Heriot-Watt University

Deliverable 1: Final Year Dissertation

A Self-Driving Donkey Car

Author:
Tian-Tong Yin

Supervisor:
Dr. Hani Ragab

A thesis submitted in fulfilment of the requirements for the degree of BSc Hons

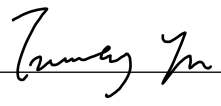
in the

School of Mathematical and Computer Sciences

November 2019

Declaration of Authorship

I, Tian-Tong Yin, declare that this thesis titled, 'A Self-Driving Donkey Car' and the work presented in it is my own. I confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed: 

Date: 22 November 2019

"The journey of a thousand miles starts with a single step."

Chinese Proverb

Abstract

With the advancements in technologies, machine learning has seen an increase in its use in many fields. One of the fields is self-driving system for cars. This thesis proposes a self-driving car based on Raspberry Pi. It proposes a car which has basic self-driving functions including lane following, impact avoiding, and sign recognizing with real-time results on a low computing-power platform. The primary technology it will be using is deep learning and computer vision. The software is running on a Raspberry Pi with custom driver board and components (Xiao-R) made for Donkey Car. The software is written in Python, based on the Donkey Car library and expanded to do more sophisticated tasks.

Key Words: Self-Driving, Computer Vision, Deep Learning

Contents

Chapter 1

Introduction

With the rapid development of machine learning and deep learning, self-driving cars—which has been talked about since the early 20th century—are finally here. There are many arguments about which one is the first self-driving car, one of the arguments being the Stanford Cart. Built in the 1960s, it is one of the earliest self-navigating vehicle using images as input [1, 2]. Technology has come a long way since. The slow-moving robot which takes 10-15 minutes to move every meter has now developed to cars equipped with multiple advanced sensors and are capable of driving in real-life scenarios.

1.1 Aim

The aim of this project is to build a self-driving RC car. The RC car should ideally run with cameras on a Raspberry Pi. The car should be able to perform critical self-driving goals such as lane detection, obstacle detection, and sign recognition, under a simplified driving environment. The RC car should be able to finish these tasks with cameras using computer vision. The car's image processing should be in real-time so it can provide reliable driving.

1.2 Objectives

The main objectives of this project should be the following:

- Investigate in self-driving car technologies
- Implement a program which can do self-driving tasks reliably using RGB images as input
- Optimize said program to run on low-computing-power platforms with decent performance
- Offload some work to be performed on a host computer if needed

1.3 Manuscript Organization

This report has four chapters: Introduction, Literature Review, Project Planning, and Initial Testing. In chapter 2, we discuss the technologies used by both self-driving RC cars and real self-driving cars. We also compare the methods and discuss about the advantages and disadvantages of these methods. In chapter 3, we discuss the requirements of our project, the initial design, and the evaluation strategy. We then talk about our project planning strategies including timeline and risks. In chapter 4, we discuss our initial experience with the provided Donkey Car, alongside with the limitations of it.

Chapter 2

Literature Review

In this chapter, we discuss some of the technologies used in self-driving cars, alongside with whether they are suitable for a self-driving RC car. This chapter will be split into three sections, which are the three main tasks involved in self-driving: environmental perception, decision making, and executing.

The upsurge of self-driving cars has been going on for a few years. It started with Tesla, Google, and now more and more traditional automobile manufacturers such as Audi, Ford, and BMW are joining in. With the help of LiDAR, radar, cameras and other sensors, autonomous vehicles now days are able to achieve L3, or even L4 driving automation [3].

Self-driving cars can be viewed as 3 main components integrated together: environmental perception, decision making and planning, and execution. The following paragraphs explores each of the components in depth.

2.1 Environmental Perception

The first task a self-driving car needs to accomplish is seeing and understanding the environment it is in. The two main parts in this section address them separately: the self-driving car will be first sensing the environment, and then understanding it through various algorithms.

2.1.1 Sensors

A critical task for self-driving cars is that they need to know their exact location and position. This does not just require a traditional navigating system, a self-driving car

also requires a very precise positioning wherever it goes[4]. Not only it needs to know which road it is on, it also needs to know which lane and its exact position on the road, on the centimetre scale. One way to achieving this is to use a high-accuracy GPS alongside with RTK (Real-time kinematic) and IMU (Internal measurement unit) devices to get a localization with pinpoint accuracy on a provided high definition map.

This method would also require the car to have the capability of doing SLAM.

SLAM (simultaneous localization and mapping), also known as CML (Concurrent Mapping and Localization), is a method for robot navigation system [5]. It allows a robot to start from a unknown position in a unknown environment, while gradually building a map for the environment, updating its position in the environment using the current map [6].

One commonly used method of implementing SLAM is using a LiDAR (Light Detection And Ranging). A LiDAR is a laser-based radar. It uses a beam of laser and its reflection to measurement the distance between it and the object it is pointing at. A LiDAR which a car/robot uses often contains one laser projector which rotates around or back and forth at a very fast speed, so that it can scan the environment within a certain angle around it with limited numbers of beams of laser [7].

In combination with SLAM results from a LiDAR and previously mentioned high definition map, the self-driving car can combine the SLAM result with the details and landmarks on a high definition map provided, and understand exactly which position it is at [8]. However, those devices are expensive and are often not robust enough for a self-driving system, and a high definition map is often not available for most locations since a map for this purpose requires very detailed information.

Another approach is visual SLAM or pose-graph SLAM. It uses SLAM from either a LiDAR (as mentioned in[4]), an RGB-D camera[9], or any other device with depth information such as ultrasonic sensors, to create a detailed map (could be 3D) of the environment, thus knowing its relative position in the environment. Instead of a provided high definition map, it will generate a "high definition map" as it goes. However, devices such as LiDAR and ultrasonic sensor which only returns depth information would require a camera to provide additional information to reach best performance.

Self-Driving RC Cars

The scenario is a bit different for self-driving RC cars. While real life sized cars can carry 4-8 cameras and LiDARs alongside with many other sensors, an RC car is usually relatively small thus its choice of sensors is limited. In the scope of this project, as a

small device which will only be running in a small environment, the RC car does not need to worry about the global map and can focus on the local map since the surrounding environment is all that matters.

There are different designs of self-driving RC cars. Some use infrared sensors in front of the car to follow a black line on the ground [10]; some use ultrasonic sensors to move around without hitting any object [11]; some use a small RiDAR to implement SLAM [12].

One of the very popular designs is to use cameras since a camera can provide richer information than other sensors. Some projects combined camera with other depth sensors[13], some used multiple cameras together as a stereo camera to get depth information out of cameras[14], but most of them use a monocular camera to do the job. Although it does not have the best performance, it is the easiest and cheapest to deploy. With a good model of computer vision, a monocular camera can also provide stable and reliable results under a relatively simple environment.

2.1.2 Computer Vision

After gathering information from the sensors, the next step for environmental perception is for the RC car to understand the information. There are many ways of doing such a task. Information returned by a LiDAR, ultrasonic sensor, or depth information from an RGB-D camera can be used most of the time by the RC car directly since they will be returning a numeric value which requires no further processing. However, if images are used as a source of information, some computer vision algorithms are required.

There are multiple ways to extract useable information from images. Traditional detection algorithms consist of two steps: feature extraction, and feature selection[15]. Basic feature extraction can be done mathematically, without a machine learning algorism. Features from an image can be extracted by calculating the gradient [15] or comparing the redundancy of pixels. Extracted features can be seen as an abstraction of the original image, focusing on the potential points-of-interest. By further applying feature selection, a small number of features will be selected out of all features and stored for the computer to use[15]. This further reduces noise factors and other interferences, improving robustness of the system. The selected features are also called feature vectors.

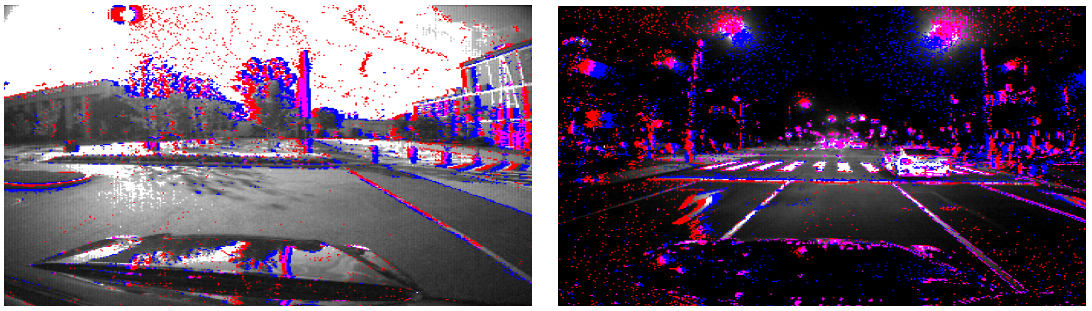
One of the widely implemented method is histogram of oriented gradients (HOG)[15, 16, 17]. HOG divides original image to a grid of small blocks, each block overlaps with the blocks next to it. Each block is normalized independently, but because of the overlap, each block after normalization still have some connection to the surrounding blocks. The

feature vector is formed by the histogram of the gradients of each pixel in each block. The gradients will be calculated for both x and y axes, representing the edges in its direction.

However, HOG features are hand-crafted, which means that it requires an algorism which is manually predefined by domain-specific experts [15, 18]. That is why newer approaches of feature extraction developed into using learned features. Learned features can be generated from a dataset using a training procedure, thus avoiding the need for a manually specified algorism [18]. This is usually achieved by the usage of neural networks. CNN (Convolutional Neural Networks), for example, can be used to generate learned features. being able to generate features on its own is not its only advantage. According to [18], CNN trained features outperform hand-crafted features significantly on heterogeneous data and generalize well on unseen datasets. Also, even small CNN such as Mini-CNN trained on minimal data can have features which perform almost as well as bigger networks such as AlexNet-CNN. This makes this approach also suitable for our project since our RC car will not be capable of running huge neural networks and we will not have access to large training data.

There is a common misconception about T-Rexes, that they can only see objects that are moving. Although this is most likely not the case, there is a type of camera which behaves in this way. A neuromorphic camera, also known as an event camera or DVS (Dynamic Vision Sensor), is a vision sensor which captures event-based motion. Compared to traditional cameras, it has much higher dynamic range, high temporal resolution, and not affected by motion blur [19]. Unlike traditional cameras which capture frames at a fixed rate, each pixel of a DVS reacts to the sudden change of brightness [20]. This means that if nothing is changing in the vision of a DVS, it will not capture anything. This feature makes DVS suitable for 3D perception for navigation. From Figure 3 in [21], we can see the special characteristics of DVS. Since a DVS only captures changes of brightness, a large plain which has little variation in texture will be less likely to be captured. In the application of cars localization, this means that there will be less noise coming from the road and the sky, while objects such as signs and other cars will attract high attention since the relative position between them and the camera will be constantly changing. This makes feature extraction much easier to perform since it is already partially done by the DVS.

A combination of LiDAR, event camera, stereo camera, and IMU are used in [21]. With these technologies joined together, the vehicle is able to reconstruct an accurate 3D map of the environment it was driven in.



Daytime Driving Nighttime Driving

FIGURE 2.1: Events (red and blue) captured by a event camera, overlaid on sample image [21].

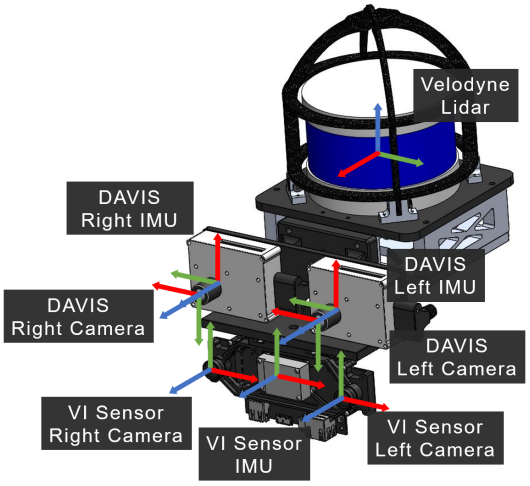


FIGURE 2.2: Model of a sensor rig combining multiple sensors [21].

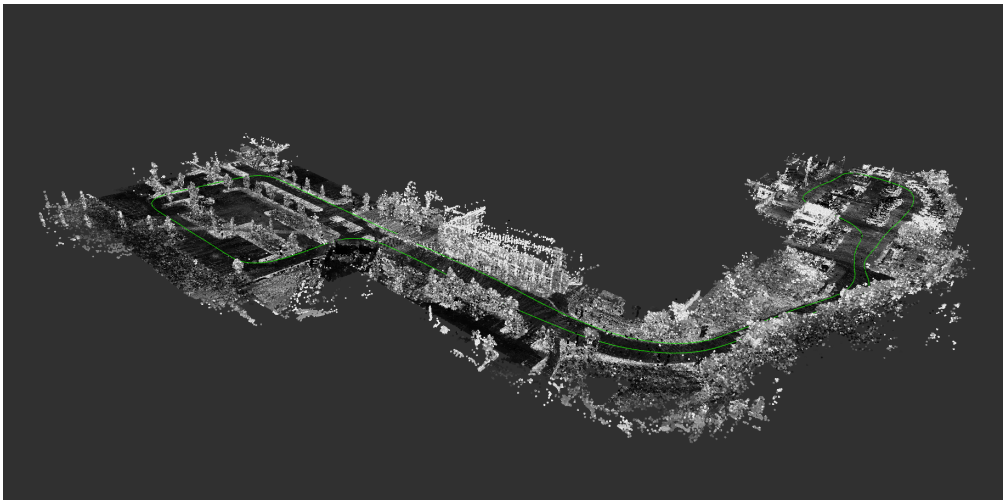


FIGURE 2.3: Full map generated from the driving sequence in 2.1, Daytime Driving, trajectory in green[21].

Convolutional Neural Networks

Convolutional Neural Networks is a popular technique for feature extraction in computer vision [15, 17, 18]. The first "convolutional neural network" is the Neocognitron by Fukushima in 1980 and the standard reference for CNN is from 1998 by LeCun et al., "Object Recognition with Gradient Based Learning". Convolutional neural network is a type of feedforward neural network which take advantage of convolution calculation.

Before passing to the CNN, images usually need some prior processing, for example, zero-centring or normalizing. This can give the neural network a better performance. A CNN has three types of special layers: convolutional layers, activating layers, and pooling layers[22, chap. 9, pp. 326–329][23, chap. 6][24]:

The convolutional layer is the core layer of CNN, as its name implies. Each convolutional layer has a kernel of a certain size in pixels, which will slide through the complete image input. The sliding is controlled by stride, which determines the distance between the kernel movement. The neurons—in this case, the pixels—which are included in the current kernel will be connected to one neuron in the next layer. So each neuron in a layer can be viewed as a segment of the previous layer (Figure 2.4). The size of the output can stay the same as the input if desired. This is called same convolution. It can be achieved by implementing different padding schemes, such as adding a circle of numbers around the input.[25]

The most important part of a convolutional layer is the kernel, which can be trained. Different kernels can extract different features, such as specific shape, colour, contrast level etc. The extract features are stored in feature maps. No matter what the depths of the input layer is, the depths for a feature map will always be 1. If the input has a depth of M, then the filter will also have a depth of M, and the result of each layer will be combined. (see Figure 2.5) We can apply the convolutional layer to the extracted feature maps again, to further extract features out of features. The first convolutional layer can extract low-level features, and convolutional layers after can extract higher-level features. After some layers have been applied, the high-level features extracted can be used for classification in a structure same as the traditional neural network.

When applying a convolutional layer, each kernel will be able to extract a certain type of feature. A convolutional layer will usually have multiple kernels to extract different features. For an oversimplified example, in a face recognition network, one kernel can be used to extract eyes and another kernel can be used to extract mouth.

The activating layer is the same as applying the activation function in traditional neural networks. It makes the output of the network non-linear. It can be viewed as non-linear

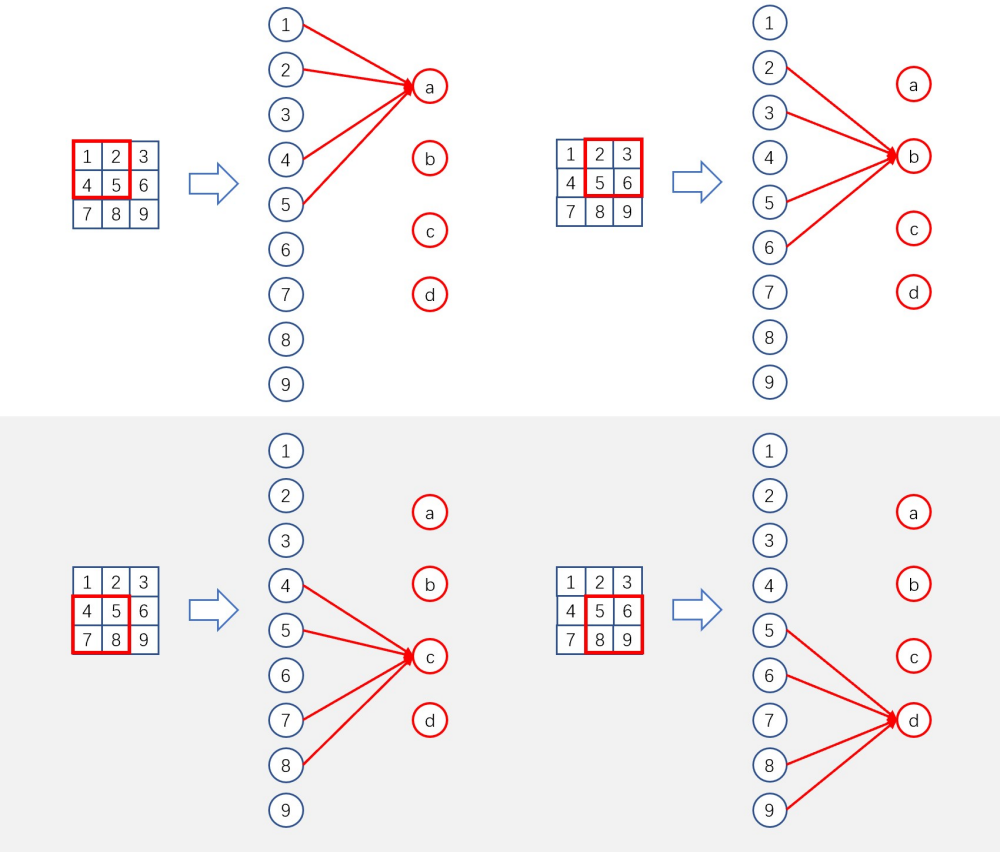


FIGURE 2.4: Convoluting a 3×3 image with a 2×2 kernel.

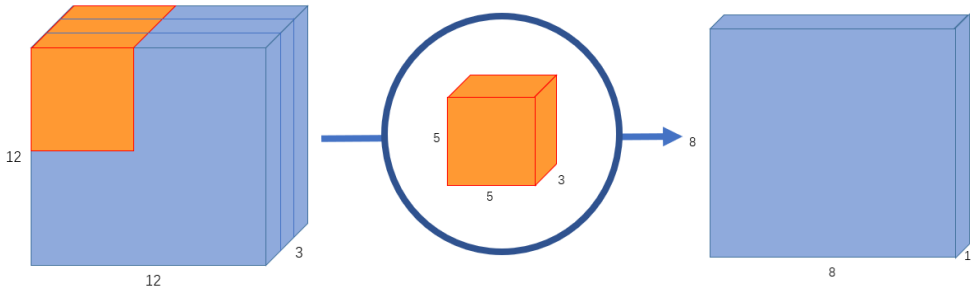


FIGURE 2.5: A traditional convolution of a $12 \times 12 \times 3$ input [26].

mapping. The most commonly used activation function in CNN is ReLU (rectified linear unit). It effectively removes the negative values by setting them to 0. If ReLU can not produce a satisfying result, its alternative such as Leaky ReLU and PReLU can be used. Other activation functions such as tanh can also be used.

Pooling layers are used to reduce the number of features, it is a downsampling layer. It separates the features into smaller, non-overlapping sub-groups, and for each sub-group returns a single feature representing the whole sub-group. There are many different types of pooling such as max pooling which returns the maximum value from the current sub-group, and average pooling which returns the average value from the current sub-group.

After enough layers of convolution and pooling, enough features have been extracted and the network will move to the classification stage. The output from previous layers will be flattened into a one-dimensional vector. Then said vector is passed to fully connected layers where each neuron is connected to all activations in the previous layer, as the same way in traditional neural networks. Classification will be done in these layers and sent to the output layer.

Although CNN has been introduced in the 1990s, its common usage started not too long ago, after the introduction of AlexNet [27]. CNN can pick up features and further extract features out of it. At the end of the network, instead of classifying an image, we are classifying a set of features. This makes it very suitable for pattern recognition as it is more generalized than normal neural networks, but it is still far from "seeing" the picture.

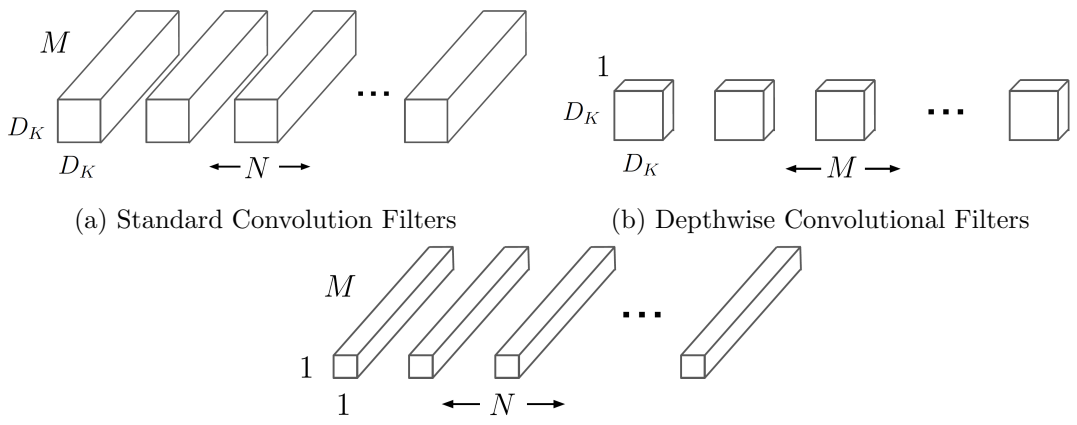
2.1.3 Computer Vision on Low Computing Power Devices

A real-life self-driving car would also usually carry a much more powerful computing device than an RC car. For example, Tesla have their computing components customized from Nvidia at first, and now manufacturing it themselves. It has two GPUs and requires liquid cooling systems. They are much more powerful than any RC car.

Most self-driving RC cars run on a Raspberry Pi, Arduino, or STM32. The Donkey Car, for example, runs on a Raspberry Pi. This limits the number of algorithms it can choose from. Our goal here is to find a suitable algorism which not only need to provide high accuracy for self-driving, but also need to have high efficiency so that it can run on the Raspberry Pi in real-time.

Antipov et al. suggested that good performance can be achieved on small CNN and relatively small datasets [18]. So small convolutional neural networks can be taken into consideration. Some lightweight CNN which provide low latency and less computation expensive have been developed. One of the popular choices is MobileNet.

MobileNet [28, 26] is a model of CNN which has a balance between latency and accuracy. The main difference between MobileNet and traditional CNN is the usage of depthwise separable convolutions. As mentioned above, the depth of the feature map will always be 1. This means that if an output of depth N, then N feature maps are required. In a standard convolutional layer, the convolution combines both filtering and combining into one step. Which means in the example of Figure 2.6, N filters of size $D_K \times D_K \times M$ needs to be applied to get an output of depth N.



(a) Standard Convolution Filters

(b) Depthwise Convolutional Filters

(c) Pointwise Convolutional Filters

FIGURE 2.6: The standard convolutional filters in (a) are separated into depthwise convolutional filters in (b) and pointwise convolutional filters in (c) [28].

Depthwise separable convolution improves the speed by separating the filtering and the combining into two steps. Unlike standard convolution which will directly combine the results after filtering, depthwise separable convolution uses depthwise convolutional filters. With M $D_K \times D_K \times 1$ filters applied as shown in Figure 2.6(b), we get a output of depth M. Then the output can easily be combined using a $1 \times 1 \times M$ filter. If the output needs to be in depth N, it can be achieved by applying the $1 \times 1 \times M$ filter N times as shown in Figure 2.6(c).

Assume a convolutional layer takes an input of size $D_F \times D_F \times M$ and produces an output of size $D_F \times D_F \times N$.

The computational cost for a standard convolution is [28]:

$$D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F \tag{2.1}$$

And the computation cost for a depthwise separable convolution is [28]:

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F \tag{2.2}$$

which is the sum of the computation cost of depthwise and pointwise convolutions.

Comparing 2.1 and 2.2, it is apparent that the computation cost for depthwise separable convolution is significantly lower. Figure 2.7, 2.8 shows a more intuitive visual representation of depthwise separable convolution.

In 2018, MobileNetV2, an improvement based on MobileNet, was introduced [29]. The paper suggested that the set of layer activations for any layer forms a "manifold of interest" [29], which could be embedded in a lower dimensional subspace. This means

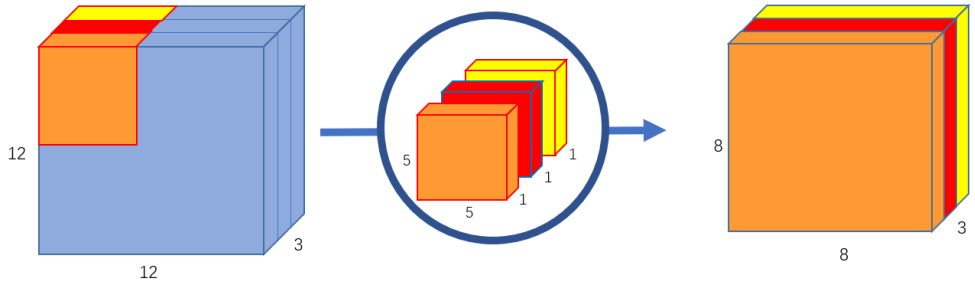


FIGURE 2.7: Depthwise convolution, uses a filter for each layer in the input depth [26].

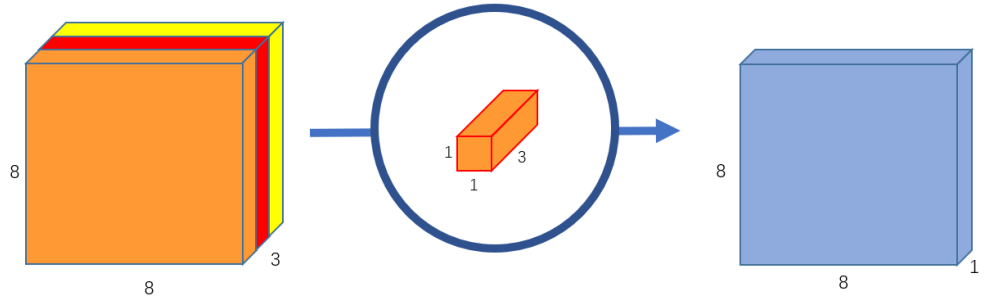


FIGURE 2.8: Pointwise convolution, uses a $1 \times 1 \times 3$ filter to combine all layers in the input depth [26].

that the activation space has information which is not "interesting" to the network. In MobileNetV1 [28], this was solved by simply reducing the dimensions of the activation space until it is filled by the manifold of interest. However, the existence of the activation function (ReLU) will cause information loss to the layer it is applied to. This is improved by the usage of linear bottlenecks and inverted residuals.

Many techniques were used to improve the performance of both speed and accuracy in CNN, such as using different kernel size in a single layer (Inception [30]), connecting one layer to another layer which is not directly after the first layer (skip connection [31]), and shuffling the channels then only convoluting part of them (channel shuffle [32]). However, these methods still require decent computing power and sometimes cannot be achieved due to the design of the project. To solve this issue, some projects used Jetson Nano [33], a mini computer built for deep learning, as the controller of the self-driving car [34]. Other options include having an additional computing unit such as Google Edge TPU [35] or Intel Neural Compute Stick [36] to accelerate the computation [37].

Another common workaround is offloading some computation to a much powerful host computer [38]. Adding an extra unit can increase the budget as an external computing unit often cost a lot. Using a host computer allows us to achieve similar results without

increasing the cost, assuming a computer is already available. Also, a host computer has a much higher performance limit. A USB-connected computing unit has limitations on its power supply, size, cooling, etc., thus impacting the maximum performance it can achieve. A host computer, on the other hand, has no such limitations. Its performance completely depends on its own, as it runs independently of the RC car.

When using a host computer, the utmost important part and potential bottleneck is the communication between the RC car (described below as controller) and the host computer (described below as computer). There are several datasets need to be sent, the controller needs to send the computer all information needed for processing. This could include images, depth images, distance information, depending on the design of the RC car. For a self-driving car, the image processing needs to be done in real-time, which means we need to achieve minimum latency during data transmission between the controller and the computer. The requirement for response time varies in different driving conditions. In real life scenario, the response time affects the stopping time heavily, especially at high speeds. This is a crucial safety hazard, which is why minimizing the response time is one of the most important tasks in self-driving cars. Elon Musk claims that Tesla's Autopilot 2.0 hardware can process 200 frames per second and the new "hardware 3" will be able to process 2000 frames per second with redundancy [39].

In this project, the RC car we are building will only be running at a relatively low speed, we estimated that 10 action updates per second would produce decent enough performance. So the aim at this time is to get a response time less than 100ms, which might change throughout the project.

There are many ways to connect a Raspberry Pi to a computer. The method with the least latency is connecting via a cable. The easiest way to do so is using an Ethernet cable. A crossover Ethernet cable can be used to connect the two devices, allowing them to communicate with each other. However, this would require both devices to have an Ethernet port.

Another method is with a serial communication circuit. For this, we need a USB to serial adapter, then the Raspberry Pi can be connected to the adapter with its GPIO pins.

Although connecting via cable has low latency and minimum package lost, it requires a physical connection to the computer at all time, which can cause some trouble. A more convenient approach is to use a wireless connection, such as Bluetooth or LAN (local area network). Since the Raspberry Pi 3B+ we are using has both Bluetooth and WiFi function, all hardware requirements are met. These methods are more mobile, but usually have higher latency and potential packet loss.

To minimize latency we need to minimize the size of data which are sent. Bagwe et al. [40] suggested that fixed-rate cameras will capture some amount of redundant data, and can be improved by implementing adaptive frame rate control. The frame rate will be increased when faster response time is required and will be reduced in low informative circumstances. A similar method is purposed by Kang et al. in [41]. He suggested that the frame rate or resolution can be reduced in a scenario where the human driver should not operate the vehicle, such as waiting at a red light. The logic is similar to the event camera introduced above, while an event camera avoids capturing unnecessary features from a frame, video frame reduction avoids capturing unnecessary frames, allowing us to have faster processing speed and shorter transmission time.

2.2 Decision Making and Planning

In section 2.1.2, we introduced some methods the car can use to understand the environment. In this section, we assume that all needed information is successfully extracted from the sensors. In real life scenario, there will be multiple types of lane markings, e.g. solid white, single solid yellow, and double solid yellow. The car needs to separate them as its decision might vary depending on which lane marking it is. In this project, our RC car will be running in a simplified environment, at the moment we will only take one type of marking into consideration.

After the self-driving car has understood the environment, it needs to decide how to react. The most basic and important task a self-driving car needs to do is lane-keeping. After detecting the lanes, the car needs to understand what is the meaning of the lane and how to respond to it. It can be achieved by estimating the lateral offset and suitable heading angle, which was done in [42].

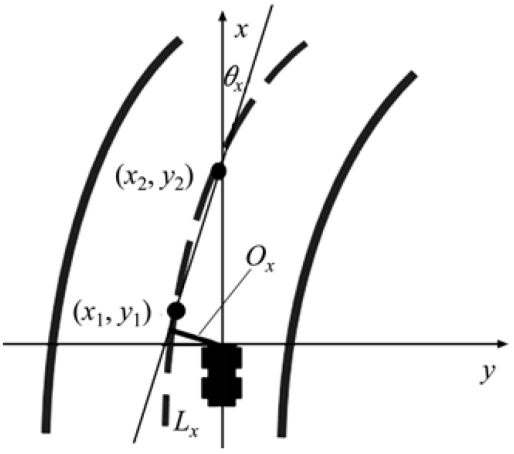


FIGURE 2.9: World coordinate system [42].

A world coordinate system is represented in figure 2.9, where the car needs to estimate the lateral offset O_x and the heading angle θ_x. One of the detected lane lines is denoted as L_x. Two points in the world coordinate system (x_1, y_1) and (x_2, y_2) where both points are on L_x are computed using camera calibration. This will be used to estimate L_x as a straight line. Then the heading angle can be estimated as [42]:

$$\theta_x = \arctan((y_2 - y_1)/(x_2 - x_1)) \tag{2.3}$$

and the lateral offset can be estimated as [42]:

$$O_x = \begin{cases} D_x, & \text{if} \quad L_x \text{ is segmented} \\ D_x - W_{lane}, & \text{if} \quad L_x \text{ is solid and } C_x < 0 \\ D_x + W_{lane}, & \text{if} \quad L_x \text{ is solid and } C_x \geqslant 0 \end{cases} \tag{2.4}$$

where

$$C_x = y_1 - \frac{y_2 - y_1}{x_2 - x_1} x_1 \tag{2.5}$$

$$D_x = -\cos(\theta_x) \times C_x \tag{2.6}$$

where W_{lane} is the width of the lane, which is assumed to be fixed. A solid lane line is considered as the left boundary if $C_x < 0$, and is considered as the right boundary if $C_x \geqslant 0$; all segmented lane lines are considered as middle lines.

With this method, we are able to calculate the lateral position of the car and which angle it should be heading. With these two values, along with some information about the car including steering angle and current speed, the car can be easily programmed to stay inside the lane.

There is an easy workaround to it: The car should be able to see both the left lane marking and the right lane marking at the same time when running correctly. It is easy to program the car to steer left if it only sees the right lane marking, and steer right if it only sees the left lane marking, as there will always be a time period that the car can only see one line before going out of the lane. This is the same method used for infrared sensors used in [10]. As shown in project [43], the trajectory of the car using this method is full of sharp turns instead of a smooth curve, thus not suggested for a self-driving car.

Another approach to this is using machine learning to teach the car how it should behave. This is used in many projects including the Donkey Car [44]. With this method, the car will take human driver action along with the lanes detected as input, and try to create a relation between the two of them.

Mathematical modelling and machine learning both have their advantages and disadvantages [45]. While providing robustness, the biggest advantage of using machine learning is that we do not have to build the mathematical model ourselves. The evaluation can be automated using machine learning, while mathematical modelling requires expert knowledge and a full understanding of the model. Sometimes, machine learning can also pick up small details which we might not have recognized. However, machine learning requires lots of training data, ideally covering all possible cases. It can behave poorly with completely new cases, while such case might be captured in a mathematical model.

The other part of a self-driving car other than lane following is to avoid obstacles and react properly to signals and street signs. The lane detection algorism and object detection algorism are often implemented in different models, allowing them to work independently with more flexibility [37, 38]. Reacting to object recognition would not require any advanced algorithm, we can simply hard-code it to do required tasks such as stopping at a red light or when detecting an obstacle.

2.3 Execution

Real-life self-driving algorithms often take control of three parts: steering, brake, and throttle. This means that it will always be monitoring the speed of the car and needs to know the exact capability of the car so that it can calculate how much throttle or brake it needs to achieve target speed. In our project, this is simplified. Our RC car uses two motors to power the wheel, which are controlled by the Raspberry Pi and the powerboard using PWM (pulse width modulation). This means that we do not have access to the throttle or brake of it, instead we have direct control over the speed of the car. So in our case, we do not need to know the current speed of the car when changing the speed, and there are only two values of concern: the speed and the steering angle.

Another approach to controlling an RC car needs to be used if the RC car has a different design. RC cars used in [11, 43] do not have steering control. Instead, the turning is controlled by setting different speed to the two sides of the car. For example, if the left wheel(s) is spinning faster than the right wheel(s), the car would turn right. This means that every turn will include some calculation and will be depended on the current speed of the car. This also allows the car to rotate without moving forward, which is convenient in many cases where a normal car needs to reverse and do much more manoeuvres. However, this approach is avoided by us because we wanted our RC car to be more similar to real cars and our controlling methods to be more realistic.

Chapter 3

Project Planning

In this chapter, we discuss the requirements of our project, how to evaluate them, alongside with the initial design of the project and the management. Finally, we discuss the professional, legal, and ethical issues of the project.

3.1 System Requirements

in this section, we talk about the functional requirements and non-functional requirements of our project. All requirements are classified in 4 different priorities: Must have, Should have, Could have, and Will not have.

Must Have (MH): These requirements are the necessity to the project. These requirements must be met otherwise the project cannot be considered as a success.

Should Have (SH): These requirements are essential to the project. Regardless of their importance, not achieving these requirements can be overlooked without effecting the Must Haves of the project.

Could Have (CH): These requirements are not necessary to the project. They provide some extra functionality and would be a nice add-on, but there are no guarantees that they will be achieved.

Will Not Have (WN): These requirements are nice ideas to be included in the project. However, due to technical difficulties, they will most likely not be implemented.

Below is the requirement table for the project. The earlier a requirement appears in the table, the more important it is.

Table 3.1: Functional Requirements

ID	Description	Priority
FR00	The car should be able to be driven by a human. It should be able to record the throttle/steering input of the driver and capture the environment in front of it using a camera.	**MH**
FR01	The car should drive and stay in between two lane lines without human intervention.	**MH**
FR02	The car should stop when an obstacle is present in front of it.	**SH**
FR03	The car should stop when a stop sign is present in a short distance.	**SH**
FR04	The car should stop when no lane line is present.	**SH**
FR05	The car should stop and go at the command of a signal light.	**CH**
FR06	The car should be able to navigate at a crossroad.	**CH**
FR07	The car should change its speed based on the speed sign.	**CH**
FR08	The car should increase its speed when at a straight road and reduce when taking a turn.	**CH**
FR09	The car should be able to behave well with other cars on the road.	**CH**
FR10	The car should be able to handle oncoming traffic on the adjacent lane.	**CH**
FR11	The car should take user input to decide where to go at a crossroad.	**CH**
FR12	The car should have a stereo camera setup for depth estimation.	**CH**
FR13	The car should be able to reverse when in a dead-end.	**WN**
FR14	The car should be able to predict if a pedestrian will be crossing the road based on his/her/their stance.	**WN**
FR15	The car should be able to read indicators by other cars.	**WN**

Table 3.2: Non-Functional Requirements

ID	Description	Priority
N-FR01	The car should always run at a safe speed where it will not damage itself or its surroundings.	**MH**
N-FR02	The car should have a hardware kill switch in case of software malfunction.	**MH**
N-FR03	The processing speed of the car should allow it to stop before hitting a obstacle or passing a stop sign.	**MH**
N-FR04	The camera of the car should should be able to capture images with resolution high enough to recognize important information.	**MH**
N-FR05	The camera of the car should should be able to capture images with a higher frame rate than the processing speed..	**MH**
N-FR06	The car's configuration e.g. max speed, max turning angle should be able to to be tuned	**MH**
N-FR07	The car should have enough power supply to run 10-20 minutes.	**SH**
N-FR08	The car's system should have available recent backup at anytime.	**SH**
N-FR09	This project should be open source after submission.	**SH**

3.2 Project Design

Our self-driving RC car uses a custom kit along with a custom powerboard produced by Xiao-R Geek Technology [46]. More details about the kit and the design of the powerboard can be found at [47]. It is compatible with the Donkey Car software, which will be used as the basis of this program, where modification and improvement will be added on.

The RC car uses rear-wheel drive configuration and a steering engine connected to the front wheels, providing a more realistic control. It uses cameras as the source of information and can be connected to a host computer.

FIGURE 3.1: XR-F1 [47]

After capturing the image, the car will do some image preprocessing on the Raspberry Pi. This includes resolution reduction, grey scaling, and other computations which take little computational power and can reduce image size. After preprocessing, the Raspberry Pi will send the image to the host computer, where predictions will be made. The predicted actions will be sent back to the Raspberry Pi and executed by the RC car.

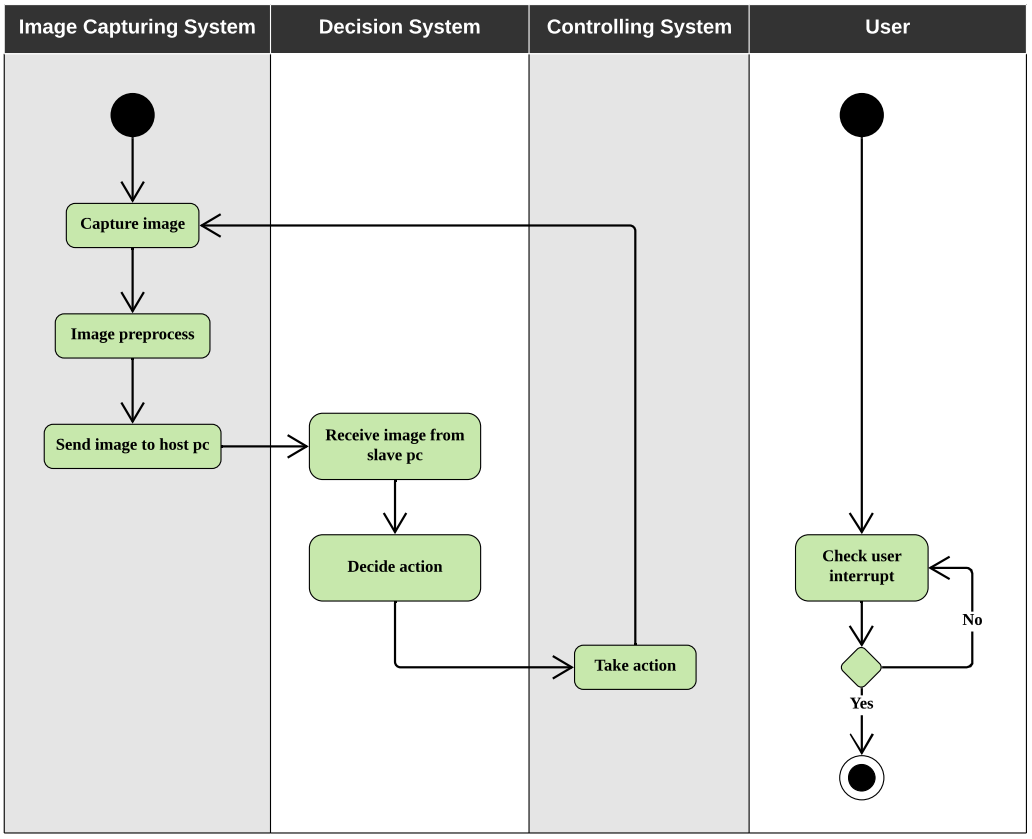


FIGURE 3.2: Activity diagram for driving sequence. The image capturing system and the controlling system will be running on the Raspberry Pi, while the decision system will be running on the host computer.

3.3 Evaluation Strategy

The project can be evaluated with some testing. An environment which includes most of the driving circumstances we want to focus on. The car will be put into that environment and evaluated by monitoring its behaviour.

- The car will be put into a lane with left turns, right turns, and dead ends. We will monitor the time it stayed in the lane and the speed of the car, and how smooth its trajectory is when driving. (FR01, FR04, FR08, FR13)
- An object will be put in front of the car during driving to test if the car can avoid hitting it. (FR02, FR12)
- A stop sign will be put on the side of the lane to test when the car will stop. (FR03, FR12)

- A sign representing a red signal light will be put on the side of the lane. After the car stops, it will quickly be replaced by a sign representing a green light, simulating the change of the signal light. (FR05)
- The car will be put at a crossroad to evaluate its reaction. (FR06, FR11)
- Various speed signs will be put on the side of the lane. We will monitor the speed changes of the car. (FR07)
- Two cars will be put into a lane facing the same direction. We will monitor the behaviour of the following car. (FR09, FR12)
- Two cars will be put into two adjacent lanes facing the same direction. We will monitor the behaviour of both cars. (FR09)
- Two cars will be put into two adjacent lanes facing the opposite direction. We will monitor the behaviour of both cars. (FR09, FR10)
- Paper cutouts of pedestrians with different postures will be put on the side of the lane. We will monitor the behaviour of the car with different postures. (FR14)
- A box with flickering lights on each side will be put on the side of the lane, representing another car attempting to merge/overtake. We will monitor the behaviour of our car driving in the lane. (FR15)

3.4 Project Management

Project management is critical in every project. In this project, we have decided to use Scrum [48].

Scrum is an adaptation of the Agile method. Scrum is separated into small blocks called sprints. Each sprint is usually 2-4 weeks, has a clear objective and isolated from change. A list of features should be made and before each sprint, a feature should be chosen and made the objective of this sprint. After each sprint, there should be a review of the sprint to help improve the progress. Figure 3.3 shows the methodology of Scrum.

Scrum is a value-driven method. Instead of finding all the requirements, finishing all the designs upfront, we will be gathering the requirements as we progress. In each sprint, we will decide which feature we want to work on, and then finish the gathering of the requirements and the design of that specific feature in that sprint. Since the whole project is broken down into features, it is much more flexible and would be easier to find all requirements, since right now we have no definitive idea what all the requirements

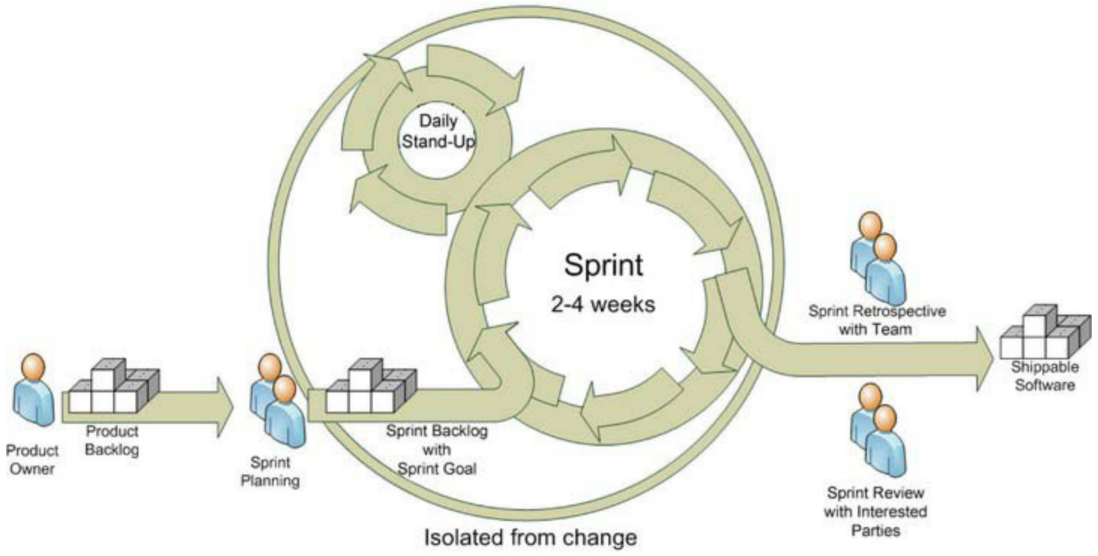


FIGURE 3.3: Scrum methodology [48]

are. This means that the requirements and designs mentioned above, as well as the project planning below, are all subject to change.

Another methodology is the waterfall method, which is a plan-driven method. This will be avoided by us because it requires finishing the design at the beginning of the project. Finding all requirements and designs before doing the project is very hard and easy to miss some important requirements, and the waterfall method makes it hard to add new requirements when doing the project because the timeline has already been filled.

3.4.1 Timetable

Figure 3.4, figure 3.5, and figure 3.6 are the initial timelines we have made, represented as Gantt charts, created using TeamGantt. It consists of three parts: stage 1, stage 2, and the semester 1 break in between. The semester 1 break is considered as a whole instead of different sprints because what happens during the break is very unpredictable. We will keep it to the best of our ability, but it will be changed when encountering additional features and requirements which we did not notice now.

3.4.2 Risk analysis and mitigation

The risks will be classified with two standards: its impact on the project, and the possibility of it happening. A higher impact risk draws more attention and will at least be avoided. A higher possibility risk should at least be planned in the likely case of happening (Table 3.3).

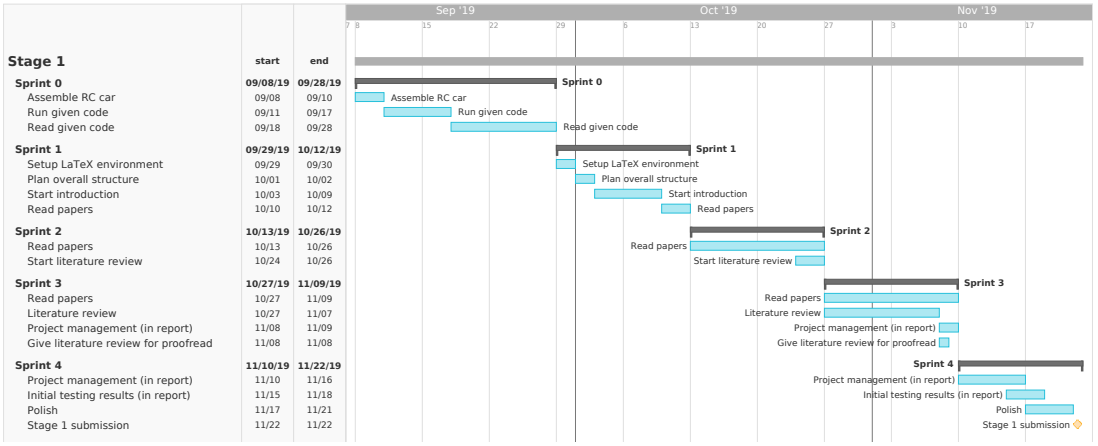


FIGURE 3.4: Stage 1

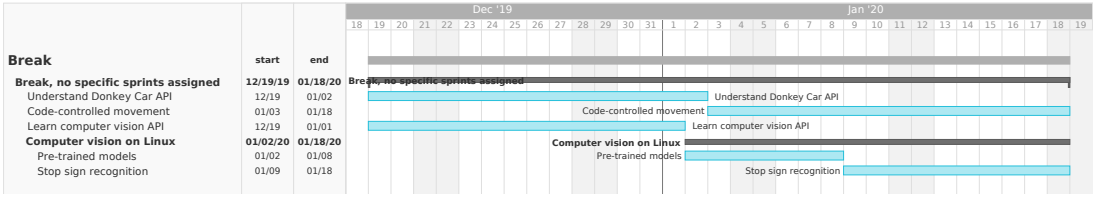


FIGURE 3.5: Semester 1 break

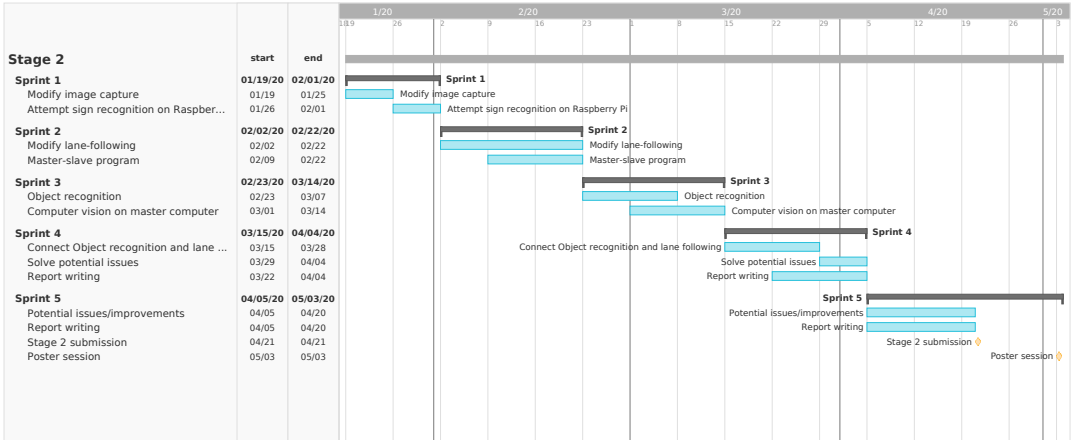


FIGURE 3.6: Stage 2

TABLE 3.3: Risk Classification

	High Impact	Medium Impact	Low Impact
High Possibility	**Avoided with a backup plan**	Avoided	Mitigation planned
Medium Possibility	**Avoided with a backup plan**	Avoided	Mitigation planned
Low Possibility	Avoided	Mitigation planned	Ignored for now

Table 3.4: Risk Identification

ID	Risk Description	Impact	Possibility	Action
R1	The powerboard may be broken due to accidents e.g. short circuit.	Medium	Low	Mitigation
R2	The Raspberry Pi we are using might be burned due to the lack of a fuse when whe power is supplied via GPIO pins.	Medium	Low	Mitigation
R3	Parts of the car might be broken due to miscalculated operations.	Medium	Medium	Avoidance
R4	The SD car the Raspberry Pi uses and the data stored in it might be corrupted or system can be tempered.	High	High	Avoidance with backup plan
R5	The car might exceed its speed limit due to misinstruction, causing damage to itself or surroundings.	High	Medium	Avoidance with backup plan

Table 3.5: Avoidance/Mitigation Plans

ID	Plan
R1	A new one can be bought with no change to the software needed.
R2	A new one can be bought with no change to the software needed.
R3	Tune the car before starting any work on it. By tuning the car, we will be able to tell the limitations of the car e.g. maximum turning angle. The limitations will be checked carefully before performing operations.
R4	System changes will be done carefully. Avoided illegal actions such as removing SD card without properly powering off. Multiple backup images of the SD card need to be created regularly in case of data corruption. Regular backups will be performed weekly.
R5	Tune the car and perform testing with wheels off ground before running it.

Table 3.4 is a list of risks we have identified and table 3.5 is the mitigation plans made.

3.4.3 Other Issues

Professional and Legal Issues

All papers and sources used in this dissertation are properly cited. Codes not created by us will be marked and original authors will be acknowledged. Papers used are either open access or have been granted access. Codes and software used are provided with the proper license. This project will not be used for commercial purposes and will be published under the MIT License once completed.

Ethical and Social Issues

This dissertation includes no human subject. The datasets used include no personal or sensitive information. The RC car will be limited to a safe speed and will not endanger any observer or its surrounding. No observer will be forced to stay against their will. This project aims to develop a low-cost yet reliable self-driving system. This can help the development of self-driving technologies and make autonomous vehicles more affordable.

Chapter 4

Initial Testing

As mentioned in the project design, our implementation of the self-driving car is based on Donkey Car, so we figured it would be a good idea to try to run the original Donkey Car before we expand to it. In this chapter, we briefly explain the experience of running the Donkey Car.

The car is assembled according to the instructions in [47]. The code we used is provided at [49] for Raspberry Pi and [50] for Ubuntu. Those are some slightly older versions of the Donkey Car, tuned to run our model of the car. The Donkey Car and the environment are set up using instructions in [47].

After launching the Donkey Car, the RC car can be controlled with a web interface. We set up the provided track, and run the car on the track to collect training data. Then, we took the training data to a Linux machine with Donkey Car installed. We trained the model in the Linux machine using the training data collect and copied the generated model back to the Raspberry Pi. Then Donkey Car can be launched again with the option to drive by its own. Now the car can be driven by the model we trained, and it will stay in between of the two lines automatically.

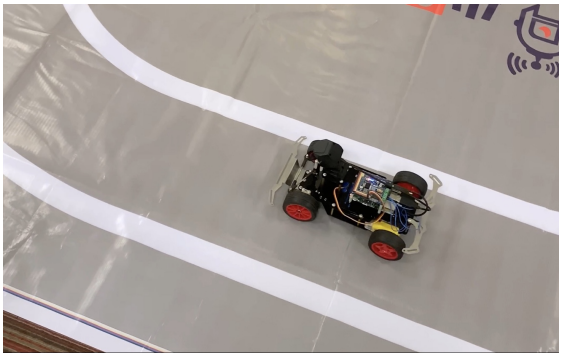


FIGURE 4.1: A video of the car automatically driving in the track can be found at https://youtu.be/VIqjQR9-1Q8

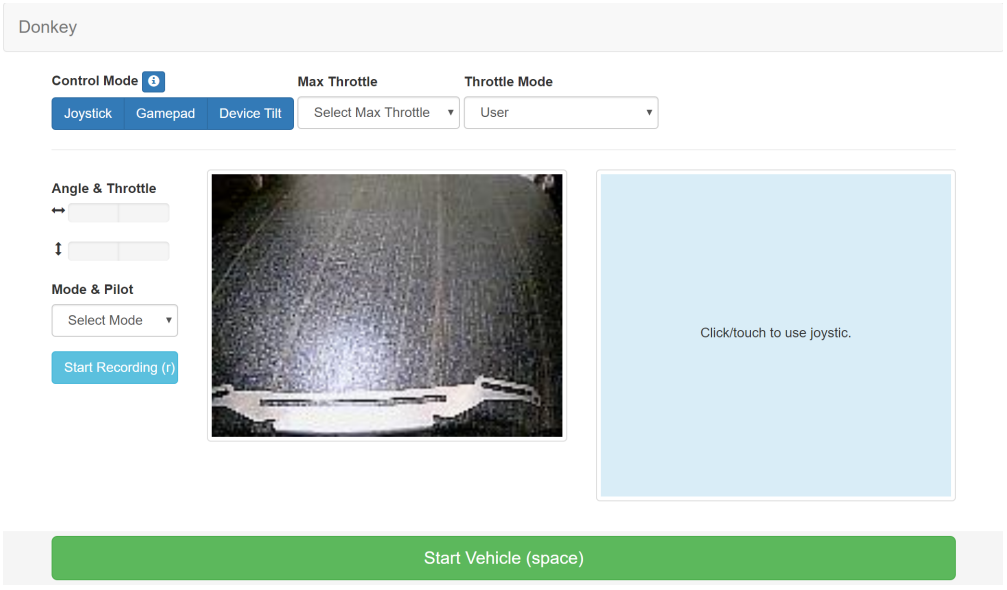


FIGURE 4.2: Web interface used for controlling the car

The car uses a 120×160 RGB image as the image input, and train it on a convolutional neural network with 5 convolutional layers. Each layer's feature number, stride, and activation function is easily configurable. After passing through 5 convolutional layers, the features are flattened and passed to 2 fully connected layers, where the features will be classified into 50 features, while randomly dropping 10% of the features twice to prevent overfitting. Then these features are used to predict steering angle and throttle.

Below is a code fragment from [49] showing the structure of the network.

```
x = img_in
x = Convolution2D(24, (5, 5), strides=(2, 2), activation='relu')(x)
x = Convolution2D(32, (5, 5), strides=(2, 2), activation='relu')(x)
x = Convolution2D(64, (5, 5), strides=(2, 2), activation='relu')(x)
x = Convolution2D(64, (3, 3), strides=(2, 2), activation='relu')(x)
x = Convolution2D(64, (3, 3), strides=(1, 1), activation='relu')(x)
x = Flatten(name='flattened')(x)
x = Dense(100, activation='relu')(x)
x = Dropout(.1)(x)
x = Dense(50, activation='relu')(x)
x = Dropout(.1)(x)
```

4.1 Limitations

Although the car currently has the function of lane following, it is far from what we want to achieve. The reaction time of the car is rather slow, making it not able to react on time when at a high speed. The camera setup currently causes the car can only see one line when turning. This is not optimal because it causes problems such as when no proper lane marking is present, but the car will still attempt to drive. Currently, the decision making model is running on the Raspberry Pi completely. As we add more features to the model, the computation power of a Raspberry Pi may be insufficient, and we would need to implement a work offloading procedure to diverge some computation to a more powerful device.

References

[1] H. Moravec, "The Stanford Cart and the CMU Rover," *Proceedings of the IEEE*, vol. 71, no. 7, p. 872–884, 1983.

[2] Stanford University, "Stanford's robotics legacy," May 2019. [Online]. Available: https://news.stanford.edu/2019/01/16/stanfords-robotics-legacy/ [Accessed: 22-11-2019]

[3] SAE International, "Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles," Jun 2018. [Online]. Available: https://www.sae.org/standards/content/j3016_201806/ [Accessed: 22-11-2019]

[4] S. Dominguez, B. Khomutenko, G. Garcia, and P. Martinet, "An optimization technique for positioning multiple maps for self-driving car's autonomous navigation," in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, Sep. 2015, pp. 2694–2699.

[5] Baidu Baike, "SLAM." [Online]. Available: https://baike.baidu.com/item/SLAM/7661974?fr=aladdin [Accessed: 22-11-2019]

[6] W. dong CHEN and F. ZHANG, "Review on the achievements in simultaneous localization and map building for mobile robot," vol. 22, no. 3, pp. 455–460, 2005.

[7] Velodyne Lidar, "Alpha puck^{TM}." [Online]. Available: https://velodynelidar.com/vls-128.html [Accessed: 22-11-2019]

[8] tangf2004, "Discussion regarding the definition of high definition map (title translated)," Jul 2018. [Online]. Available: https://blog.csdn.net/tangf2004/article/details/81162489 [Accessed: 22-11-2019]

[9] C. Kerl, J. Sturm, and D. Cremers, "Dense visual SLAM for RGB-D cameras," *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013.

[10] D. Cain, B. Layng, K. McNulty, R. O'Connor, and T. Estrada, "Design of an autonomous, line followingpace car for athletic training," *ASEE2014 Zone I Conference*, Apr 2014.

[11] V. Mougios, "Arduino controlled car with hc-sr04," Jan 2017. [Online]. Available: https://create.arduino.cc/projecthub/Bill_Mougios/arduino-controlled-car-with-hc-sr04-a8e57d [Accessed: 22-11-2019]

[12] M. Cho, "A study on the obstacle recognition for autonomous driving RC car using LiDAR and thermal infrared camera," in *2019 Eleventh International Conference on Ubiquitous and Future Networks (ICUFN)*, July 2019, pp. 544–546.

[13] R. D. Henry, "Automatic ultrasonic headway control for a scaled robotic car," 2001. [Online]. Available: http://hdl.handle.net/10919/36262

[14] A. Tőkés, "Stereo vision and lidar powered donkey car," Apr 2019. [Online]. Available: https://www.hackster.io/bluetiger9/stereo-vision-and-lidar-powered-donkey-car-575769 [Accessed: 22-11-2019]

[15] W. Shi, M. B. Alawieh, X. Li, and H. Yu, "Algorithm and hardware implementation for visual perception system in autonomous vehicle: A survey," *Integration*, vol. 59, p. 148–156, 2017.

[16] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, June 2005, pp. 886–893 vol. 1.

[17] M. Lorentzon, "Feature extraction for image selection using machine learning," 2017. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-142095 [Accessed: 22-11-2019]

[18] G. Antipov, S.-A. Berrani, N. Ruchaud, and J.-L. Dugelay, "Learned vs. hand-crafted features for pedestrian gender recognition," *Proceedings of the 23rd ACM international conference on Multimedia - MM 15*, 2015.

[19] D. Gehrig, A. Loquercio, K. G. Derpanis, and D. Scaramuzza, "End-to-end learning of representations for asynchronous event-based data," *CoRR*, vol. abs/1904.08245, 2019. [Online]. Available: http://arxiv.org/abs/1904.08245 [Accessed: 22-11-2019]

[20] G. Gallego, T. Delbrück, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. J. Davison, J. Conradt, K. Daniilidis, and D. Scaramuzza, "Event-based vision: A survey," *CoRR*, vol. abs/1904.08405, 2019. [Online]. Available: http://arxiv.org/abs/1904.08405 [Accessed: 22-11-2019]

[21] A. Z. Zhu, D. Thakur, T. Özaslan, B. Pfrommer, V. Kumar, and K. Daniilidis, "The multivehicle stereo event camera dataset: An event camera dataset for 3d perception," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2032–2039, July 2018.

[22] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org [Accessed: 22-11-2019]

[23] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015. [Online]. Available: http://neuralnetworksanddeeplearning.com/index.html [Accessed: 22-11-2019]

[24] Stanford University. Cs231n convolutional neural networks for visual recognition. [Online]. Available: http://cs231n.github.io/convolutional-networks/ [Accessed: 22-11-2019]

[25] A. Wiranata, S. A. Wibowo, R. Patmasari, R. Rahmania, and R. Mayasari, "Investigation of padding schemes for faster r-cnn on vehicle detection," in *2018 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC)*, Dec 2018, pp. 208–212.

[26] C.-F. Wang, "A basic introduction to separable convolutions," Aug 2018. [Online]. Available: https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728 [Accessed: 22-11-2019]

[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017. [Online]. Available: http://doi.acm.org/10.1145/3065386 [Accessed: 22-11-2019]

[28] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861 [Accessed: 22-11-2019]

[29] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun 2018. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2018.00474 [Accessed: 22-11-2019]

[30] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015, pp. 1–9.

[31] Z. Ni, G. Bian, X. Xie, Z. Hou, X. Zhou, and Y. Zhou, "Rasnet: Segmentation for tracking surgical instruments in surgical videos using refined attention

segmentation network," *CoRR*, vol. abs/1905.08663, 2019. [Online]. Available: http://arxiv.org/abs/1905.08663 [Accessed: 22-11-2019]

[32] N. Ma, X. Zhang, H. Zheng, and J. Sun, "Shufflenet V2: practical guidelines for efficient CNN architecture design," *CoRR*, vol. abs/1807.11164, 2018. [Online]. Available: http://arxiv.org/abs/1807.11164 [Accessed: 22-11-2019]

[33] NVIDIA, "Jetson Nano developer kit," Oct 2019. [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano-developer-kit [Accessed: 22-11-2019]

[34] D. Retana, "Getting started in AI and computer vision with Nvidia Jetson Nano." Oct 2019. [Online]. Available: https://towardsdatascience.com/getting-started-in-ai-and-computer-vision-with-nvidia-jetson-nano-df2cacbd291c [Accessed: 22-11-2019]

[35] Google, "USB accelerator." [Online]. Available: https://coral.withgoogle.com/products/accelerator/ [Accessed: 22-11-2019]

[36] Intel, "Intel neural compute stick 2," Oct 2019. [Online]. Available: https://software.intel.com/en-us/neural-compute-stick [Accessed: 22-11-2019]

[37] D. Tian, "DeepPiCar - part 1: How to build a deep learning, self driving robotic car on a shoestring budget," May 2019. [Online]. Available: https://towardsdatascience.com/deeppicar-part-1-102e03c83f2c [Accessed: 22-11-2019]

[38] Z. Wang, "Self driving RC car," Aug 2015. [Online]. Available: https://zhengludwig.wordpress.com/projects/self-driving-rc-car/ [Accessed: 22-11-2019]

[39] F. Lambert, "Tesla claims to have 'world's most advanced computer for autonomous driving' with autopilot 3.0 update coming next year," Aug 2018. [Online]. Available: https://electrek.co/2018/08/01/tesla-chip-most-advanced-computer-autonomous-driving-autopilot-hardware-3-update/ [Accessed: 22-11-2019]

[40] G. Bagwe, "Video frame reduction in autonomous vehicles," 01 2018.

[41] L. Kang, W. Zhao, B. Qi, and S. Banerjee, "Augmenting self-driving with remote control: Challenges and directions," in *Proceedings of the 19th International Workshop on Mobile Computing Systems & Applications*, ser. HotMobile '18. New York, NY, USA: ACM, 2018, pp. 19–24. [Online]. Available: http://doi.acm.org/10.1145/3177102.3177104 [Accessed: 22-11-2019]

[42] X. Liu, X. Xu, and B. Dai, "Vision-based long-distance lane perception and front vehicle location for full autonomous vehicles on highway roads," *Journal of Central South University*, vol. 19, no. 5, pp. 1454–1465, May 2012. [Online]. Available: https://doi.org/10.1007/s11771-012-1162-7 [Accessed: 22-11-2019]

[43] A. Garg, "How to make line follower robot using arduino," Apr 2019. [Online]. Available: https://www.instructables.com/id/Line-Follower-Robot-Using-Arduino-2/ [Accessed: 22-11-2019]

[44] Donkey Car, "Donkey car." [Online]. Available: https://www.donkeycar.com/ [Accessed: 22-11-2019]

[45] Lean Systems, "Machine learning vs. mathematical modelling in practice," May 2018. [Online]. Available: https://www.leansystems.co/blog/machine-learning-vs-mathematical-modelling [Accessed: 22-11-2019]

[46] "Xiao-r technology." [Online]. Available: http://www.xiao-r.com/ [Accessed: 22-11-2019]

[47] "Xr-f1 tutorials and datasheets." [Online]. Available: http://www.xiao-r.com/index.php/Study/catalog/cid/29/ [Accessed: 22-11-2019]

[48] J. Blankenship, M. Bussa, and S. Millett, *Managing Agile Projects with Scrum*. Berkeley, CA: Apress, 2011, pp. 13–27. [Online]. Available: https://doi.org/10.1007/978-1-4302-3534-7_2 [Accessed: 22-11-2019]

[49] "donkeycar-pi for XR-F1." [Online]. Available: https://github.com/991693552/donkeycar-pi [Accessed: 22-11-2019]

[50] "donkeycar for linux." [Online]. Available: https://github.com/991693552/donkeycar-linux [Accessed: 22-11-2019]