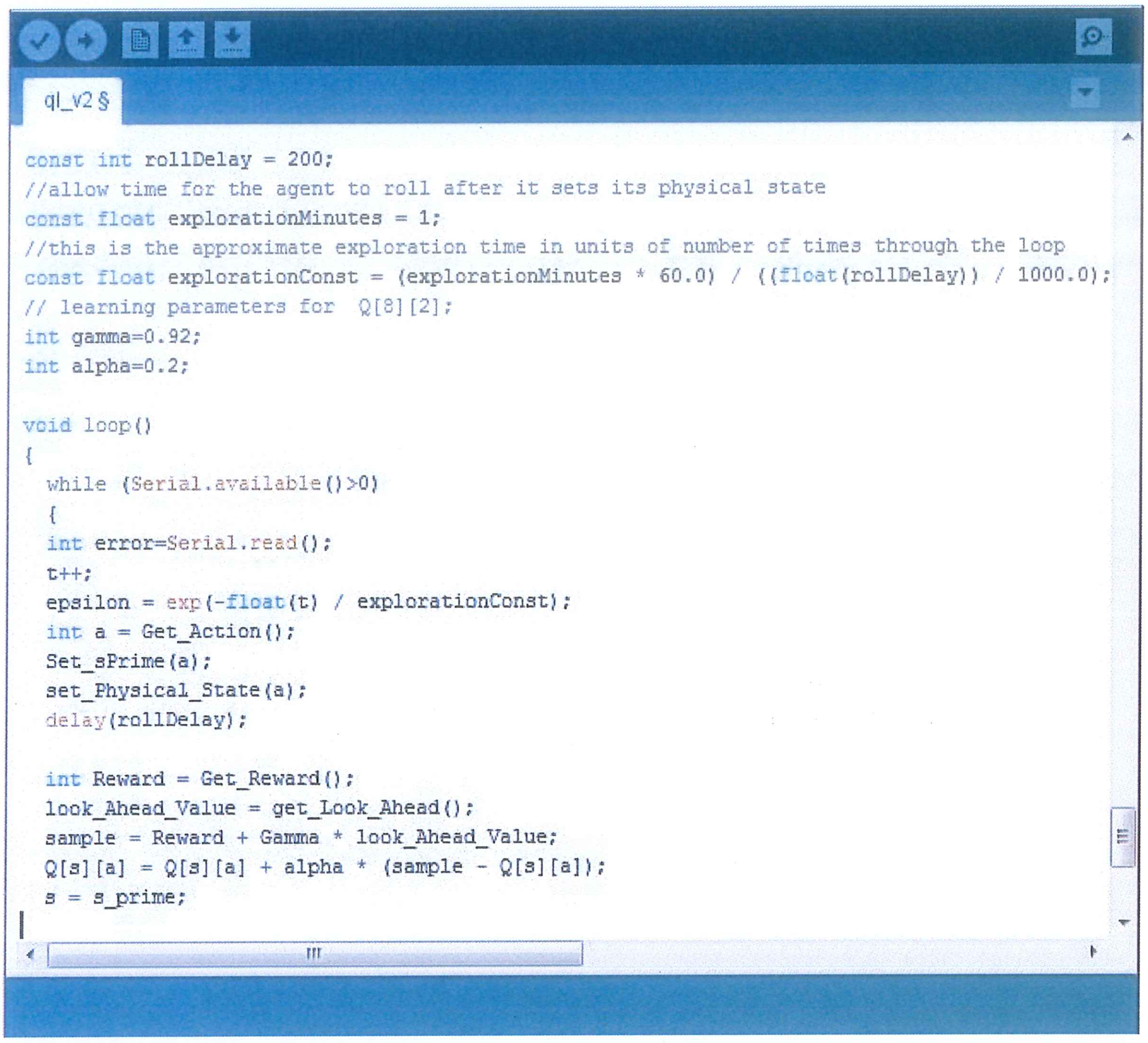


```
        char *pChar;
        char Data = static_cast <char> (dist);
        pChar = (char *) malloc(sizeof(char));
        pChar = &Data;
        SP-> WriteData(pChar,1);    //write data into arduino
```

Figure 5.13: Sending distance data into the Arduino.

4. Q-Learning

As shown in Figure 5.14.The function set_Physical_State responsible of adding or subtracting five degrees for servo motor and 20 PWM for throttle. The reward function receives the difference of distance measured by the camera between the center point and the front ball placed on the car, if that difference was in range of [0-5] then the reward would be increased otherwise its value going to decrease, so that the learning algorithm realizes how bad it was taking that current action through that state.



```
const int rollDelay = 200;
//allow time for the agent to roll after it sets its physical state
const float explorationMinutes = 1;
//this is the approximate exploration time in units of number of times through the loop
const float explorationConst = (explorationMinutes * 60.0) / ((float(rollDelay)) / 1000.0);
// learning parameters for  Q[8][2];
int gamma=0.92;
int alpha=0.2;

void loop()
{
  while (Serial.available()>0)
  {
  int error=Serial.read();
  t++;
  epsilon = exp(-float(t) / explorationConst);
  int a = Get_Action();
  Set_sPrime(a);
  set_Physical_State(a);
  delay(rollDelay);

  int Reward = Get_Reward();
  look_Ahead_Value = get_Look_Ahead();
  sample = Reward + Gamma * look_Ahead_Value;
  Q[s][a] = Q[s][a] + alpha * (sample - Q[s][a]);
  s = s_prime;
```

Figure 5.14: The implementation of Q-learning algorithm

5. Results

The project objectives have been accomplished by firstly using balls to detect the car position and the center point of rotation with a high-quality camera. Two balls were placed on the car with two different colors in both front and rear spots and a processing code for the taken images was also made to distinguish the colors of balls hence, determine their relative distances and calculate the difference between the center point and one of the balls on the car. A Q-Learning algorithm is made to be able to change the steering angle of the car to achieve the circular motion by adding or subtracting eight degrees each time step whereas the allowed range of angles that the servo can make is 55-122 degrees due to its mechanical constraints. The whole process is gathering between the Q-learning algorithm in the microcontroller and the vision system by inserting the output of the camera which is the difference distance that we mentioned before into the Q-learning algorithm as a continuous updated value to maintain the reward. This operation of learning was done in a limited area which was a challenging part due to the random motion at the beginning of learning that might drive the car out of the camera range. After a number of experiments or episodes of learning at the beginning, the car starts to learn how to drive correctly, firstly its motion tend to be some random movements and that is why the ultrasonic was good in that part to ensure the car will not be harmed itself by crashing with the wall during the learning process. After about five minutes it followed a track that is looked like a circle this path was corrected after seven minutes to be a circle with 1.7 m diameter.

The algorithm of reinforcement learning has been made using C language on the microcontroller. The used microcontroller was arduino due to its familiar environment of programming and to different size of board that yields to decrease the putting weight on the car. Whereas it pins used as follow:

1- Two pins for Bluetooth
2- Single PWM pin to control the motor speed which was fixed in our experiments
3- Single pin used for the servo motor
4- Double pins for the ultrasonic sensor to protect the car from crashing the walls

The issues that we have faced in our project was due to the high current in the car motor so we have used a mosfet however, a high reversed current has generated and there was no diode able to afford it because of the fast speed of discharging, so that circuit from resister and inductor has been made to reduce the amount of the reversed current that damages the diode. Another issue was in vision side, we were not able to detect the balls edges due to their size so we have replaced them with bigger one. Also, the Xbee was not sending the negative values from the camera code to the Arduino hence we have replaced it with a Bluetooth. The ultrasonic sensor was used to turn the motor off whenever it has a distance of 10 cm from wall. An IR sensor has a low response to detect the speed of motor rotation so we were not able to use it to measure the speed of the motor likewise of the rotary encoder its drawbacks was due to its weight and size that the car has no space to place it on.

6

CHAPTER 6
Summary and future work

6.1 Summary

6.2 Future work

6.1 Summary

In this project, a new approach of mobile robot learning is presented using a reinforcement learning as a method of machine learning. The RL algorithm helped in solving the problem of motion planning without having an explicit dynamical model of the robot, besides exploring the unknown environment that the robot is located in, this method has successfully learned how to make the desired trajectory though a process of reward and punishment during an exploration period.

Our focus was assigned into one of the branches of RL algorithms which is Q-Learning. This algorithm has accomplished an efficient learning not only in regard to the taken learning time which was less than 10 minutes, but also about the accuracy of the learned trajectory in comparison with the desired one.

6.2 Future work

In this project an autonomous car has learned to move in some path that we assumed it as circular track. The environment that the robot was exploring through has some limitations due to its space means that we were not able to make the car move in a larger paths or trajectory due to the fixed cameras above the arena. In future work we like to extend that area to mimic the real vehicle driving conditions. Besides, the drift motion may be accomplished with powerful motors with their driver and a bigger car than we have used in our project, to make sure that it will carry the whole sensors and battery without affecting its learning process. This drifting could be around some fixed point or another moving point in more challenging way. In regard to the capture system a Wii camera can be chosen instead of our used camera which has an advantage over ours that it has no need for image processing, besides that it can be cover more area easily by align it only, which is on the contrary of the normal camera that has to be calibrated again. The sensors inside the Wii such as a gyroscope and accelerometer can be used too as an alternative way to calculate the speed and the orientation of the car.

Appendix A: Q-Learning Code

```
include <servo.h>
#define In1 10            // if you put H-bridge
#define In2 11
#define PWM 6

Servo Ourservo;
float New_Distance = 0;
float Old_Distance = 0;
float Fn_Reward=0;
float final_Angle;
float final_Speed;
float epsilon;
int  alpha = 0.01;
int  Gamma = 0.92;
int  error =555555;

//This servo moves up with increased angle
const int Position_States = 3;
float Initial_Angle = 95;
float Servo_Max = 110;
float Servo_Min = 80;
float Delta_Servo = (Servo_Max - Servo_Min) / (float(Position_States) - 1.0); //The change in
servo's angle when an action is performed on it
int  s1 = int((Initial_Angle - Servo_Min) / Delta_Servo) - 1;        //This is an integer between
zero and Position_States-1 used to index the state number of servo
float delayTime1 = 2.5 * Delta_Servo;  ////that can be change -time i mean-
//throtle "DC motor"
const int Speed_States = 3;
float Initial_Speed = 0;
float Speed_Max = 70;
float Speed_Min = 60;

float Delta_Speed = (Speed_Max - Speed_Min) / (float(Speed_States) - 1.0); //The change in dc speed when an action is performed on it

int   s2 = (int(abs(Initial_Speed - Speed_Min) / Delta_Speed) - 1)-11;          //This is an integer between zero and Speed_States-1 used to index the state number of dc motor

float delayTime2 = 2.5 * Delta_Speed;

//////______////

const int Total_States  = 9;

const int Total_Actions = 4;

float Q[Total_States][Total_Actions;[

int s = int(s1 * Speed_States + s2;(

int sPrime = s;

void setup ()

{

 pinMode(PWM, OUTPUT;(

 pinMode(In1, OUTPUT;(

 pinMode(In2, OUTPUT;(

 Ourservo.attach(9;(

 Ourservo.write(Initial_Angle;(

 analogWrite (PWM, Initial_Speed;(

 digitalWrite(InA1, LOW;(

 digitalWrite(InB1, LOW;(

 Serial.begin(9600;(}

{ int Get_Reward} ()

///////////////////////////////////////////////////////////

 //*/Delta_Distance = Old_Distance - New_Distance

              //if (abs(Delta_Distance) < ... || abs(Delta_Distance(.... < (

               //Delta_Distance = 0;     //***Back///***

*/Old_Distance = New_Distance;

*/if (Old_Distance == New_Distance
    Fn_Reward;++
  elsex
    Fn_Reward/*;--
 if (error <5)
  Fn_Reward;++
  else
 Fn_Reward;--
   return Fn_Reward;
 {
 int Get_Action()
 }
  int   action;
  float value;
  float Max_value = -10000000.0;
  int   Max_action;
  float Rand_value;
  int   allowed_Actions[4] = { -1, -1, -1, -1;
  boolean randomActionFound = false;
//  find the optimal action.
 //  Exclude actions that take you outside the allowed-state space.
  if ((s1 + 1) != Position_States(
 }
   allowed_Actions[0] = 1;
   value = Q[s][0;[
   if (value > Max_value(
 }
     Max_value = value;
     Max_action = 0;
 {  {
  if ((s1 != 0(
}
 allowed_Actions[1] = 1;
  value = Q[s][1;[
  if (value > Max_value(
}
    Max_value = value;
    Max_action = 1;
{
{
 if ((s2 + 1) != Speed_States(
}
 allowed_Actions[2] = 1;
  value = Q[s][2;[
  if (value > Max_value(
}
    Max_value = value;
    Max_action = 2;
{
{
 if ((s2 !=0)
 {
 allowed_Actions[3] = 1;
  value = Q[s][3;[
  if (value > Max_value(
}
    Max_value = value;
    Max_action = 3;
{
{
// implement epsilon greedy
```

```
  Rand_value = float(random(0, 101;((
              if (Rand_value < (1.0 - epsilon) * 100.0(
// }choose the optimal action with probability 1-epsilon
  action = Max_action;
{
else
}
  while (!randomActionFound(
}
     action = int(random(0, 4));      //otherwise pick an action between 0 and 3 randomly (inclusive),
but don't use actions that take you outside the state-space
     if (allowed_Actions[action] ==1 )
       randomActionFound = true;
   }
}

// action = aMax;
  return (action);
}
void set_Physical_State(int action)
}
  if (action == 0){
    final_Angle = Initial_Angle + Delta_Servo;
    wristServo.write(final_Angle);
    Initial_Angle = final_Angle;
    delay(delayTime1);

{
 else if (action == 1} (
    final_Angle = Initial_Angle - Delta_Servo;
    wristServo.write(final_Angle);
    Initial_Angle = final_Angle;
    delay(delayTime1);
  {
   else if (action == 2){
    final_Speed = Initial_Speed + Delta_Speed;
    analogWrite(PWM, final_Speed);
    Initial_Speed = final_Speed;
    delay(delayTime2);
   }
   else {
    final_Speed = Initial_Speed - Delta_Speed;
    analogWrite(PWM, final_Speed;(
    Initial_Speed = final_Speed;
    delay(delayTime2);
   }{
float get_Look_Ahead ()
{
  float valMax = -100000.0;
  float val;
  if ((s1 + 1) != Position_States){
    val = Q[sPrime][0];
    if (val > valMax){
      valMax = val;
    }
}
  if (s1 != 0){
    val = Q[sPrime][1];
    if (val > valMax){
      valMax = val;
```

```
{
{
 if ((s2 + 1) != Speed_States){
  val = Q[sPrime][2];
  if (val > valMax){
   valMax = val;
 }}
 if (s2 != 0)|{
  val = Q[sPrime][3];
  if (val > valMax){
   valMax = val;
  }}
 return valMax;
}
const int rollDelay = 200;                  //allow time for the agent to roll after it sets its physical state
const float explorationMinutes = 1;         //the desired exploration time in minutes
const float explorationConst = (explorationMinutes * 60.0) / ((float(rollDelay)) / 1000.0); //this is
the approximate exploration time in units of number of times through the loop
float look_Ahead_Value = 0.0;
float sample = 0.0;
int a = 0;
int i = 0;
int t = 0;


void loop()
}
 while (Serial.available()>0(
}
 int error=Serial.read;()
 t;++
  epsilon = exp(-float(t) / explorationConst;(
  int a = Get_Action;()
  Set_sPrime(a;(
  set_Physical_State(a;(
  delay(rollDelay;(
  int Reward = Get_Reward;()
  look_Ahead_Value = get_Look_Ahead;()
  sample = Reward + Gamma * look_Ahead_Value;
  Q[s][a] = Q[s][a] + alpha * (sample - Q[s][a;([
//  Q[s][a]= Q[s][a] + alpha *[(Reward + Gamma * Q_max[s_prime][a_prime])- Q[s][a;[ [
  s = s_prime;
  if (t == 2) {              //need to reset Q at the beginning since a spurious value arises at the first
initialization (something from the rangefinder(..
    initializeQ;()
    printQ;()
{
// Serial.println(a);
 i;++
 if (i == 9} (
   printQ;()
   Serial.print(s);
```

Appendix B: Car test application code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO.Ports;
using System.IO;

namespace application_of_car
}
   public partial class CAR : Form
}
      public CAR()
}
         InitializeComponent;()
{

      private void Right_Click(object sender, EventArgs e(
}
         serialPort1.Write("R;("
{

      private void textBox1_TextChanged(object sender, EventArgs e(
}


{
       private void Form1_Load(object sender, EventArgs e(
    }
           String[] ports = SerialPort.GetPortNames;()
           COMPORT.Items.AddRange(ports;(
    {
       private void button2_Click(object sender, EventArgs e(
    }
           try
    }
              serialPort1.PortName = COMPORT.Text;



              serialPort1.Open;()
              progressBar1.Value = 100;
              button2.Enabled = false;
              button5.Enabled = true;
              COMPORT.Text = "ON;"
    {
           catch (Exception err(
    }
              MessageBox.Show(err.Message,          "Error",          MessageBoxButtons.OK,
    MessageBoxIcon.Error;(
              button2.Enabled = true;
              button5.Enabled = false;
              COMPORT.Text = "OFF;"
    {
    {
        private void forword_Click(object sender, EventArgs e(
    }
          serialPort1.Write("F;("
    {

        private void button1_Click(object sender, EventArgs e(
    }
          serialPort1.Write("L;("
    {

        private void button3_Click(object sender, EventArgs e(
    }
          serialPort1.Write("S;("
    {

        private void button5_Click(object sender, EventArgs e(
    }
          if (serialPort1.IsOpen(
    }
              serialPort1.Close;()
              progressBar1.Value = 0;
              button2.Enabled = true;
              button5.Enabled = false;
              COMPORT.Text = "OFF;"
    {
    {

        private void button4_Click(object sender, EventArgs e(
    {     }
```

```
{
}
```

Appendix C: Camera code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO.Ports;
using System.IO;

namespace application_of_car
}
  public partial class CAR : Form
}
     public CAR()
}
        InitializeComponent;()
{

     private void Right_Click(object sender, EventArgs e(
}
        serialPort1.Write("R;("
{

     private void textBox1_TextChanged(object sender, EventArgs e(
}
     {     private void Form1_Load(object sender, EventArgs e(
}
```

```
        String[] ports = SerialPort.GetPortNames;()
        COMPORT.Items.AddRange(ports;(

{
      private void button2_Click(object sender, EventArgs e(
}
        try
}
            serialPort1.PortName = COMPORT.Text;
            serialPort1.Open;()
            progressBar1.Value = 100;
            button2.Enabled = false;
            button5.Enabled = true;
            COMPORT.Text = "ON;"

    {       catch (Exception err(
}
            MessageBox.Show(err.Message,          "Error",          MessageBoxButtons.OK,
MessageBoxIcon.Error;(
            button2.Enabled = true;
            button5.Enabled = false;
            COMPORT.Text = "OFF;"
{
{
      private void forword_Click(object sender, EventArgs e(
}
        serialPort1.Write("F;("
{

      private void button1_Click(object sender, EventArgs e(
}
      serialPort1.Write("L;("
{

    private void button3_Click(object sender, EventArgs e(
}
      serialPort1.Write("S;("
{

    private void button5_Click(object sender, EventArgs e(
}
      if (serialPort1.IsOpen(
}
        serialPort1.Close;()
        progressBar1.Value = 0;
        button2.Enabled = true;
        button5.Enabled = false;
        COMPORT.Text = "OFF;"
{
{

    private void button4_Click(object sender, EventArgs e(
}
   {
{
{
```

Appendix D: Dynamic Model

The dynamic model is another approach that used to implement the equation of motion in chapter 4 which describes the car's behavior and take into account the torque and forces affected on the system.

Fig A.1 demonstrates the free body diagram of the bicycle model approximation that has been assumed in this work. So, in the body-coordinates there is many parameters that can be defined as so,

β : represents the yaw angle or the heading of the whole bicycle that rotates a bout z -axis at the center of gravity point.

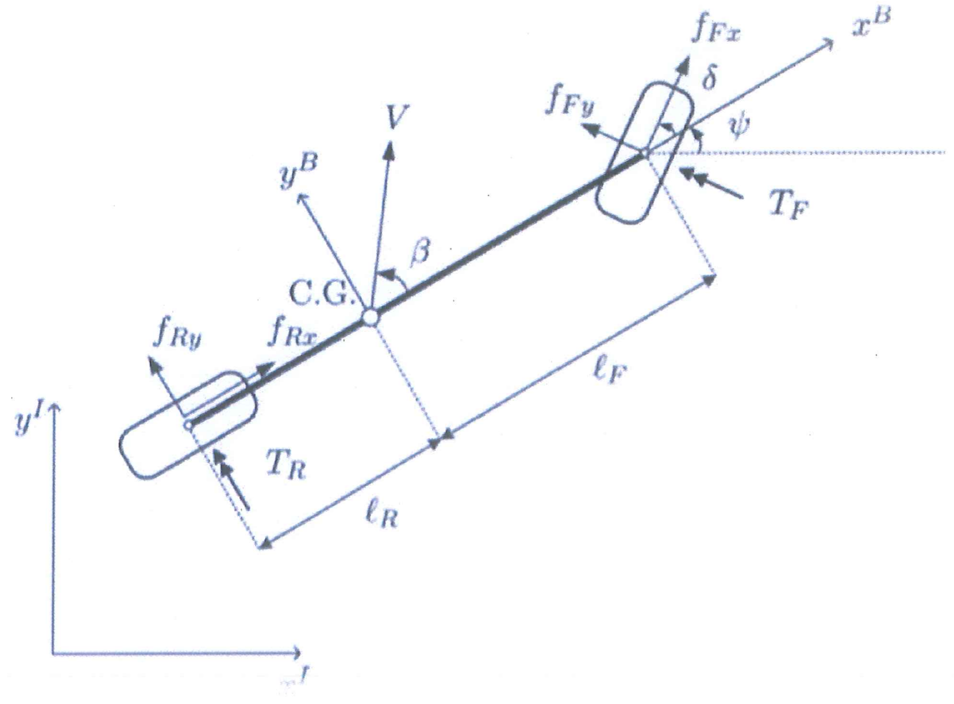


Fig A.1: Dynamic bicycle model approximation of the vehicle

- δ: Refers to steering angle of the front wheel, that makes the bicycle rotate about z-axis, drift from x^B where the vehicle's forward direction.
- T_F: referred to the applied torque on the front wheel.
- T_R: referred to the applied torque on the rear wheel.
- l_f : the length between the center of front wheel and center of gravity
- l_f : the length between the center of rear wheel and center of gravity
- $fij\ (i = F, R\ and\ j = x, y)$: denotes to the longitudinal and lateral friction forces at the front and rear wheels, respectively [8].

$$\beta = \operatorname{atan}(\frac{V_y}{V_x}) \tag{A.2}$$

$$V_x = V\cos\beta \tag{A.3}$$

$$V_y = V\sin\beta \tag{A.4}$$

Based on Newton second law for linear motion at Eqn A.5 for x-axis, y-axis and the and for angular motion about z-axis at Eqn A.6, by apply them into this system to get equations A.7, A.8 and A.9

$$\sum F = ma \tag{A.5}$$

$$\sum M = \mathrm{I}\ddot{\psi} \tag{A.6}$$

$$f_{\mathrm{Fx}}\cos\delta - f_{\mathrm{Fy}}\sin\delta + f_{\mathrm{Rx}} = m(\dot{\mathrm{V}}_{\mathrm{x}} - \mathrm{V}_{\mathrm{y}}\dot{\Psi}) \tag{A.7}$$

$$f_{Fx}\sin\delta + f_{Fy}\cos\delta + f_{Ry} = m(\dot{\mathrm{V}}_y + V_x\dot{\Psi}) \tag{A.8}$$

$$\left(f_{Fy}\cos\delta + f_{Fx}\,sin\delta\right) l_F - f_{Ry}\,l_R = I_z\ddot{\psi} \tag{A.9}$$

References

[1] K. N. V Satyanarayana, B. Tapasvi, P. Kanakaraju, and G. Rameshbabu, “Based on machine learningAutonomous car using raspberry-pi .,” vol. 7, no. 12, pp. 76–82, 2017.

[2] S. S. Madhu Govindarajan, "www.hackster.io," 6 August 2018. [Online]. Available: https://www.hackster.io/user885477/reinforcement-learning-for-a-self-balancing-motorcycle-3fdbaf. [Accessed 12 December 2018].

[3] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, “Control of a Quadrotor with Reinforcement Learning,” no. 644227, pp. 1–8, 2017.

[4] P. Kormushev, S. Calinon, and D. G. Caldwell, “Robot motor skill coordination with EM-based reinforcement learning,” in *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings*, 2010, pp. 3232–3237.

[5] adminYosh, "comocalcular," 1 3 2019. [Online]. Available: https://comocalcular.com.br/fisica/forca-centripeta-conceito-e-formula/?fbclid=IwAR26CNkRYzlPfzDtwOtyWqjqmDTtiy9HrU_b3hnpwhxpk79WmEg5aPFogVQ.

[6] P. R. Montague, “Reinforcement Learning: An Introduction, by Sutton, R.S. and Barto, A.G.,” *Trends in Cognitive Sciences*, vol. 3, no. 9. p. 360, 1999.

[7] aliexpress, "tr.aliexpress.com," [Online]. Available: https://tr.aliexpress.com/item/A959-A-2-4G-1-18-Scale-4WD-Electric-RTR-Off-road-Buggy-RC-Car-Toys/32841462141.html?spm=a2g10.10010108.1000016.1.48d5185fX3qUSI&isOrigTitle=true.

[8] oncetop, "oncetop," 18 12 2018. [Online]. Available: http://www.oncetop.com/Products/OT-RK-540PHBrushMoto.html.

[9] NovalithIC, "robotpower," 17 12 2018. [Online]. Available: http://www.robotpower.com/downloads/BTS7960_v1.1_2004-12-07.pdf.

[10] electronics, "core-electronics," 17 12 2018. [Online]. Available: https://core-electronics.com.au/nano-v3-0-board.html.

[11] xbee, "www.sparkfun.com," [Online]. Available: https://www.sparkfun.com/pages/xbee_guide.

[12] NovalithIC, "robotpower," 17 12 2018. [Online]. Available: http://www.robotpower.com/downloads/BTS7960_v1.1_2004-12-07.pdf.

[13] amazon, "amazon," 17 12 2018. [Online]. Available: https://www.amazon.com/Optex-FA-background-suppression-photoelectric/dp/B00UGB9GO4.

[14] testsieger, "testsieger," 17 12 2018. [Online]. Available: https://www.testsieger.de/testberichte/logitech-hd-webcam-c270.html.

[15] luckystore, "luckystore," 17 12 2018. [Online]. Available: http://luckystore.com.sg/cameras/web-camera/logitech-c920-hd-pro-webcam.html.