

**IMPLEMENTATION OF INTERNET OF THINGS(IOT) ON CAR
PLATE RECOGNITION SYSTEM WITH DATABASE FOR
SURVEILLANCE PURPOSE**

ANTHONY LAI CHI AN



اونيورسيتي تيكنيكل مليسيا ملاك

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

**IMPLEMENTATION OF INTERNET OF THINGS(IOT) ON CAR
PLATE RECOGNITION SYSTEM WITH DATABASE FOR
SURVEILLANCE PURPOSE**

ANTHONY LAI CHI AN

**This report is submitted in partial fulfillment of the requirements
for the degree of Bachelor of Electronic Engineering with Honours**



اونيورسيتي تیکنیکل ملیسيا ملاک

Faculty of Electronic and Computer Engineering

Universiti Teknikal Malaysia Melaka

JUNE 2020

DECLARATION

I declare that this report entitled “Implementation of Internet of Things on Car Plate Recognition System For Surveillance Purpose” is the result of my own work except for quotes as cited in the references.



Signature : اونیورسیتی تکنیکل ملیسیا ملاک

Author : Anthony Lai Chi An

Date :24 June 2020.....

APPROVAL

I hereby declare that I have read this thesis and in my opinion, this thesis is sufficient in terms of scope and quality for the award of Bachelor of Electronic Engineering with Honours.



اونيورسيتي تيكنيكل مليسيا ملاك

Signature :

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

Ir. Dr. Ridza Azri Ramlee
Senior Lecturer

Supervisor Name :

..... Fac. of Electronics & Comp. Eng. (FKEKK),
Universiti Teknikal Malaysia Melaka (UTeM),
Hang Tuah Jaya, 76100 Durian Tunggal,
Melaka, Malaysia

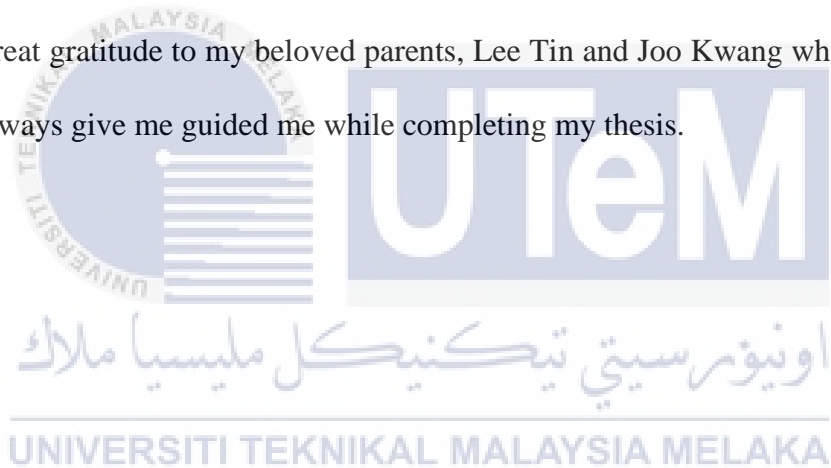
Date :

24 June 2020

.....

DEDICATION

I would like to dedicate my work to my friends and my project supervisor. I also feel great gratitude to my beloved parents, Lee Tin and Joo Kwang whose supportive and always give me guided me while completing my thesis.



ABSTRACT

The project aims to develop an application to associate with car plate recognition to check a vehicle validity on a database over the internet. Node.js as API is used to set up the code environment, while MQTT is used to do message publishing and subscribing. As for database checking, the time-series based Influx database is used. One of the challenges in this research is lowering the cost of setting up this app and the performance of the system over time. To decrease the cost of this project, only free and open-sourced protocols and programs are used in this project. Some system might have data congestion over time due to delay or loss of network connections that leads to an overload of unprocessed data accumulating in a network. In this case, Test-Driven Development (TDD) practice was implemented while programming to make the code clearer, simpler for execution, and bug-free. Unit tests were implemented to check the performance of the system to verify the consistency of this system over a specified period. The unit test results are taken and put in into a graph of easier view for descriptive analysis. The result shows that the program developed runs on zero product costs and able to obtain consistent working performance for a test period of 2 months.

ABSTRAK

Projek ini bertujuan untuk mengembangkan aplikasi untuk mengaitkan dengan pengenalan plat kereta untuk memeriksa kesahan kenderaan di pangkalan data melalui internet. Node.js sebagai API digunakan untuk mengatur lingkungan kod, sementara MQTT digunakan untuk melakukan penerbitan pesan dan berlangganan. Untuk pemeriksaan pangkalan data, pangkalan data Influx berdasarkan siri masa digunakan. Salah satu cabaran dalam penyelidikan ini adalah menurunkan kos penyediaan aplikasi ini dan prestasi sistem dari masa ke masa. Untuk mengurangkan kos projek ini, hanya protokol dan program sumber terbuka yang digunakan dalam projek ini. Beberapa sistem mungkin mengalami kesesakan data dari masa ke masa kerana kelewatan atau kehilangan sambungan rangkaian yang menyebabkan kelebihan data yang belum diproses terkumpul di dalam rangkaian. Dalam kes ini, praktik Pengujian Bergerak Uji (TDD) dilaksanakan semasa pengaturcaraan untuk menjadikan kod lebih jelas, lebih mudah untuk pelaksanaan dan bebas bug. Uji unit dilaksanakan untuk memeriksa prestasi sistem untuk mengesahkan konsistensi sistem ini dalam jangka masa yang ditentukan. Hasil ujian unit diambil dan dimasukkan ke dalam grafik pandangan yang lebih mudah untuk analisis deskriptif. Hasilnya menunjukkan bahawa program yang

dikembangkan berjalan pada kos produk secara percuma dan dapat memperoleh prestasi kerja yang konsisten dalam jangka masa 2 bulan.



ACKNOWLEDGMENTS

I would like to express my gratitude towards my research supervisor, Ir. Dr. Ridza Azri Ramlee for continuous support and advice while writing the dissertation. His guidance helped me in correcting grammar errors and wrong placement of contents. I learned the basic confinement of writing an actual thesis.

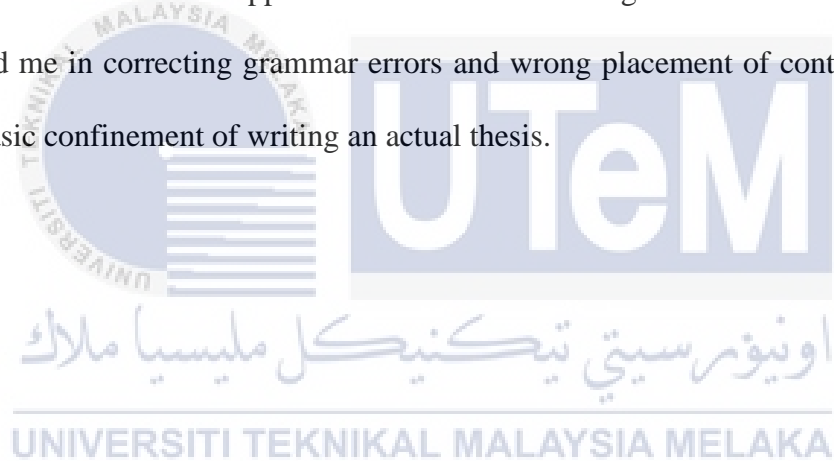


TABLE OF CONTENTS

Declaration	ii
Approval	i
Dedication	i
Abstract	i
Abstrak	ii, iii
Acknowledgments	iv
Table of Contents	v, vi, vii
List of Figures	viii, ix
List of Tables	x
List of Symbols and Abbreviations	xi
List of Appendices	xiii
CHAPTER 1 INTRODUCTION	1
1.1 Project Outcome	1
1.2 Problem Statement	2
1.3 Objective	2
1.4 Scope of Work	2
1.5 Report Organization	3
CHAPTER 2 BACKGROUND STUDY	5
2.1 Car plate Recognition	5

2.1.1	Image Pre-processing	8
2.1.2	Pattern Matching Optical Character Recognition	10
2.1.3	Convolutional Neural Network (CNN)	12
2.1.4	Comparison between CNN and Pattern matching	12
2.2	Internet of Things (IoT) in The Surveillance System	13
2.2,1	Application Program Interface (API)	14
2.2.2	Communication Protocol	14
2.2.3	Comparison Between Node.js and PHP	15
2.3	Database in The Surveillance System	15
2.3.1	Comparison Between InfluxDB and Firebase	16
CHAPTER 3 METHODOLOGY		18
3.1	Project Implementation Flowchart	18
3.2	Software Flowchart	21
3.3	Car Plate Recognition Part Methodology	23
3.4	IoT Part Methodology	24
3.4.1	Excel Format	24
3.4.2	Including Protocols	25
3.4.3	Setting Up a Broker	25
3.4.3.1	Error Setting Up Broker On Non-listening Port	27
3.4.4	Publishing Message – MQTT	27
3.4.5	Subscribing Message – MQTT	28
3.4.6	Correcting JSON Format	29
3.4.7	Extract Information From Database	29
3.4.7.1	Managing Plain Result From Database	30
3.4.7.2	Setting Up a Condition For Vehicle Check	30

3.5	Database Part Methodology	31
3.5.1	Database Set Up	31
3.5.2	Writing Information Into Database	34
3.5.2.1	Duplicated Message	34
3.5.2.2	Influx Database Error Writing Data	35
3.5.2.3	Inserting Timestamp	36
3.6	Program Installation	36
CHAPTER 4 RESULTS AND DISCUSSION		38
4.1	Car Plate Recognition Part	38
4.1.1	Desired Car Plate Result	38
4.1.2	Simple Car Plate Recognition Test	40
4.1.3	Non-ideal Angle of Car Plate Recognition	40
4.1.3.1	Tilted Car Plate Image	41
4.1.3.2	Side Image of Car Plate Recognition	42
4.2	App Performance Test	43
4.2.1	First Part App Test	45
4.2.2	Second Part App Test	50
CHAPTER 5 CONCLUSION AND FUTURE WORKS		55
5.1	Conclusion	55
5.2	Future Work	56
REFERENCES		58
APPENDICES		61

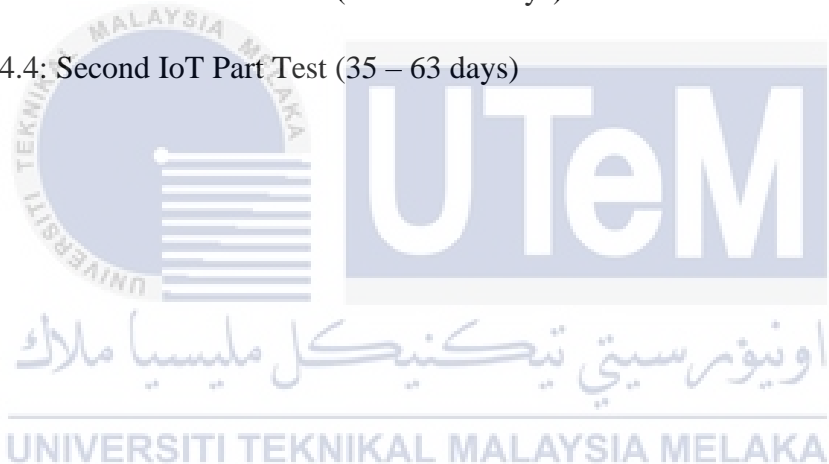
LIST OF FIGURES

Figure 2.0: Original Image(left) and Greyscale Image(right)	9
Figure 2.1: Greyscale Image(left) and Binary Image(right)	9
Figure 2.2: Binary Matrix of a Segmented Character	10
Figure 2.3: Pattern Matching Algorithm on Image	11
Figure 2.4: The Binarized Image and Detected Character on MATLAB	11
Figure 3.1: Project Implementation Flowchart	19
Figure 3.2: The Block Diagram of the Hardware System	20
Figure 3.3: Software Flowchart	21
Figure 3.4: Overview of Project	22
Figure 3.5: Original Image before Image Treatment	23
Figure 3.6: Output Image after Image Treatment	23
Figure 3.7: The recorded Data in Excel File	24
Figure 3.8: Error: xlsx Function is Not Defined	25
Figure 3.9: List of Port Connections	26
Figure 3.10: Terminal Message to Show Broker is Ready	27
Figure 3.11: Error Setting Up Broker	27
Figure 3.12: Terminal Message After Publishing Topic	28
Figure 3.13: Error while Extracting Information From Influx Database Using String	29

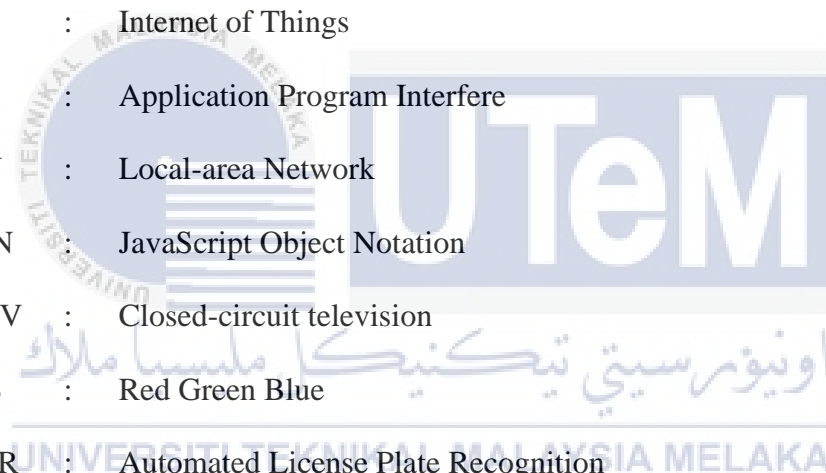
Figure 3.14: The Complete Output if Extracted Information is Not Managed	30
Figure 3.15: Desired Result from Influx Query	31
Figure 3.16: Influx Database Folder	32
Figure 3.17: Running “influx.exe” in Background	33
Figure 3.18: “influx.exe” First Impression	33
Figure 3.19: Error: Database is required.	34
Figure 3.20: Inserting Data into Database	34
Figure 3.21: Duplicated Feedback	35
Figure 3.22: Message Re-arranged	35
Figure 3.23: Error Inserting Data Without Value	35
Figure 3.24: Sample Query To Insert Specified Timestamp	36
Figure 4.0: Desired OCR Output	39
Figure 4.1: Undesired OCR Output	39
Figure 4.2: (a) Car Plate (b) OCR Test Result	40
Figure 4.3: (a) Tilted Car Plate (b) OCR Test Result	41
Figure 4.4: Top View Illustration of the Condition	42
Figure 4.5: (a) Side Image of Car Plate (b) OCR Result	42

LIST OF TABLES

Table 2.1: Comparison of Firebase and Influx Database	17
Table 4.1: First IoT Part Test (Initial – 28 days)	45
Table 4.2: First IoT Part Test (35 – 63 days)	47
Table 4.3: Second IoT Part Test (Initial – 28 days)	50
Table 4.4: Second IoT Part Test (35 – 63 days)	52



LIST OF SYMBOLS AND ABBREVIATIONS



OCR	:	Optical Character Recognition
BoF	:	Bag of Features
CNN	:	Convolutional Neural Network
IoT	:	Internet of Things
API	:	Application Program Interfere
LAN	:	Local-area Network
JSON	:	JavaScript Object Notation
CCTV	:	Closed-circuit television
RGB	:	Red Green Blue
ALPR	:	Automated License Plate Recognition
TDD	:	Test-driven Development
GPU	:	Graphic Processing Unit

LIST OF APPENDICES

Appendix A: MATLAB Face Recognition Source Code	61
Appendix B: Node.js First IoT Part Source Code	62
Appendix C: Node.js Second IoT Part Source Code	65



CHAPTER 1

INTRODUCTION



1.1 Project Overview

This project is about the development of a prototype to implement IoT on a car plate recognition system for vehicle surveillance purposes. By using MATLAB software, the prototype will detect the character on the car plate from an image and a Node.js based application will communicate the result with a database for validity checking over the internet.

1.2 Problem Statement

In this era, the number of vehicles is increasing fast, following the crime rate associated with the vehicle. While the Internet of Things (IoT)-connected technologies are getting famous drastically over the years to save human power, a low-cost and efficient IoT associated vehicle surveillance system is favored.

According to the 2018 Node.js User Survey Report, developers have been using Node.js mostly for front-end development (webpage design) and full-stack [15]. In this research, developing a product-cost free back-end program with IoT feature that shows consistent performance will be focused. Only free protocols and programs are used in the research. Unit test over 2 months will be implemented to check for its system consistency.

1.3 Objective

- (i) To develop a system that can detect car plate number using Optical Character Recognition (OCR) in an ideal condition.
- (ii) To develop an algorithm for extracting car plate owner's information registered in the database.
- (iii) To establish a local-based database to check the authenticity of visitors entering a premise.

1.4 Scope of Work

This project is focused on the development of an application that could associate with a car plate recognition system to communicate with a database through the internet to check for a vehicle's owner information.

A MATLAB built-in OCR model is used to recognize a manually segmented Malaysia car plate. 10 samples of car plates are used to run the OCR application. 5 car plates image taken from phone camera and the other 5 images taken from the internet.

The program is built with the only cost-free products and cost-free protocols. For the application's performance test, a sample of 40 units test will be taken every 7 days for 2 months, on both subscribing and publishing part of the IoT application. Average value will be calculated every 7 days and a graph will be plotted for descriptive analysis. The program will be running continuously on the laptop during the test period without running any other program other than this IoT program.

1.5 Report Organization

There are five chapters in this thesis. Chapter 1 introduces the summary of the car plate recognition for surveillance purposes and mentioned the method together with the software used. It also included problem statement, the scope of work, and objective of this project.

Chapter 2 explains the background of this project and introduces MATLAB, and OCR and Bag of Features. This chapter discusses methods of deep learning approach and compares the method between OCR and CNN. Other than that, this chapter also discusses application program interface (API), comparing the latest Node.js with traditional PHP in terms of speed, function, and safety. Then a background study about Node.js, JavaScript, Visual Studio Code, MQTT protocol, and other software used in this project. The database to be used in the project is also discussed later in Chapter 2. Chapter 2 compares Influx database and Firebase in terms of its function, the way of data is saved and accessibility of the database from the chosen API.

Chapter 3 discusses the method used in this project. Hardware and software flowcharts are listed here. The implementation of software in this project is briefly described. A software overview of this project is shown in this chapter to explain the relationship between each software in this project. This chapter also includes steps of troubleshooting the software. Other than that, the steps to do the installation of any related software is listed in this chapter to make sure every protocol and documentation that is required to make sure the source code works in this project is included.

Chapter 4 will show the result of this project, and at the same time discuss problems met while building the app. This chapter will explain the meaning of some source code and how some issue is solved using different protocols. This chapter will also pinpoint some detail while running the app to prevent error or delay. This chapter is divided into 3 main parts to make sure the discussion is in sequence and clear. The parts are the IoT part, Car Plate Recognition Part, and Database part.

Chapter 5 summarizes the objectives achieved by this project and points out some flaws in this project that can be improved in the future. Other than that, chapter 5 will suggest some fields or systems that could inherit this system capability.

CHAPTER 2

BACKGROUND STUDY



2.1 Car plate recognition in vehicle surveillance system

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

With the 4.0 industry revolution today, IT technology such as automation and artificial intelligence has been developed rapidly. Several kinds of innovative devices had been made for video surveillance under security purposes. For example, the well-known closed-circuit television (CCTV), it uses video cameras to capture real-time recording and transfer the recording to monitors. Earlier, the poor quality and high installation cost have limited the development of its application. But the good news is now that CCTV is improved, it has a better frame per second (FPS) rates, and higher video resolution, more applications could be implemented. The famous creation recently is the dashboard camera that can be installed in a car, it works as a continuously recording camera or called loop recording to record mostly a front view of the car through

the windscreen. The dashboard cameras can provide strong evidence to be used in court in case of any unwanted event like road accidents and vandalism. Therefore, most car drivers opt to equip a dashboard camera in their vehicle. In some countries, the installation of dashboard cameras was demanded especially on public transportation.

The only difference between a CCTV and a regular dashboard camera is that they have different mobility. For most of a time, a CCTV is set up in fixed areas that needed high-security priority, such as gold shops, banks, colleges, and areas where monitoring is needed. Therefore, its mobility is low and the coverage is low. As for a car dashboard camera, it is set up in a car. The camera can make recording wherever the car goes. In the case of CCTV, the advantage is that its hardware performance is higher since it is fixed on a spot. The monitoring is relatively more efficient in terms of detecting car movement in a specified area.

Numerous researches have been done to optimize the application of CCTV for surveillance purposes. The car plate tracking system based on videos often suffers from a similar problem, the accuracy and time taken to recognize the characters on car plate, and how the detected character communicate with a database to determine whether the vehicles are authorized to be in the area. To solve this issue, we propose an OCR based system that could do a car plate recognition system from a video source, that can check with a database from video source for availability of a vehicle in that area. In this project, the system is developed to utilize CCTV, as the effectiveness of car plate detecting depends on diverse information, different font, and arrangement of plate characters. Fortunately, the development of the IT industry today can easily process images using image treatment and machine learning techniques.

Our system can be implemented to handle previous surveillance works that are both time-consuming and higher labor costs. For example, an automated gate can determine the availability of a vehicle to enter the closed area by reading the car plate through the CCTV. Human monitoring through CCTV and looking through the database would require significant amounts of effort and time, not to mention human error in case of health, mental, or other issues. In the case of our system, based on the car plate number, we can quickly determine if a car is authorized to be in the area. Besides, our system can improve the efficiency of crime detection, for example, a stolen car, by speeding up the location tracking process through the CCTV.

However, despite the advantages of our proposed system, the system is not that easy to be implemented. Firstly, the system should have large storage to store vehicle and owner's information and a very good processing speed to handle the big data involved, imagine recording through a video device for 24 hours every day. The volume of space or data consumed is very huge. Therefore, the system should be capable of handling big data to avoid any delay inside the system else the system might just stop. Secondly, the system might misrecognize a car plate due to light lamination, different fonts of car, dirt covering characters on the car plate. Thirdly, a good segmentation method should be developed to efficiently detect and extract the car plate from the images.

2.1.1 Image Pre-processing

Image Processing Toolbox is used to do image pre-processing before running them through OCR. This toolbox provides a comprehensive set of reference-standard algorithms and workflow apps for image processing, analysis, visualization, and algorithm development.[18] Basic and useful functions like, grey thresh conversion, binarization conversion, and RGB conversion can be found in the toolbox. We can perform image segmentation, image enhancement, noise reduction, geometric transformations, image registration, and 3D image processing.[18]

This toolbox allows you to operate a basic image processing task. Segmentation of image, image data comparison, registration comparison techniques, processing big image data. The visualization feature also consists of applications that let you explore images, 3D volumes, and videos; adjust contrast; create histograms; and manipulate regions of interest (ROIs).[18] In Salman's research,[19] he concludes that MATLAB is a very useful and easy-to-use platform, especially the image processing toolbox he used in his research.

However, this project pinpoints on developing an IoT application instead of enhancing the current OCR system. Therefore, the car plate area is manually segmented and extracted from the original image before the image undergoes image treatment. This treatment converts the true color image RGB to the grayscale image. The process is to eliminate the hue and saturation information while retaining the luminance. A formula of 30% Red, 60% Green and 10% Blue is used to convert the original image to greyscale image.

$$\text{Greyscale Image} = (0.30 * \text{Red} + 0.60 * \text{Green} + 0.10 * \text{Blue}) * (\text{Original Image})$$



Figure 2.0 Original image(left) and greyscale image(right)

After that the image is scaled from 0 to 1 according to their grey thresh value on each pixel, gray threshold value less than 0.5 became 0 and the other becomes 1. The figure below shows the comparison of a greyscale image and a binarized image.



Figure 2.1: Greyscale Image (left) and Binary Image (Right)

Then according to 0 or 1, the image will be binarized while 0 equals white and 1 equals black. The purpose of making the image black and white is to prepare the image for the OCR process. It will increase the accuracy rate of the OCR process because binarization simplified the input data.

2.1.2 Pattern Matching Optical Character Recognition

Originally Optical Character Recognition (OCR) is a method used to detect and “read” a printed or in some case a handwritten text by recognizing each character inside an image of physical documents scanned and form a document based on the physical document.

The process of OCR is segmenting each character of a document and translating the characters into a digital code through pattern matching that can be processed using a computer, usually used to convert a hardcopy into PDF files in a computer. After segmentation is made, a character is converted into its binary matrices, white pixels to zero, black pixel to one. The image pre-processing is necessary.

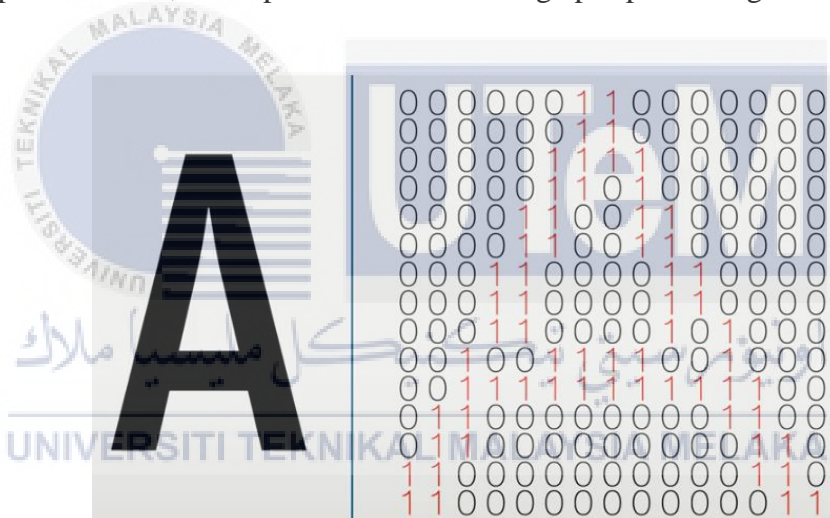


Figure 2.2 Binary matrix of a segmented character

A circle of calculating the radius from origin to furthest point is created as shown in the figure below. The black part of an alphabet A is labeled as 1, otherwise 0. The contrast will determine whether the area is 1 or 0.

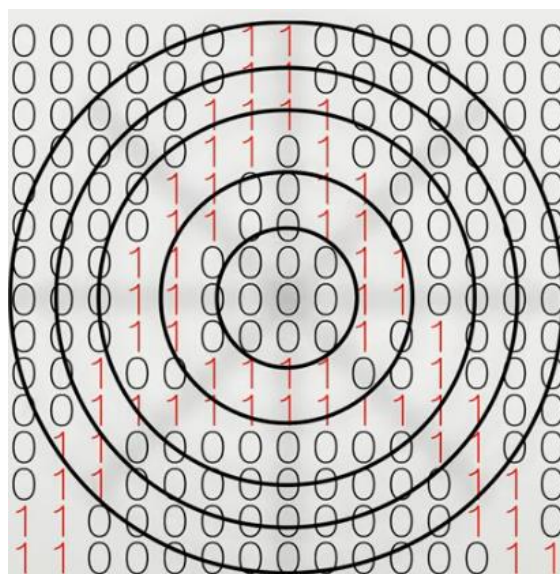


Figure 2.3 Pattern matching algorithm on image

At this point, the algorithm will compare every section of the matrices' image from the origin to the furthest radius with trained samples to find which character has statistically the most common with the image. This is called pattern matching. Figure below shows the binarized result and labeling from OCR in MATLAB.

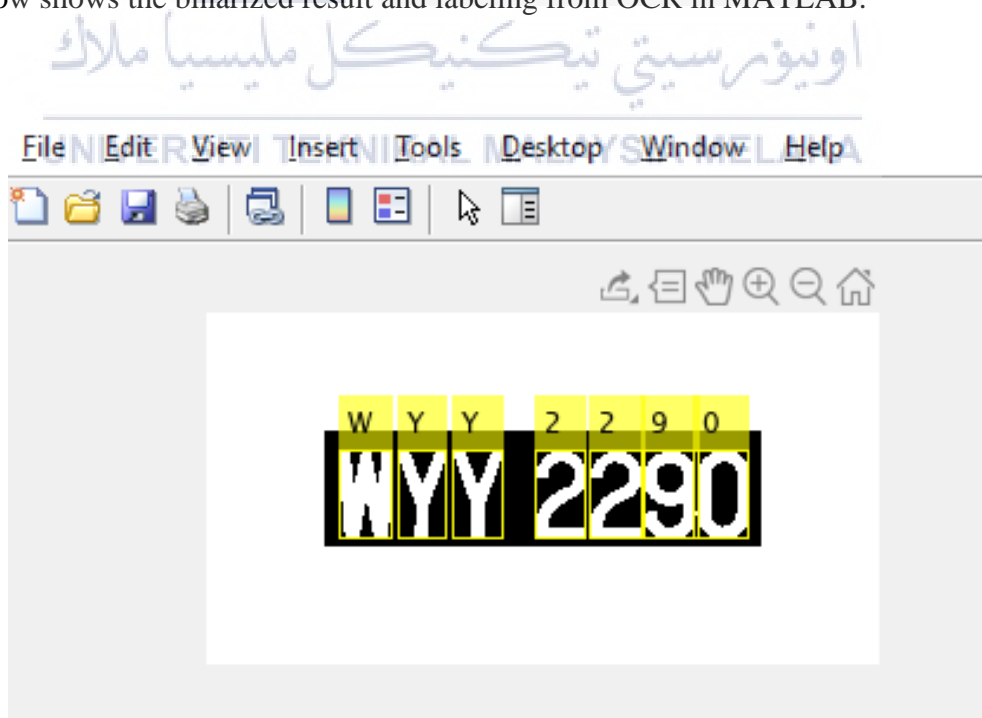


Figure 2.4 The binarized image and detected character on MATLAB

2.1.3 Convolutional Neural Network (CNN)

CNN is a class of deep neural networks. CNN's are comprised of neurons or multi-layered neurons that self-optimize through learning. Every neuron will take in input then operate to perform self-learning. The neurons from each layer are fully connected to all the neurons in the next layer.

CNN's perform deep learning by comparing images pixel by pixel by looking into the images' digital data. The similar pieces that CNN's will be looking for are called features. By looking forward to rough and similar feature matching in two images, CNN's get a lot better at finding tiny similarities than finding a match looking at a whole image. Whole-image matching schemes are naïve and have low accuracy because the decision is too quick and simple.

2.1.4 Comparison Between pattern matching and CNN

According to the article [2], the disadvantages of Convolutional Neural Network (CNN) is it is expensive to be implemented, especially if a complex task is assigned, a good Graphic Processing Unit (GPU) is required. Other than that, CNN requires a lot of samples for training because it needs multiple passes through. On the other hand, pattern matching by itself can recognize the car plate well by providing a low amount of training samples, since the fonts of car plate characters will not differ much due to traffic law enforcement of Malaysia.

Based on research by Kapadia [4], template matching method OCR is straight forward and reliable. However, a small difference in font or tilt will affect the recognition process. Therefore, to improve the accuracy, the segmentation process has to include reducing tilt-ness but since Malaysia car plate's fonts do not differ much, a lesser training template is needed. Based on research by Ravendra [6], the simulation

results using OCR in MATLAB, which is the traditional template matching too, also shows that the system could accurately detect and recognize the car plate of vehicle under different light luminance as long as the training templates are clear enough with less noise. It can be set up to be implemented on the guard house or guard house of any restricted area. [6]

As for neural network research conducted by Nagare[5], it is found to be more accurate and at the same time relatively more tolerant of more fonts of character and tilt. However, it requires high investment to do that on your own such as high GPU, big amount of training images, high storage. Therefore, a high-performance based hardware is required. Based on Kocer's research [7], two separate ANN's were used for classification, which is for letters and numbers. The purpose is to increase the correct detection rate of the recognition session of the car plate character. Some words and letters have similar features. Using two separate ANN can prevent the mix up of recognition, for example, "0" and "O", "2" and "Z", "8" and "B".

2.2 Internet of Things (IoT) in the Surveillance System

For Application Program Interface (API) to build the application, Node.js is used for coding environment and runs on JavaScript language. This API runs on JavaScript language. JavaScript is invented in 1995 at Netscape Corporation. It is also known as LiveScript.[14] Although it has Java keyword in JavaScript, it has nothing to do with Java language. This language could build programs into the user's web browser. They can be written in any webpage's HTML and update on its own as soon as the page reloads.

2.2.1 Application Program Interface (API)

An application program interface is a set of routines, protocols, and tools for building software applications and it determines how a programmer interacts with software components. In this project, Node.js, a free open-sourced, cross-platform JavaScript run-time environment.[15] This API allows developers to write instruction code outside of a browser or webpage.

2.2.2 Communication Protocol

A communication protocol is a set of rules or systems that enable two or more entities to communicate by transferring information. The protocols should be download and included in the entity's source code before using them after referring to their documentation. Therefore, agreement upon parties involved is inevitable.

In this project, while for transferring messages between machines, MQTT, an “Internet of Things” connectivity protocol is used. MQTT was invented by Dr. Andy Stanford-Clark of IBM, and Arlen Nipper of Arcom in 1999 and is well implemented in many industries around the globe.[16] This messaging protocol can do lightweight message transferring very well. The advantage of using MQTT is it consumes less power and has high efficiency of passing information to one or more receivers.[16]

Visual Studio Code is used to edit source code here. It is developed by Microsoft company. This program supports debugging, intelligent code suggesting, preferences, terminal running, and many other useful features. In the Stack Overflow 2019 Developer Survey [17], this program has been rated the most popular developer environment tool. More than half of an approximate of 87,000 respondents claimed to be using this program to edit source code.

2.2.3 Comparison between Node.js and PHP

Based on Lei's research [8], Node.js performs much faster than traditional PHP in handling large requests. Node.js is also an emerging technology and has many advantages in the front end development, which is also a good API to build a dashboard while enhancing this project in the future.

Based on Bangare's research [11], the advantage of Node.js over traditional PHP is that all layers of the framework are written in JavaScript language only. Therefore, the program is always well fit and synchronized. He also mentioned that any database deployed on its own developer's system, in this project it refers to Influx Database deploying on Node.js, which has the advantage of a developer having full control over the privacy and security of it.

Other than that, Node.js can be used to access many kinds of databases such as NoSQL, MongoDB, and Firebase, said Bangare [11]. Therefore, Developers have more choice of database to select.

2.3 Database in the Surveillance System

The database acts as a place to save data or memory. It can generally store data and access it through a computer system. The database can be based on either a local or a cloud server. A cloud server is remote, which you can access whenever you have internet, mostly renting the space from the server owner. For the local server, the server is what you own physically or have on-site with you. For this project, a local server is used. The database used is Influx Database, which is time-based, whenever you save data into the database, the time stamp will be recorded in the database as well.

In this project, the database serves as the data storage of the vehicle owner's information. Whenever a vehicle's car plate is scanned and recognized, a request will be sent from MATLAB to the app to extract data from the database. The database is based in locally in this project.

According to Nasar [12], A time-series based database is specifically developed to manage or record changes over time. Therefore, this database could allow further improvement or enhancement of the surveillance system by doing vehicle data analysis, tracking system performance.

2.3.1 Comparison between InfluxDB and Firebase

Firestore is a cloud-hosted document store, while Influx Database is a time series-based database. Influx Database will provide a timestamp for any insert of data. Other than that, Influx Database supports SQL-like query language to store or access data while Firestore does not. The advantage of Influx Database is that it supports JavaScript with Node.js and 15 other famous programming languages while Firestore only supports 3 languages.[9][10] The table below shows the comparison of the Firestore and Influx database.

Table 2.1: Comparison of Firebase and Influx Database

Name	Firebase [9]	Influx Database [10]
Database type	Cloud-hosted Realtime document store.	Time-series based.
Structured Query Language (SQL)	No	SQL-like query language
Supported Language	Java, JavaScript, and Objective-C only	16 languages including JavaScript (Node.js), Java, PHP and Python

Furthermore, Balis' research[13] said that Influx database has an advantage of having a feature that could group data by time, allowing users to request data within a range, this allows the project to further enhance the surveillance system by monitoring and analyze vehicles that go through the camera. The number of vehicles could be analyzed by grouping them by week, month, or year to provide analytical service in the future.

CHAPTER 3

METHODOLOGY



3.1 Project Implementation Flowchart

To ensure that the process of developing the project can be done smoothly and systematically, a project implementation flowchart with seven parts as shown in Figure 3.1 was created. Firstly, for the background of this study, past research papers will be referred for the project. The technique and platform of the researches will be recorded

and compared to find the best method. Next, the hardware used in this project are a laptop of 8 gigabytes RAM and a mobile phone with a camera.

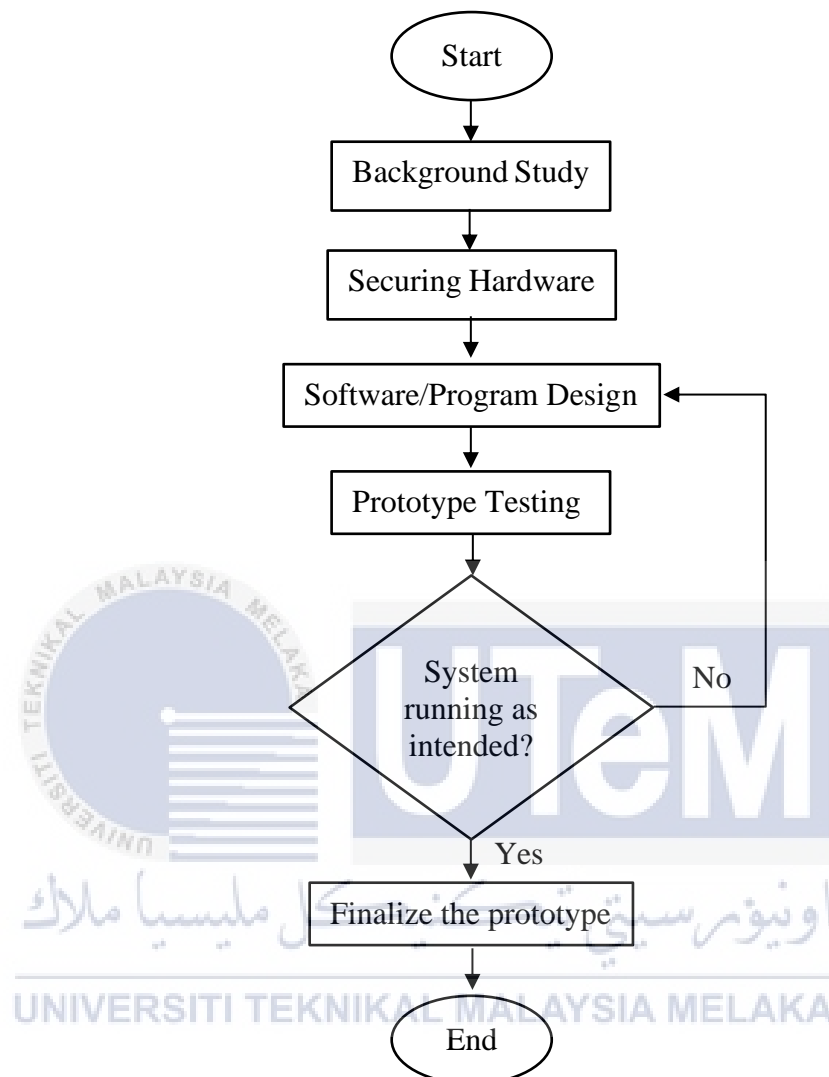


Figure 3.1 Project Implementation Flowchart

For the software and program design, the programming language that involves in this project is C, JavaScript, and query. C language will be used to operate and run the optical character recognition on MATLAB. After the phone camera is connected to a laptop, it will send a real-time recording to MATLAB via Bluetooth and to process through OCR. The result of the OCR will be saved into an excel file (xlsx format).

The crucial part of program designing is the app. It is designed using Node.js as API, and JavaScript as a programming language. The app extracts the data from the excel file.

Based on the block diagram as shown in Figure 3.2, a photo will be taken on a mobile phone camera, the phone will send the image of the vehicle passing by to a laptop to be processed by MATLAB.

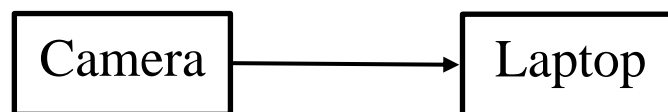


Figure 3.2: The block diagram of the hardware system

After the software and program design had been done, the prototype will be tested, and troubleshooting will be done if required. After confirming that the system running as intended, the prototype will be used to test the functionality and efficiency of the overall prototype.

3.2 Software Flowchart

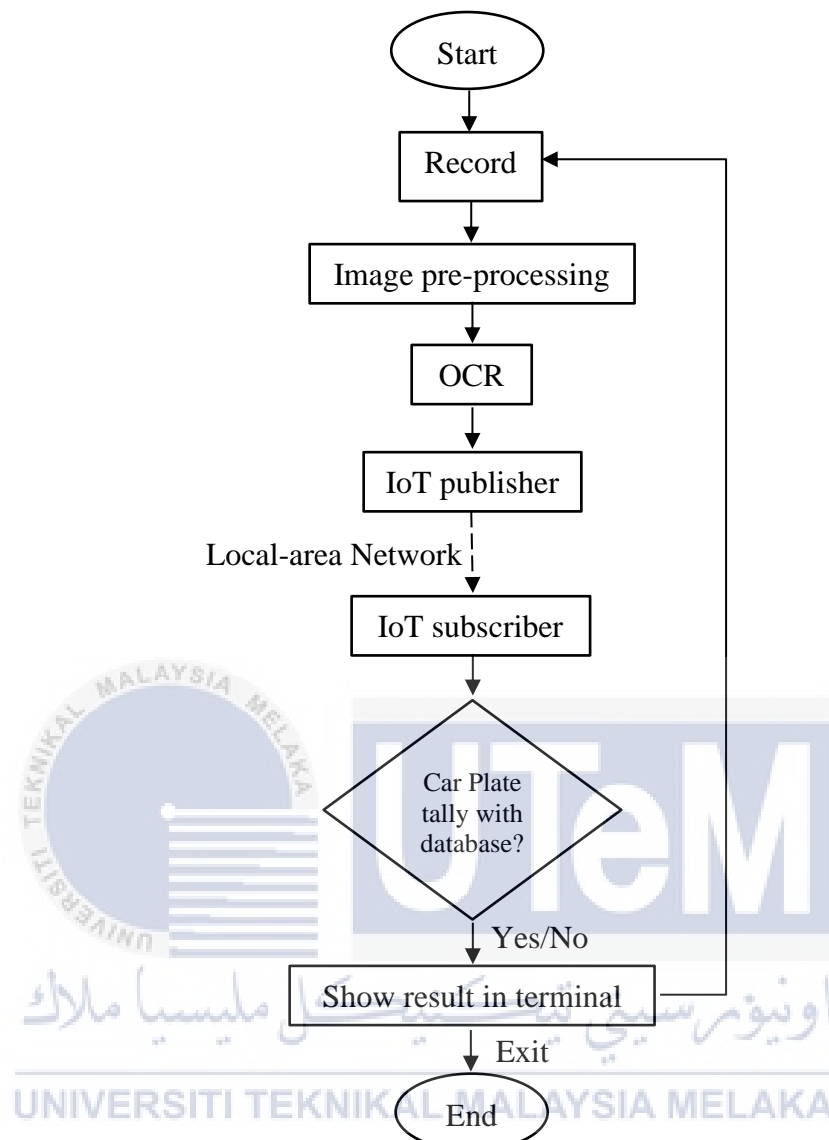


Figure 3.3: Software Flowchart

The software is categorized into 3 main parts. The image processing part, which involves MATLAB to do image treatment and OCR. This part is likely to enhance the recorded image so it can improve OCR performance.

The second part is the IoT part, where the app will extract information from MATLAB and Database. This part involves MQTT for publishing and subscribing purposes. This app can be separated into two working folders, by means, it can operate on two different computers. Note that this can only work if both of the computers are

connecting to the same local-area network (LAN). One is to extract information from MATLAB through an excel file, then publish them on the broker. Another one is to subscribe to the message and request information from the database according to the message.

The third part is the database part. Registered vehicle's information will have to be stored inside the database to enable future extraction. This can be done by using Influx query in influx.exe.

The methodology of doing the parts mentioned above will be mentioned in detail along with possible errors faced during the setup process in the following paragraph. The image below shows the overview of the project.

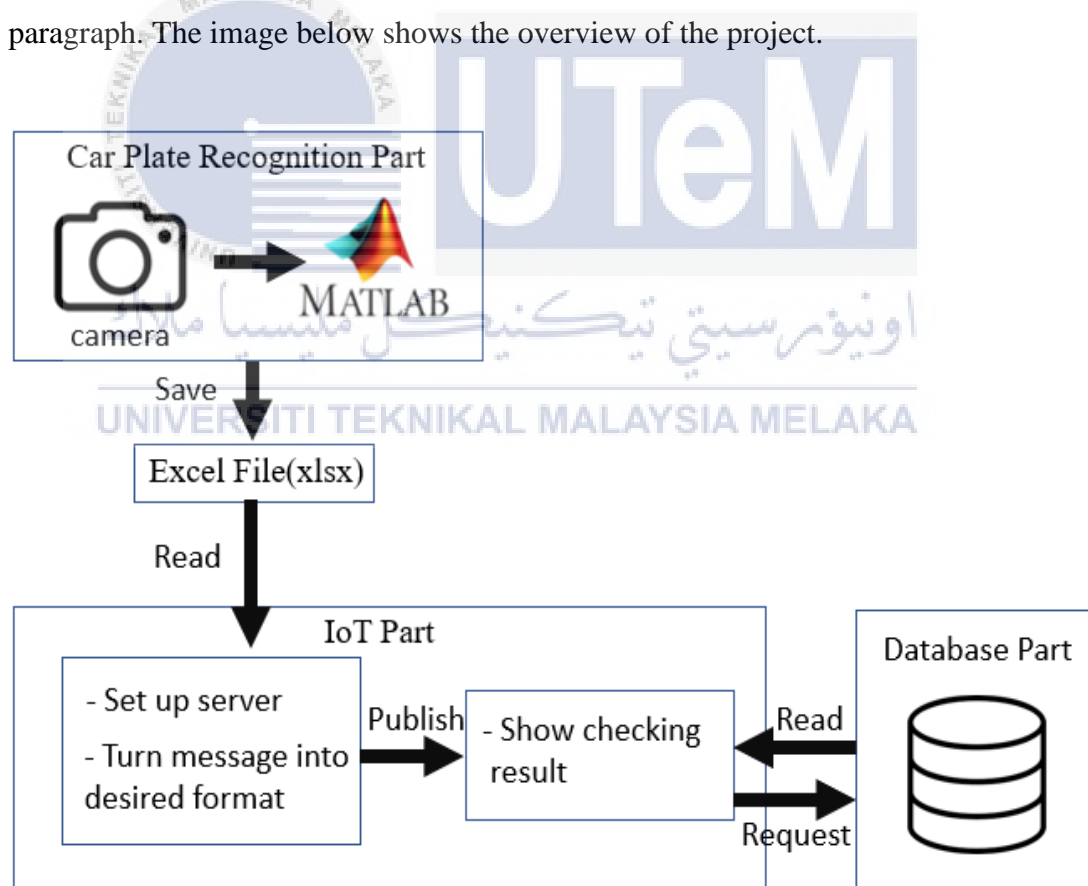


Figure 3.4: Overview of Project

3.3 Car Plate Recognition Part Methodology

A mobile phone camera is connected to a laptop via Bluetooth. Then, by using MATLAB the recording sent from the phone is snapped and undergoes image treatment before sending it to OCR.

The first step is converting the image from RGB to gray color using 'rgb2gray(I)' function. Then, extract the gray thresh information from the converted image using 'graythresh(I)', where the gray color is given value from 0 to 1. 0 stands for white and 1 stands for black. The next step is binarization, which is converting the gray image into a black and white image using '~imbinarize (I, threshold)' function. In this case, the '~' sign in front of the function stands for negative, which means according to grayscale from 0 to 1, 0 to 0.49 will become black while 0.5 to 1 will be turned into white. This process is to ease the OCR process by making the characters black and significant. Figures below show the comparison between the original image and the image that is ready to go for the OCR process.

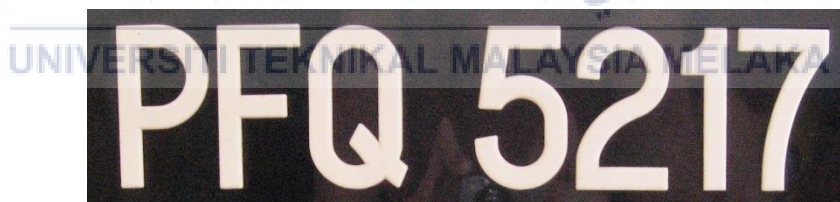


Figure 3.5: Original Image before Image Treatment

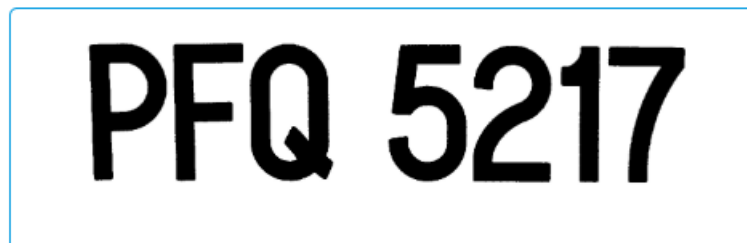


Figure 3.6: Output Image after Image Treatment

The image will then go through an OCR process by using the ‘ocr’ function in MATLAB. The result will be converted into text and saved in a Microsoft Excel file (xlsx format).

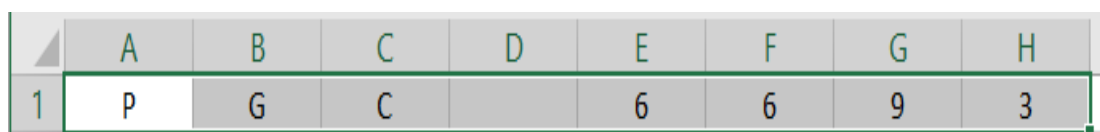
3.4 IoT Part Methodology

IoT part is the main part of this project as we are implementing this system on ALPR. So, message transferring and good communication between two or more entities are very crucial.

3.4.1 Excel Format

In this project, an approach to upload the result of OCR to Thing Speak IoT platform before but the platform itself can only allow transferring message that contains number only, which is not ideal for uploading car plate characters because car plate usually consists of the alphabet too. It is designed for data aggregation and analytic, therefore it is not suitable for this project. There are no current available IoT tools that can work with MATLAB to send a message through the internet, therefore another alternative is used, which is through xlsx format.

An app is created using Node.js to read the data from the excel file. This excel file will update every 3 seconds to ensure the data is up to date in case the MATLAB OCR recognized any character from the camera. This xlsx folder cannot be opened while the MATLAB is writing data. The image below shows the recorded result saved from MATLAB and stored in an excel file.



	A	B	C	D	E	F	G	H
1	P	G	C		6	6	9	3

Image 3.7: The recorded data in excel file

3.4.2 Including Protocols

To enable xlsx-node protocol, the documentation had to be downloaded by opening a terminal in the working folder. Then, including the functions or protocols in the source code, else the app cannot extract data from the excel folder because xlsx function will become an error. The API will show an error while debugging because this program does not recognize the function if the protocol or documentation is not included. The image below shows the result of running the app to extract data from excel without calling the function.



Image 3.8: Error: xlsx function is not defined

Therefore, to enable communication between Influx database, the app, and MATLAB, different functions had to be included in the source code after downloading them in command prompt. We also need to include ‘Mosca’ protocol for broker set up purpose, MQTT for publishing and subscribing messages, Influx-node for operating database and xlsx-node for extracting information from excel file to the app

3.4.3 Setting up a broker

A broker is set up before performing message transferring. Without the broker, MQTT alone will not work because there is no server to perform subscribing and publishing messages. In this project, the broker is set up on a localhost, which is on the laptop’s IP address, in this case, it is 192.168.0.162.

After a port is used to set up the broker on this laptop. A computer port act as an interface between computer and computer or computer with external devices. If a port is not available, it cannot be used to set up the broker. Therefore, to check a broker, one can open a terminal and type in 'ipconfig'. Afterward, type "netstat -a". A list of port numbers will come out along with their current state. The figure below shows the list of port numbers after executing the command on the laptop. Referring to the figure below, the port number will be shown at the end of the 'Local Address'. For example, the port in the first line is 135, following by 445, 5040, and so on.

```
C:\Users\Asus>netstat -a

Active Connections

Proto Local Address           Foreign Address         State
TCP   0.0.0.0:135             DESKTOP-HMC7EBG:0      LISTENING
TCP   0.0.0.0:445             DESKTOP-HMC7EBG:0      LISTENING
TCP   0.0.0.0:5040            DESKTOP-HMC7EBG:0      LISTENING
TCP   0.0.0.0:7680            DESKTOP-HMC7EBG:0      LISTENING
TCP   0.0.0.0:8086            DESKTOP-HMC7EBG:0      LISTENING
TCP   0.0.0.0:49664           DESKTOP-HMC7EBG:0      LISTENING
TCP   0.0.0.0:49665           DESKTOP-HMC7EBG:0      LISTENING
TCP   0.0.0.0:49666           DESKTOP-HMC7EBG:0      LISTENING
TCP   0.0.0.0:49667           DESKTOP-HMC7EBG:0      LISTENING
TCP   0.0.0.0:49668           DESKTOP-HMC7EBG:0      LISTENING
TCP   0.0.0.0:49669           DESKTOP-HMC7EBG:0      LISTENING
TCP   0.0.0.0:55852           DESKTOP-HMC7EBG:0      LISTENING
TCP   0.0.0.0:55853           DESKTOP-HMC7EBG:0      LISTENING
TCP   127.0.0.1:5354          DESKTOP-HMC7EBG:0      LISTENING
TCP   127.0.0.1:8088          DESKTOP-HMC7EBG:0      LISTENING
TCP   127.0.0.1:27275         DESKTOP-HMC7EBG:0      LISTENING
TCP   127.0.0.1:49670         DESKTOP-HMC7EBG:0      LISTENING
TCP   192.168.0.162:139       DESKTOP-HMC7EBG:0      LISTENING
TCP   192.168.0.162:55068     104.18.25.243:http     CLOSE_WAIT
```

Figure 3.9: List of Port Connection

According to the MQTT official website [16], the broker can only be set up on 5 ports, 1883 (unencrypted), 8883 (encrypted), 8884 (encrypted but client certificate required), 8080 (on WebSockets, unencrypted), 8081 (on WebSockets, encrypted). In this case, the 'Mosca' server was set up on port 1883.

After the broker is set up, the application will show “ready” in the terminal to verify that the broker is ready for subscribing and publishing tasks. Figure below shows the terminal message after a broker is set up and ready after running the program.

```
PS D:\hello> node app.js
ready
subscribed to alpr.js
```

Figure 3.10: Terminal Message to Show Broker is Ready

3.4.3.1 Error Setting Up Broker on Non-listening Port

If a port on a non-listening state, the broker could not be set up on this port. Else the application will show an error. Figure below shows error if a broker is set up on a non-listening port. In this case, the application could not detect port 8086 because it is not listening. The port is not available or already in use.

```
error: Error: connect ECONNREFUSED 127.0.0.1:8086
at TCPConnectWrap.afterConnect [as oncomplete]
  errno: 'ECONNREFUSED',
  code: 'ECONNREFUSED',
  syscall: 'connect',
  address: '127.0.0.1',
  port: 8086
```

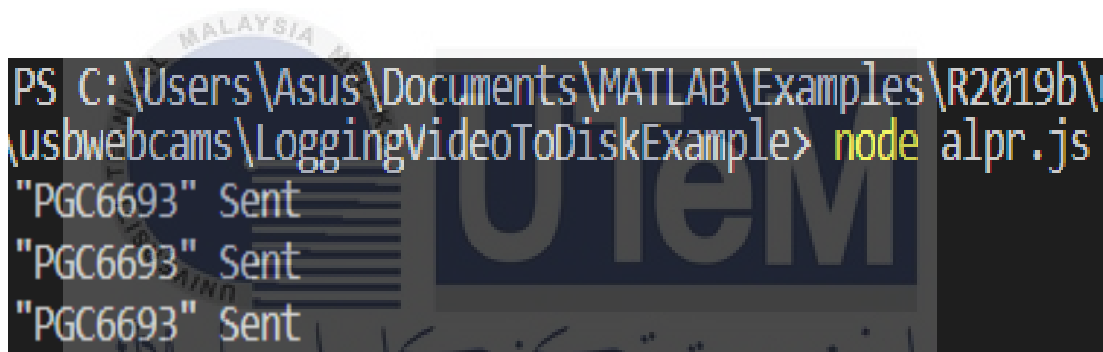
Figure 3.11: Error Setting Up Broker

3.4.4 Publishing Message - MQTT

After extracting OCR result from excel to the app, the app rearranged the characters from ['P','G','C',' ','6','6','9','3',' ',' '] into ['PGC6693']. The excel-folder has to be closed on any tab of the computer else it cannot be read to the app. The characters are combined into a single string. Spaces or unwanted signs are eliminated to match the format of car plate characters saved in the database. The string is then published

with a topic, in this project the topic name is 'myTopic'. A terminal message is set to inform the user the message is published. The purpose of showing the result in the terminal is to inform the user that the published message is sent, else the user would not be informed. The message will be directed to '{string_final}' in the code, as an object to make it dynamic. This will cause the result in terminal changes according to extracted data from excel.

To make sure the message published is correct, an instruction is written to show the published message on the terminal. Image below shows the result in the command prompt or terminal after the app is running in the working folder.



```
PS C:\Users\Asus\Documents\MATLAB\Examples\R2019b\usbwebcams\LoggingVideoToDiskExample> node alpr.js
"PGC6693" Sent
"PGC6693" Sent
"PGC6693" Sent
```

Figure 3.12: Terminal Message after Publishing Topic

3.4.5 Subscribing Message – MQTT

This app will publish a message to the broker so on the other side, the app will subscribe to the message by directing through the topic. The subscribed message will then be used to extract information from Influx Database. When the topic is subscribed, a terminal message “subscribed to alpr.js”, will be shown to verify the subscription towards the topic. Please refer to Figure XX: Terminal Message to Show Broker is Ready.

In this case, the topic name is simply “myTopic”, but it had to be exactly the same as the published topic, considering spaces, symbols, and big or small letters.

After subscribing, the app receives a line of string. Note that only string or array could be sent through MQTT.

3.4.6 Correcting JSON format

The subscribed topic will be parsed into an object from string. The purpose of turning this message into a JSON object is to ease the use of Influx Database's query function inside the app. A string format of the message will be rejected. Image below shows the result shown in the terminal if the message is not in the right format to extract data from Influx database.

```
message: 'A 400 Bad Request error occurred: {"error": "error parsing query: found IDENT, expected ; at line 1, char 32"}\n'
```

Figure 3.13: Error while extracting Information from Influx Database Using String

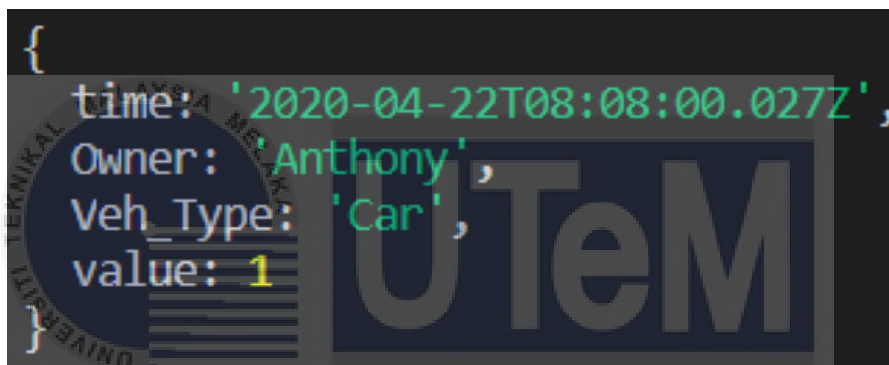
3.4.7 Extract Information From Database

After parsing the received message into a JSON object, the app selects and reads specific information from Influx Database. By doing this the extracted message is dynamic. Any changes at the subscribed message will change the requested information from the database.

Influx query protocols enable this app to access Influx database from this app internally by using Influx's query language. Note that Influx.exe has run in the background to enable the accessing of the database. The extracted information from the database is then changed into string so that it can be printed out on the terminal to be verified.

3.4.7.1 Managing Plain Result from Database

Influx database is a time-series database, in this case, the data insert time for the vehicle will be extracted along when a car plate information is requested. If the vehicle is recorded twice in the database, the output will show two messages. To avoid this, the app will rearrange them and only take in the first registered information. Image below shows the sole output if the information extracted is not managed. This information cannot be rearranged or used in other functions inside the app because this information is not changed into a JSON object.



```
{
  time: '2020-04-22T08:08:00.027Z',
  Owner: 'Anthony',
  Veh_Type: 'Car',
  value: 1
}
```

Figure 3.14: The Complete Output if extracted information is not Managed

3.4.7.2 Setting up a condition for vehicle check

If the car plate number and its information is not stored in database, the app will give an output of empty arrays on the terminal which is unclear for the user. To make the result clear while the app is checking for availability, a condition is programmed. If the database gives an empty array as feedback, then it means the vehicle is not registered, the terminal will show a line of the message, “Warning! The vehicle is not recorded in database.” Else the app will show registered vehicle’s information. In this case, the shown information is vehicle owner’s name and vehicle type. “console.log” function is to determine what to be shown inside the terminal or command prompt

area. Image below shows the source code of setting a condition to differentiate if a vehicle is registered inside the database.

The source code is crucial to make the terminal output clear. In this project, it will select important information only and rearrange them into clear and simple sentences to be shown in the terminal. The coding without parsing the extracted data and rearranging will give output in a relatively more complicated format. Image below shows the result on the terminal.

```

OUTPUT  TERMINAL  DEBUG CONSOLE  PROBLEMS
The owner is Anthony.
The vehicle is registered as Car for vehicle type.

```

Figure 3.15: Desired Result from Influx Query

3.5 Database Part Methodology

The database is set up to store vehicle information for this project. A database could be based on local or cloud. In this project, the database is based on the laptop, which is local.

3.5.1 Database Set Up

The way to write data into the database is through “influx.exe” application. The program is created while installing Influx database on the computer. Image below shows how the folder looks like. The first application is the influx.exe.

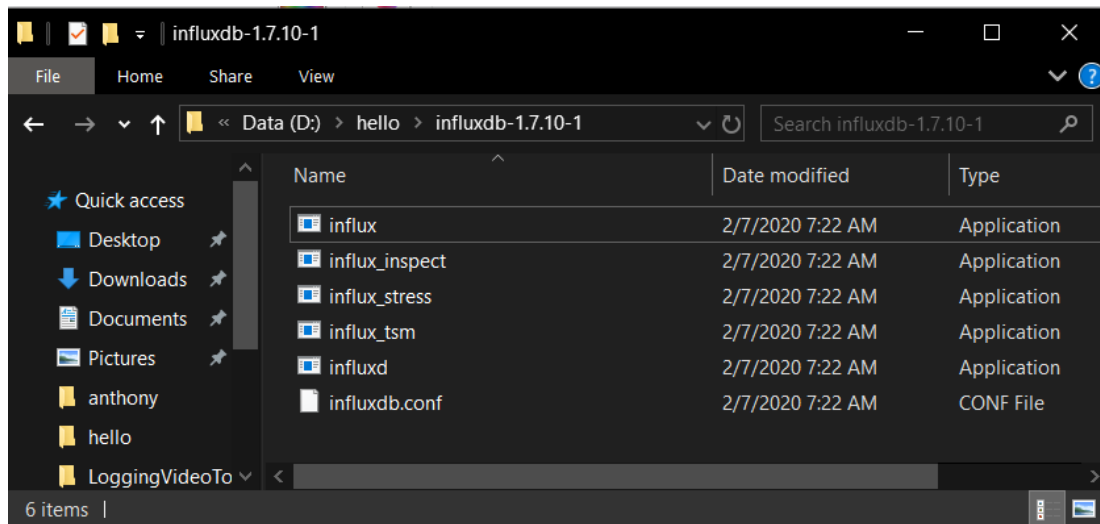
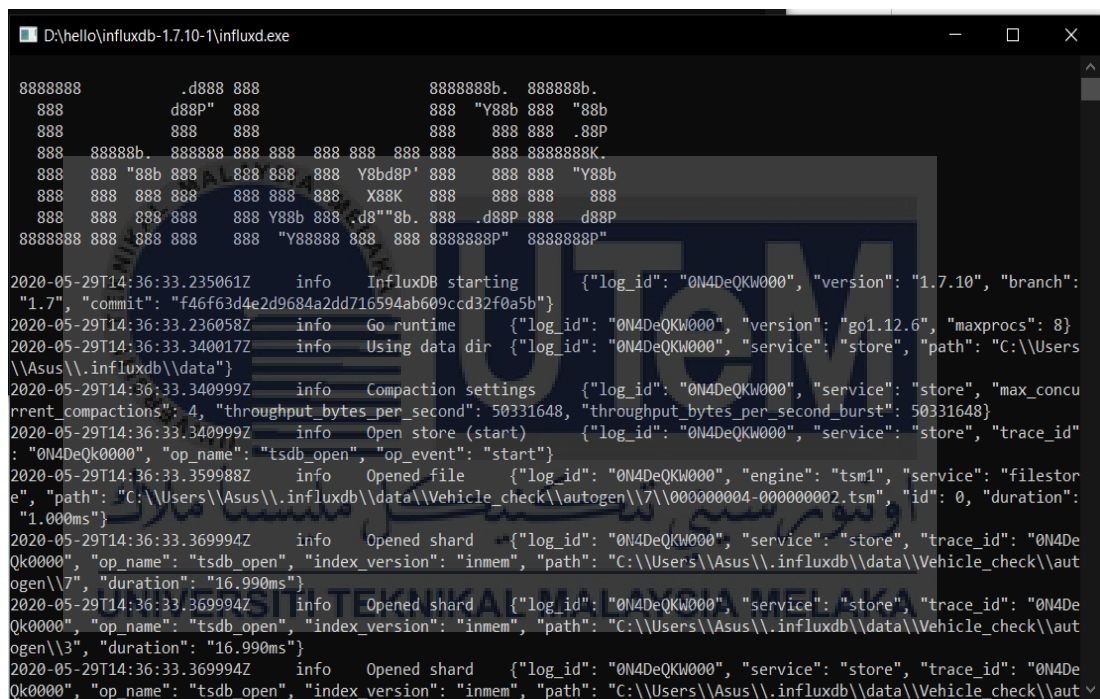


Figure 3.16: Influx database folder

Another alternative to access Influx database through Node.js. The 'influx-node' protocol could be installed. It allows users to write data through 'influx.query' function or set up a database through running a source code instead of giving sets of instruction using the influx-query protocol.

This alternative is not convenient because every time a user wanted to insert data, value, and information need to be changed here and re-run the source code again. It is not user friendly for an administrator who does not know much about the coding.

Other than that, before accessing through “influx.exe”. The database needs to be running in the background. In this case, it is “influxd.exe”. Afterward, instructions are executed through the Influx-query language inside of “influx.exe”. Without running the database in the background, or the influxd.exe, “influx.exe” will crash or close the window on its own. This program has to be running in the background when Node.js is running, else the database will not be accessed. Image below shows how influxd.exe looks like while running.



```

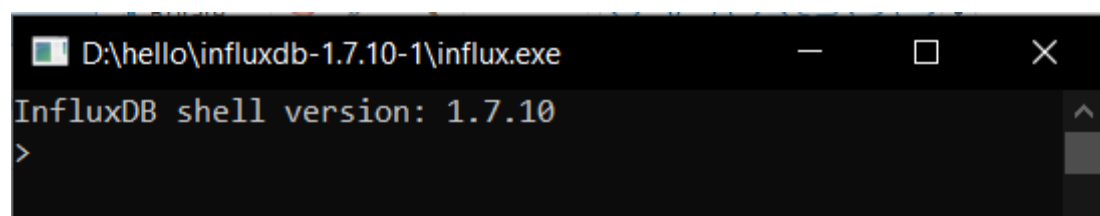
8888888      .d888 888                8888888b. 888888b.
888          d88P" 888                888  "Y88b 888  "88b
888          888  888                888  888 888  .88P
888 888888b. 888888 888 888 888 888 888 888 888 88888888K.
888 888 "88b 888 888 888 888 Y8bd8P' 888 888 888 "Y88b
888 888 888 888 888 888 888 X88K 888 888 888 888
888 888 888 888 888 Y88b 888 .d8"8b. 888 .d88P 888 d88P
88888888 888 888 888 888 "Y88888 888 888 88888888P" 88888888P"

2020-05-29T14:36:33.235061Z info InfluxDB starting {"log_id": "0N4DeQKW000", "version": "1.7.10", "branch": "1.7", "commit": "f46f63d4e2d9684a2dd716594ab609ccd32f0a5b"}
2020-05-29T14:36:33.236058Z info Go runtime {"log_id": "0N4DeQKW000", "version": "go1.12.6", "maxprocs": 8}
2020-05-29T14:36:33.340017Z info Using data dir {"log_id": "0N4DeQKW000", "service": "store", "path": "C:\\Users\\Asus\\AppData\\Local\\influxdb\\data"}
2020-05-29T14:36:33.340999Z info Compaction settings {"log_id": "0N4DeQKW000", "service": "store", "max_concurrent_compactions": 4, "throughput_bytes_per_second": 50331648, "throughput_bytes_per_second_burst": 50331648}
2020-05-29T14:36:33.340999Z info Open store (start) {"log_id": "0N4DeQKW000", "service": "store", "trace_id": "0N4DeQk0000", "op_name": "tsdb_open", "op_event": "start"}
2020-05-29T14:36:33.359988Z info Opened file {"log_id": "0N4DeQKW000", "engine": "tsml", "service": "filestore", "path": "C:\\Users\\Asus\\AppData\\Local\\influxdb\\data\\Vehicle_check\\autogen\\7\\000000004-000000002.tsm", "id": 0, "duration": "1.000ms"}
2020-05-29T14:36:33.369994Z info Opened shard {"log_id": "0N4DeQKW000", "service": "store", "trace_id": "0N4DeQk0000", "op_name": "tsdb_open", "index_version": "inmem", "path": "C:\\Users\\Asus\\AppData\\Local\\influxdb\\data\\Vehicle_check\\autogen\\7", "duration": "16.990ms"}
2020-05-29T14:36:33.369994Z info Opened shard {"log_id": "0N4DeQKW000", "service": "store", "trace_id": "0N4DeQk0000", "op_name": "tsdb_open", "index_version": "inmem", "path": "C:\\Users\\Asus\\AppData\\Local\\influxdb\\data\\Vehicle_check\\autogen\\3", "duration": "16.990ms"}
2020-05-29T14:36:33.369994Z info Opened shard {"log_id": "0N4DeQKW000", "service": "store", "trace_id": "0N4DeQk0000", "op_name": "tsdb_open", "index_version": "inmem", "path": "C:\\Users\\Asus\\AppData\\Local\\influxdb\\data\\Vehicle_check\\autogen\\3", "duration": "16.990ms"}

```

Figure 3.17: Running “influxd.exe” in Background

Meanwhile, after you open “influx.exe”, you can write or read from this database. Image below shows the first impression of the window after running influx.exe. The version of Influx Database is shown and ready to run instruction.



```

D:\hello\influxdb-1.7.10-1\influx.exe
InfluxDB shell version: 1.7.10
>

```

Figure 3.18: “influx.exe” first impression.

A database named “Vehicle_check” is built for this project. Writing any data will not work unless a database is chosen to access it. Image below shows the error if a data is inserted without setting a database.

```
> insert CBJ6238,Owner=YuGuan,Veh_Type=Car value=1
ERR: {"error":"database is required"}

Note: error may be due to not setting a database or retention policy.

Please set a database with the command "use <database>" or
INSERT INTO <database>.<retention-policy> <point>
```

Figure 3.19: Error: database is required.

3.5.2 Writing Information Into Database

After setting a database, data could be inserted into the database. In this project, important criteria to record while registering a vehicle is the owner’s name, vehicle type and, car plate number. Image below shows a sample data is inserted into the database.

```
> use Vehicle_check
Using database Vehicle_check
> insert CBJ6238,Owner=YuGuan,Veh_Type=Car value=1
```

Figure 3.20: Inserting Data into database

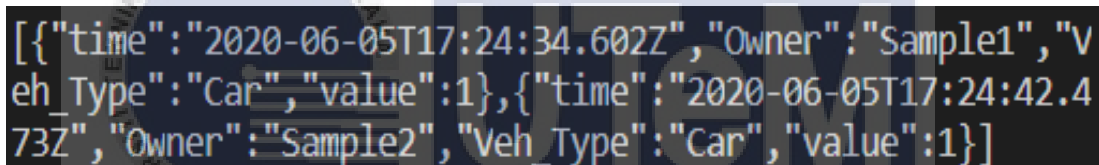
Note that there will be no message or notification to notify that this line of query is executed. The data will be written into the database after executed.

3.5.2.1 Duplicated Messages

If the same query is executed mistakenly again, the record will duplicate, which means there will be two similar information inside the database. Or in the other hand, the same vehicle information is registered more than once or updated again. In this case,

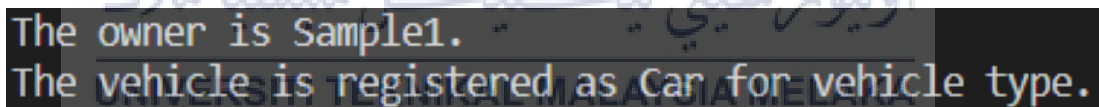
whenever the vehicle's information is requested, the original result feedback given by the database is duplicated too.

To resolve this issue, feedback from the database had to be re-arranged again. This again pinpointed the importance of re-arranging database feedback results. This means if two or more messages are given through database feedback, the program only takes in the first feedback message to be withdrawn and show in the terminal as the desired message. Refer to Figure 3.15: Desired Result from Influx Query. Figure below shows the terminal message when there is duplicated information of a sample vehicle and the next shows how the app re-arrange the message and only shows the first one as output.



```
[{"time": "2020-06-05T17:24:34.602Z", "Owner": "Sample1", "Veh_Type": "Car", "value": 1}, {"time": "2020-06-05T17:24:42.473Z", "Owner": "Sample2", "Veh_Type": "Car", "value": 1}]
```

Figure 3.21: Duplicated feedback



```
The owner is Sample1.  
The vehicle is registered as Car for vehicle type.
```

Figure 3.22: Message Re-arranged

3.5.2.2 Influx Database Error Writing Data

The reason for setting value equals 1 in the end of the query is because Influx database does not allow inserting “null” or empty value for its parameter. Image below shows error inserting data if the value is empty or not mentioned. Refer to Figure 3.20: Inserting Data Into Database for the differences.

```
er=YuGuan,Veh_Type=Car': missing fields"}
ERR: {"error":"unable to parse 'CBJ6238,Owr
```

Figure 3.23: Error inserting data without value

3.5.2.3 Inserting Timestamp

There will be times that the system is not available or in maintenance. A specific timestamp needed to be inserted manually to make sure the registration time is accurate. Since Influx Database is a time-series based database, any information inserted will be given a current timestamp. If the desired timestamp is needed to be inserted manually, it must be separated from the field(s) by a space. Other than that, it must be in Unix time and are assumed to be in nanoseconds.[10] The figure below shows the sample query to insert a data with specified timestamp in Unix format. The timestamp is highlighted with a black rectangular box.

```
Connected to http://localhost:26131 version 1.3.5
InfluxDB shell version: 1.3.5
> insert log value=1 1504225728000123456
```

Figure 3.24: Sample Query To Insert Specified Timestamp

3.6 Program Installation

MATLAB Installation

1. Download the product to your computer, then locate and click the setup.exe.
2. Select 'Get Add-ons' from the 'Add-ons' drop-down menu from the MATLAB desktop. The Add-on files are in the "MathWorks Features" section.

Visual Studio Code Installation

1. Download the product to your computer, then locate and click the **setup.exe**.

Node.js and protocol Installation

1. Download Node.js from its official website, then locate and click the **setup.exe**. Afterward, open a command prompt then enter “node -v” to verify if Node.js is installed inside the working folder you wanted to do the project.
2. Open a command terminal on your working folder. Then type ‘npm install mosca –save’ and wait for the download to complete. Then type ‘npm install mqtt –save’ and wait for the download to complete.
3. Open a command terminal on your working folder. Then type ‘npm install –save influx’ and wait for the download to complete.
4. Open a command terminal on your working folder. The type ‘npm install -g json’ and wait for the download to complete.
5. Open a command terminal on your working folder. Then type ‘npm install exceljs’ and wait for the download to complete.

Microsoft Excel Installation

1. Login to you student email account on the device you wish to install Microsoft Office.
2. Navigate to Office 365 screen.
3. Click ‘install’ under the Install Office session. If requested to sign in after installation, use your student email to login instead of inserting a license key.

Influx Database Installation

1. Download the product to your computer
2. Locate and click the “setup.exe”. In the influx folder
3. Double click the “influxd.exe” to run the database
4. Double click the “influx.exe” to insert command.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Car Plate Recognition Part

In this project, OCR only process image but not recording, a snapshot function is used to snap a moment from the recording. Then the image snapped will go through image pre-processing before OCR to improve the result. Since this project does not include segmentation, the pre-processing is much simpler because the region of interest is focused while setting up a camera.

4.1.1 Desired Car Plate Result

The result of OCR is only accurate given that the alphabets and numbers are in a standard format. Sometimes the detection of character is correct but not in the correct order, this will also affect the result because the app created later only takes in one format. The image below shows an example of the desired format and an undesired format of an OCR result in MATLAB terminal.

```
ans =
      'PGC 6683'
```

Figure 4.0: Desired OCR output

```
ans =
      ' PGC 6683'
```

Figure 4.1: Undesired OCR output

The difference between the two of the OCR output sample shown is that the undesired OCR output has too many unwanted spaces. It will write spaces while saving into excel file in the process.

Any result of OCR is turned into text and saved into an excel file, which is xlsx format. The results are written into the excel file from the slot 'A1' to 'K1'. The process will repeat itself after 3 seconds. A new OCR result will be updated into the excel file every 3 seconds. The purpose of updating the excel file every 3 seconds is to keep the data update time-to-time for checking purposes.

4.1.2 Simple Car Plate Recognition Test

Since the car plate recognition system in this project does not include segmentation. Manual cropping is used. Figure below shows an original car plate will its test result after OCR.

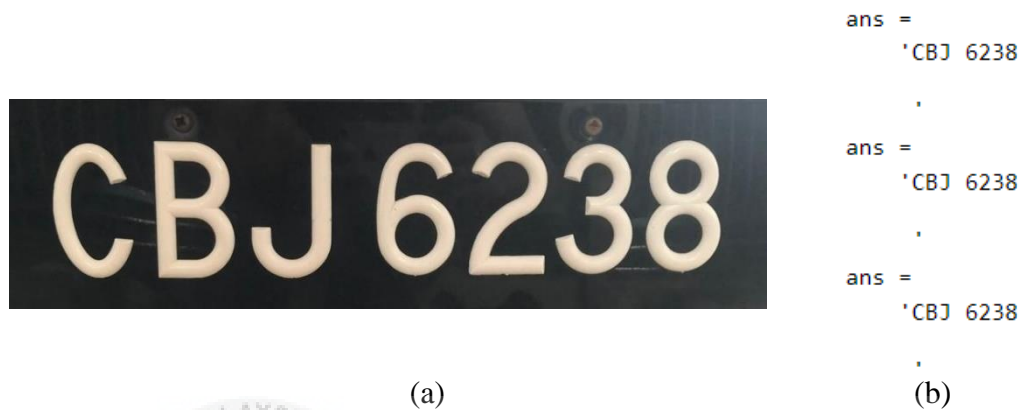


Figure 4.2: (a) Car Plate (b) OCR Test Result

This car plate image is taken by camera from a range of 20cm with good light illumination and the front of the car with an upright angle. Also, the car plate is in a single row. Therefore, the condition is ideal. The test results in (b) shows that the OCR operates and updates its result every 3 seconds. The results are consistent and accurate in this case.

4.1.3 Non-ideal Angle of Car Plate Recognition

A good system should be able to operate well in any kind of condition. For the car plate recognition part, the accuracy of OCR is a very important criteria in ALPR. If this system were to be put to use, the accuracy of this car plate recognition part has to be optimized. Yet, this is not an easy task. There are various non-ideal conditions to be considered while running ALPR such as light illumination, different car plate fonts, rainy day, mud or dirt covering part of the car plate and angle of the camera.

Therefore, a small test is carried out to test the capability of this OCR. The OCR is used to recognize a tilted car plate and to recognize a car plate from a 45° side angle.

4.1.3.1 Tilted Car Plate Image

To test the accuracy of the car plate recognition system in a non-ideal situation. An image of a car plate with 15° tilt angles is tested. The image is also taken by the same camera from a range of 20cm with good light illumination and from the front of the car plate. Figure below shows the original image and its test result.



Figure 4.3: (a) Tilted Car Plate (b) OCR test result

The result is that the OCR could not detect any of the characters on the image. Although this project is not focusing on image pre-processing, this could be improved in the future to enhance the system.

4.1.3.2 Side Image of Car Plate Recognition

To test the accuracy of car plate recognition in another non-ideal condition. The camera is placed from the side with an angle of 45° from the right front. Figure below shows the illustration of the condition from a top view.

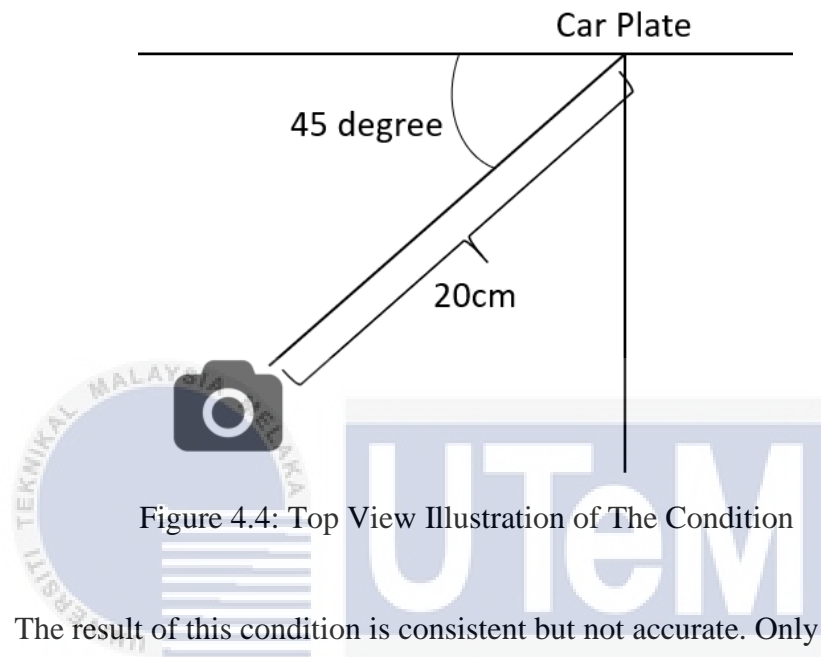


Figure 4.4: Top View Illustration of The Condition

The result of this condition is consistent but not accurate. Only 5 characters out of 7 are recognized correctly. The result is counted a fail if any of the characters are not recognized. More pre-processing and model training needed to overcome this issue. Figure below shows the car plate image from the side with its result.



Figure 4.5: (a) Side Image of Car Plate (b) OCR Result

4.2 App Performance Test

This part tests the performance of the app by testing the executing time taken for each cycle of the app. This test is carried out on 3 parts. The IoT part itself is divided into 2 parts while the last one is the OCR part. According to Node.js's documentation [15], two functions of 'console.time()' and 'console.timeEnd()' can be used in a JavaScript code to set timer in order to test executing time, it is called unit testing. This test is important because in most cases, developers will do the unit tests before adding new code to make sure the coding meets a defined requirement. If the unit test does not pass, then no new coding shall be added. This method is called test-driven development (TDD).

A total of 10 sets of 40 sample data of unit tests were taken for each part of the app. A set for the app to run initially, a set after 7 days of continuous running, a set after 14 days, and a set after 7 days up to 63 days (approximately 2 months). Then an average of result is calculated to plot graph. This is to test the consistency of the performance of the application. The purpose of the test is to see if the app has any network congestion that might cause decreasing in performance through time.

IoT part has 2 parts. The first part includes a broker set up, subscriber, and database communication while the other part manages data extracted from excel and publish them into the broker. These 2 parts could be running from 2 different devices as long as they are connecting to the same LAN, but in this research, both were running in the same laptop.

For the OCR part, it is programmed to run on one demo image repeatedly for 7 days. The OCR cycle is programmed to wait for 3 seconds after one loop of

recognition to ease the burden on the computer. A loop is ranged from the image processing part until saving OCR results into an excel file.



4.2.1 First Part App Test (Publisher, broker and, OCR communication)

Table 4.1 First IoT Part Test (Initial – 28 days)

Test No.	Run-Time per cycle (initially)	Run-Time per cycle (7 Days)	Run-Time per cycle (14 Days)	Run-Time per cycle (21 Days)	Run-Time per cycle (28 Days)
1	646.763ms	670.147ms	631.487ms	687.416ms	668.911ms
2	639.418ms	673.547ms	645.782ms	673.547ms	681.348ms
3	704.379ms	689.723ms	661.497ms	679.723ms	673.547ms
4	651.417ms	647.841ms	700.348ms	672.841ms	699.723ms
5	700.339ms	653.536ms	673.547ms	653.536ms	647.841ms
6	635.508ms	683.448ms	699.723ms	678.348ms	673.536ms
7	661.731ms	703.547ms	647.841ms	659.121ms	668.110ms
8	659.121ms	686.417ms	653.536ms	661.497ms	641.197ms
9	661.497ms	639.467ms	647.586ms	701.516ms	643.526ms
10	645.793ms	689.371ms	673.512ms	673.547ms	689.741ms
11	661.586ms	672.597ms	654.027ms	699.723ms	648.357ms
12	743.358ms	732.168ms	648.110ms	647.841ms	661.586ms
13	637.871ms	663.536ms	661.497ms	653.536ms	673.547ms
14	653.027ms	688.721ms	683.536ms	646.824ms	699.723ms
15	648.110ms	684.325ms	689.741ms	650.107ms	647.841ms
16	633.714ms	641.682ms	673.547ms	648.110ms	653.536ms
17	651.438ms	731.421ms	699.723ms	673.547ms	691.586ms
18	702.301ms	645.793ms	727.841ms	689.723ms	723.870ms
19	649.181ms	691.586ms	653.536ms	677.841ms	655.824ms
20	712.840ms	728.110ms	722.643ms	693.536ms	698.515ms

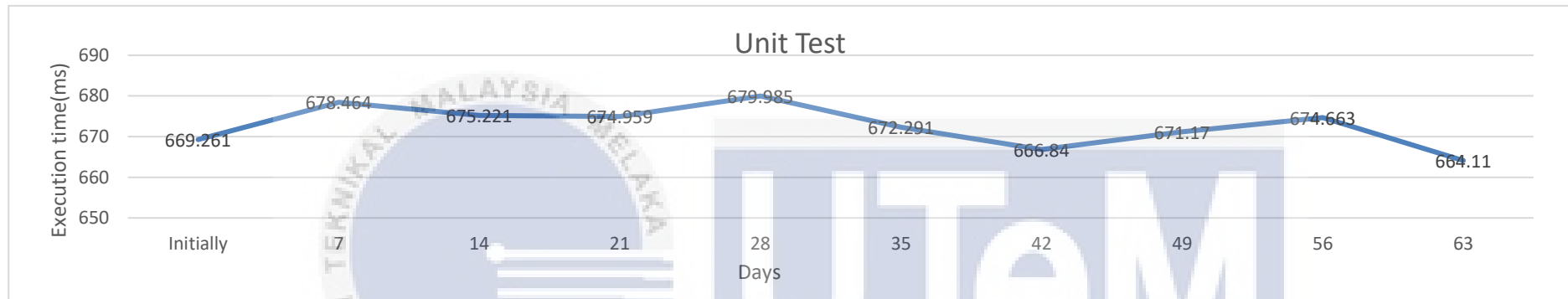
Test No.	Run-Time per cycle (initially)	Run-Time per cycle (7 Days)	Run-Time per cycle (14 Days)	Run-Time per cycle (21 Days)	Run-Time per cycle (28 Days)
21	669.734ms	657.824ms	648.468ms	720.318ms	659.367ms
22	720.114ms	669.714ms	676.984ms	655.824ms	721.235ms
23	639.376ms	625.834ms	653.027ms	703.870ms	659.121ms
24	680.455ms	693.356ms	678.110ms	701.642ms	661.497ms
25	673.547ms	679.534ms	688.715ms	659.121ms	703.870ms
26	699.723ms	647.527ms	653.536ms	661.497ms	644.815ms
27	647.841ms	723.315ms	689.741ms	680.455ms	648.975ms
28	653.536ms	644.121ms	698.375ms	673.547ms	661.586ms
29	689.741ms	632.457ms	703.648ms	693.685ms	655.639ms
30	698.375ms	713.820ms	680.455ms	659.121ms	680.455ms
31	701.356ms	704.132ms	647.841ms	661.497ms	673.547ms
32	655.824ms	655.824ms	723.870ms	693.651ms	679.548ms
33	674.545ms	703.870ms	684.815ms	635.922ms	710.495ms
34	640.453ms	644.815ms	647.841ms	680.455ms	659.121ms
35	687.232ms	713.870ms	655.824ms	643.572ms	661.497ms
36	703.870ms	647.841ms	710.226ms	723.841ms	703.874ms
37	644.815ms	708.870ms	669.121ms	674.815ms	679.355ms
38	658.312ms	687.738ms	661.497ms	653.536ms	663.532ms
39	684.192ms	692.671ms	655.824ms	689.741ms	675.824ms
40	647.988ms	660.437ms	703.870ms	668.375ms	698.186ms
Average	669.261ms	678.464ms	675.221ms	674.959ms	674.985ms

Table 4.2 First IoT Part Test (35 – 63 days)

Test No.	Run-Time per cycle (35 Days)	Run-Time per cycle (42 Days)	Run-Time per cycle (49 Days)	Run-Time per cycle (56 Days)	Run-Time per cycle (63 Days)
1	658.103ms	667.715ms	692.840ms	652.315ms	678.813ms
2	662.206ms	670.636ms	704.123ms	703.795ms	698.346ms
3	683.396ms	655.468ms	658.689ms	685.259ms	647.568ms
4	695.948ms	670.318ms	657.678ms	663.366ms	681.916ms
5	643.689ms	676.867ms	664.756ms	650.714ms	657.346ms
6	674.187ms	660.349ms	658.619ms	735.585ms	649.989ms
7	681.994ms	659.186ms	650.678ms	658.398ms	652.961ms
8	670.198ms	655.198ms	623.446ms	714.593ms	700.131ms
9	642.896ms	648.674ms	668.634ms	658.357ms	673.256ms
10	633.256ms	646.544ms	645.763ms	697.189ms	638.831ms
11	657.917ms	692.714ms	635.274ms	716.123ms	656.966ms
12	681.267ms	618.385ms	674.773ms	661.957ms	620.189ms
13	643.721ms	688.766ms	643.633ms	625.561ms	628.190ms
14	654.543ms	623.423ms	721.568ms	644.633ms	619.816ms
15	700.002ms	654.168ms	638.296ms	658.883ms	636.185ms
16	702.803ms	707.136ms	702.714ms	612.395ms	703.532ms
17	689.684ms	633.358ms	730.001ms	687.755ms	645.362ms
18	655.348ms	645.517ms	648.468ms	700.124ms	651.736ms
19	693.915ms	647.756ms	644.203ms	692.599ms	661.416ms
20	700.846ms	689.233ms	623.706ms	633.186ms	660.597ms

Test No.	Run-Time per cycle (35 Days)	Run-Time per cycle (42 Days)	Run-Time per cycle (49 Days)	Run-Time per cycle (56 Days)	Run-Time per cycle (63 Days)
21	638.831ms	671.925ms	634.545ms	680.209ms	689.224ms
22	656.966ms	661.497ms	641.124ms	634.040ms	656.317ms
23	699.818ms	655.824ms	661.416ms	659.172ms	611.818ms
24	683.132ms	714.593ms	660.597ms	644.330ms	661.416ms
25	692.599ms	658.357ms	656.966ms	681.994ms	660.597ms
26	633.186ms	697.189ms	661.416ms	670.198ms	656.966ms
27	661.416ms	681.994ms	660.597ms	642.896ms	661.497ms
28	660.597ms	670.198ms	681.818ms	692.599ms	655.824ms
29	651.736ms	642.896ms	714.593ms	633.186ms	645.362ms
30	702.714ms	702.050ms	658.357ms	624.494ms	651.736ms
31	638.296ms	661.416ms	697.189ms	684.693ms	637.622ms
32	661.497ms	660.597ms	702.714ms	661.416ms	714.593ms
33	655.824ms	651.736ms	730.001ms	660.597ms	658.357ms
34	730.001ms	652.396ms	681.994ms	702.714ms	697.189ms
35	647.062ms	737.504ms	670.198ms	730.001ms	623.708ms
36	685.181ms	661.497ms	692.896ms	659.822ms	661.416ms
37	626.713ms	655.824ms	673.654ms	714.593ms	660.597ms
38	714.593ms	645.362ms	681.497ms	658.357ms	638.296ms
39	658.357ms	651.736ms	675.824ms	697.189ms	702.714ms
40	697.189ms	643.587ms	623.548ms	689.231ms	730.001ms
Average	672.291ms	666.840ms	671.170ms	674.663ms	664.110ms

Graph 4.0 Line Graph of Unit Test for First IoT Part



Referring to the graph above, the graph does not show a uniform pattern. The system shows no constant dropping in performance through this period of approximately 2 months. From the first day of running the system until the 63rd days of continuous running, the execution time of the program differs within a small range of 11.624ms. The mean from these average execution times is 665.664ms which is 0.665 seconds. By dividing the differing range by the mean, the percentage different is 1.75%. According to Friansa's research [22], their system of extracting data from the IoT system to their battery monitoring system takes 1.04 ± 0.66 seconds. Despite different IoT features and data were implemented, this project has a relatively lower average execution time compared to the IoT system in their research.

4.2.2 Second Part App Test (Subscriber and Database Communication)

Table 4.3 First IoT Part Test (Initial – 28 days)

Test No.	Run-Time per cycle (initially)	Run-Time per cycle (7 Days)	Run-Time per cycle (14 Days)	Run-Time per cycle (21 Days)	Run-Time per cycle (28 Days)
1	314.170ms	291.369ms	324.837ms	294.241ms	319.661ms
2	300.461ms	321.604ms	322.334ms	300.514ms	311.078ms
3	318.483ms	297.720ms	301.501ms	296.989ms	301.851ms
4	321.181ms	301.464ms	318.483ms	303.917ms	310.849ms
5	299.671ms	294.715ms	321.181ms	308.122ms	305.910ms
6	295.822ms	293.698ms	299.671ms	294.663ms	325.153ms
7	316.519ms	313.546ms	296.933ms	308.939ms	308.721ms
8	300.885ms	295.319ms	324.837ms	299.614ms	316.472ms
9	308.151ms	303.144ms	317.720ms	293.660ms	324.625ms
10	305.466ms	321.661ms	313.628ms	318.483ms	293.663ms
11	303.819ms	299.793ms	301.151ms	321.181ms	309.842ms
12	308.987ms	290.362ms	315.164ms	299.671ms	302.608ms
13	305.273ms	318.483ms	301.540ms	290.463ms	302.799ms
14	327.223ms	321.181ms	329.711ms	289.650ms	304.650ms
15	316.961ms	299.671ms	322.334ms	311.783ms	315.195ms
16	293.724ms	301.177ms	318.393ms	302.346ms	292.020ms
17	303.467ms	307.859ms	303.542ms	316.624ms	318.483ms
18	304.831ms	323.104ms	326.307ms	328.493ms	321.181ms
19	301.796ms	290.727ms	312.993ms	300.651ms	299.671ms
20	310.124ms	293.534ms	308.295ms	299.187ms	313.604ms

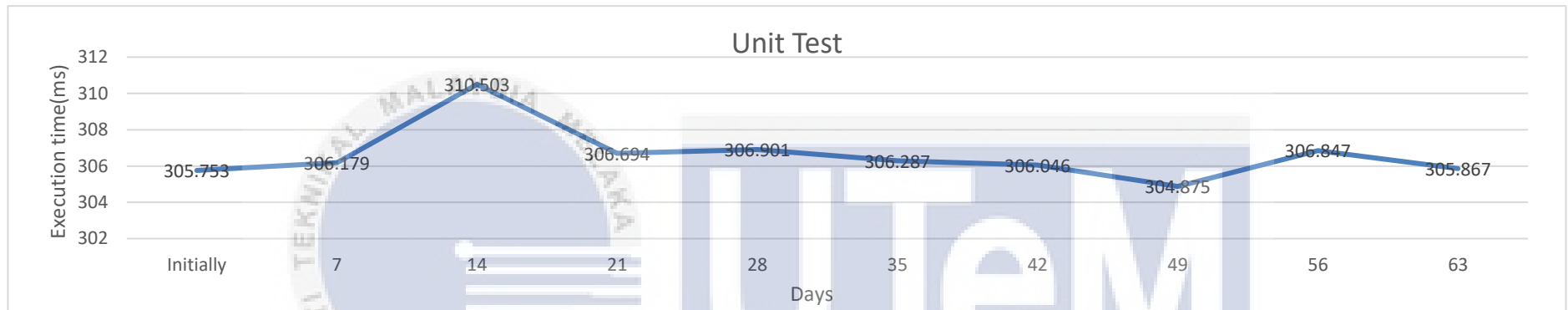
Test No.	Run-Time per cycle (initially)	Run-Time per cycle (7 Days)	Run-Time per cycle (14 Days)	Run-Time per cycle (21 Days)	Run-Time per cycle (28 Days)
21	299.871ms	321.661ms	277.618ms	284.958ms	269.184ms
22	302.187ms	299.793ms	318.483ms	290.463ms	290.463ms
23	320.233ms	290.727ms	312.993ms	289.650ms	289.650ms
24	317.810ms	290.727ms	299.671ms	311.783ms	311.783ms
25	306.191ms	293.534ms	299.671ms	312.993ms	300.651ms
26	296.624ms	322.334ms	311.783ms	315.195ms	321.181ms
27	301.447ms	316.961ms	299.671ms	326.307ms	299.671ms
28	306.283ms	289.677ms	299.793ms	292.993ms	318.483ms
29	317.123ms	316.624ms	318.483ms	328.493ms	321.181ms
30	304.881ms	328.493ms	311.181ms	291.783ms	315.195ms
31	289.174ms	290.727ms	293.727ms	312.993ms	300.651ms
32	300.243ms	293.534ms	308.939ms	308.721ms	290.727ms
33	311.423ms	321.181ms	299.614ms	316.472ms	308.939ms
34	306.061ms	299.671ms	296.624ms	318.483ms	299.614ms
35	283.414ms	322.334ms	328.393ms	321.181ms	293.534ms
36	306.487ms	308.939ms	308.721ms	326.307ms	278.433ms
37	304.831ms	289.614ms	316.472ms	312.993ms	308.721ms
38	300.729ms	317.203ms	301.181ms	269.614ms	316.472ms
39	310.124ms	326.307ms	308.939ms	308.721ms	299.793ms
40	289.431ms	312.993ms	299.614ms	306.472ms	287.659m
Average	305.790ms	306.179ms	310.503ms	306.694ms	306.901ms

Table 4.4 Second IoT Part Test (35 – 63 days)

Test No.	Run-Time per cycle (35 Days)	Run-Time per cycle (42 Days)	Run-Time per cycle (49 Days)	Run-Time per cycle (56 Days)	Run-Time per cycle (63 Days)
1	290.745ms	305.917ms	292.717ms	294.498ms	322.146ms
2	307.805ms	293.347ms	307.838ms	318.685ms	328.725ms
3	291.606ms	295.804ms	306.918ms	328.779ms	304.554ms
4	310.773ms	328.192ms	292.023ms	313.375ms	297.205ms
5	313.198ms	329.700ms	307.543ms	305.617ms	296.262ms
6	298.451ms	321.937ms	306.263ms	308.340ms	326.612ms
7	292.539ms	309.948ms	294.164ms	313.602ms	300.974ms
8	298.672ms	301.704ms	328.425ms	307.947ms	290.517ms
9	310.516ms	304.597ms	291.098ms	312.328ms	295.330ms
10	307.270ms	296.133ms	315.017ms	325.544ms	315.159ms
11	301.122ms	298.892ms	292.119ms	307.593ms	297.025ms
12	294.745ms	325.988ms	324.918ms	326.388ms	311.725ms
13	307.226ms	320.253ms	305.241ms	309.863ms	302.116ms
14	297.477ms	290.652ms	293.275ms	316.573ms	314.084ms
15	300.141ms	299.903ms	312.564ms	303.545ms	314.181ms
16	290.913ms	316.416ms	292.023ms	304.711ms	315.223ms
17	298.512ms	325.774ms	320.293ms	303.545ms	290.574ms
18	318.670ms	307.323ms	295.526ms	315.169ms	312.855ms
19	319.216ms	291.059ms	325.139ms	327.535ms	298.408ms
20	305.753ms	299.656ms	310.898ms	300.028ms	307.602ms

Test No.	Run-Time per cycle (35 Days)	Run-Time per cycle (42 Days)	Run-Time per cycle (49 Days)	Run-Time per cycle (56 Days)	Run-Time per cycle (63 Days)
21	298.892ms	314.474ms	287.732ms	283.101ms	288.754ms
22	325.988ms	306.552ms	293.646ms	293.337ms	314.204ms
23	320.253ms	306.994ms	283.847ms	291.846ms	303.463ms
24	290.652ms	292.483ms	294.329ms	300.687ms	290.277ms
25	307.602ms	307.643ms	312.616ms	297.715ms	314.298ms
26	277.345ms	282.444ms	312.545ms	284.446ms	305.599ms
27	307.925ms	287.853ms	312.306ms	314.729ms	292.794ms
28	324.781ms	307.133ms	290.937ms	288.139ms	283.513ms
29	293.545ms	281.373ms	295.279ms	282.991ms	299.579ms
30	305.166ms	301.738ms	290.475ms	294.068ms	285.177ms
31	327.538ms	294.997ms	312.359ms	313.735ms	306.788ms
32	300.718ms	299.332ms	299.023ms	296.299ms	290.546ms
33	303.565ms	293.170ms	305.966ms	303.302ms	285.297ms
34	304.711ms	289.802ms	285.006ms	297.026ms	293.345ms
35	315.908ms	307.133ms	308.649ms	314.367ms	283.614ms
36	302.453ms	288.783ms	308.411ms	292.369ms	307.049ms
37	298.716ms	302.986ms	289.773ms	284.221ms	317.281ms
38	309.308ms	313.815ms	296.741ms	303.216ms	288.674ms
39	301.443ms	296.429ms	300.432ms	301.398ms	305.527ms
40	309.631ms	319.498ms	302.933ms	281.221ms	311.609ms
Average	306.287ms	306.046ms	304.875ms	306.847ms	305.867ms

Graph 4.1 Line Graph of Unit Test for Second IoT Part



Referring to the graph above, the graph does not show a uniform pattern. The system shows no constant dropping in performance through this period of approximately 2 months. From the first day of running the system until the 63rd days of continuous running, the execution time of the program differs within a small range of 5.628ms. The mean from these average execution times is 306.599ms or 0.306 seconds. By dividing the differing range by the mean, the percentage difference is 1.84%. According to Friansa's research [22], their system of extracting data from the IoT system to their battery monitoring system takes 8.71 ± 12.12 seconds. Despite different database and data were used, this project has a relatively lower average execution time compared to the IoT system in their research.

CHAPTER 5

CONCLUSION AND FUTURE WORKS



5.1 Conclusion

For this thesis, we addressed the issue faced while setting up a simple IoT system that could work with car plate recognition. One of the main contributions of my work is to achieve the task with a very low cost. The developed prototype serves a purpose to do surveillance tasks less expensively, also the system does not drop in performance after working continuously over a great period.

To conclude, this system has achieved the objectives. It can recognize simple car plate character and turn an image into a digital data in computer and communicate with a database to verify a vehicle.

Other than that, this prototype is a simple cost-effective system that could save money, time, and increase efficiency. On top of that, the proposed API and software performed well in sending messages, extracting information from the database.

This work could be implemented and inherited in many other applications such as automatic mail sorting system, student/employee identification system, door and gate entry system, community automated parking management, traffic statistic, stock/inventory management system, internal plagiarism system, and highway automated tolling.

5.2 Future Work

One of the most crucial parts of this system is the car plate recognition system. Therefore, this part should be enhanced and improved to optimize accuracy. To increase accuracy, more image pre-processing should be done. Issues like light illumination, noises, segmentation error, recognizing on a rainy day, and different font of car plate character should be considered in the future. If we could overcome these objectives, this project could be improved.

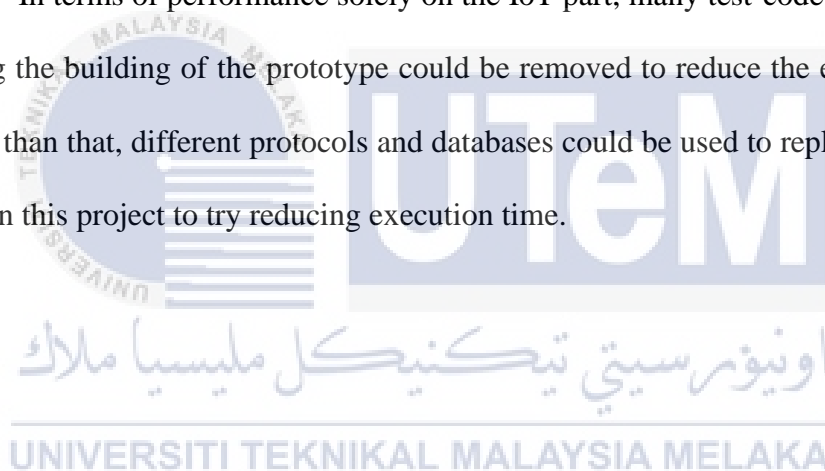
In terms of user friendly, a dashboard or webpage could be built to enable easy access to the system. The current system only shows result in a terminal, it will be difficult for people to understand how to start or run the system without coding knowledge. A clear and effective front-end dashboard could improve user experience. A dashboard could provide relevant information in order by just taking simple input from the user.

In terms of security, a password could be customized into the database and the excel-folder. This could improve the security of the system by preventing intruders

changing the car plate number from the excel file in the beginning or adding information on their own in the database.

As for the database, it could be set up to the cloud. A cloud database could release the administrative burden. If a database gets larger, it requires more space, this increases the demand towards the hardware. More frequent maintenance needed to sustain the database. Other than that, using a cloud database could improve security, because the company that provides cloud services who needs to work on the security system. This leaves fewer issue for the administrator to concern.

In terms of performance solely on the IoT part, many test-code that is inserted during the building of the prototype could be removed to reduce the execution time. Other than that, different protocols and databases could be used to replace the current ones in this project to try reducing execution time.



REFERENCES

- [1] Kavitha, H., Singh, M., Rai, S. K., & Biradar, S. (2017). Smart Suspect Vehicle Surveillance System. *Communication and Power Engineering*, 186.
- [2] Hosseini, H., Xiao, B., Jaiswal, M., & Poovendran, R. (2017, December). On the limitation of convolutional neural networks in recognizing negative images. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)* (pp. 352-358). IEEE.
- [3] Simin, N., & Mei, F. C. C. (2013). Automatic car-plate detection and recognition system. *EURECA*, 113-114.
- [4] Kapadia, P. S. (2011). Car license plate recognition using template matching algorithm.
- [5] Nagare, A. P. (2011). License plate character recognition system using neural network. *International Journal of Computer Applications*, 25(10), 36-39.
- [6] Ravendra Ratan Singh Rinky Sharma (2015) Automatic License Plate Recognition. Volume IV, Issue XI, November 2015. IJLTEMAS.
- [7] Kocer, H. E., & Cevik, K. K. (2011). Artificial neural network-based vehicle license plate recognition. *Procedia Computer Science*, 3, 1033-1037.

[8] Lei, K., Ma, Y., & Tan, Z. (2014, December). Performance comparison and evaluation of web development technologies in php, python, and node. js. In *2014 IEEE 17th international conference on computational science and engineering* (pp. 661-668). IEEE.

[9] Official website of Firebase Database. Website:

firebase.google.com/products/realtime-database

[10] Official website of Influx Database. Website:

www.influxdata.com/products/-influxdb-overview

[11] Bangare, S. L., Gupta, S., Dalal, M., & Inamdar, A. (2016, March). Using Node. Js to build high speed and scalable backend database server. In *Proc. NCPCI. Conf* (p. 19).

[12] Nasar, M., & Kausar, M. A. (2019). Suitability Of Influxdb Database For Iot Applications. *International Journal of Innovative Technology and Exploring Engineering*, 8(10), 1850-1857.

[13] Balis, B., Bubak, M., Harezlak, D., Nowakowski, P., Pawlik, M., & Wilk, B. (2017, January). Towards an operational database for real-time environmental monitoring and early warning systems. In *ICCS* (pp. 2250-2259).

[14] Official website of JavaScript. Website: <https://javascript.info/intro>

[15] Official website of Node.js. Website: <https://nodejs.dev/>

[16] Official website of MQTT. Website: <http://mqtt.org/>

[17] Developer Survey Result 2019. Website:

<https://insights.stackoverflow.com/survey/2019>

[18] Official website of MATLAB. Website:

<https://www.mathworks.com/discovery/what-is-matlab.html>

[19] Salman, N. H., & Hadi, G. M. (2013). Integrated image processing functions using MATLAB GUI. *Journal of advanced computer science and technology research*, 3(1), 31-38.

[20] Ivo, A. A., Guerra, E. M., Porto, S. M., Choma, J., & Quiles, M. G. (2018). An approach for applying Test-Driven Development (TDD) in the development of randomized algorithms. *Journal of Software Engineering Research and Development*, 6(1), 9.

[21] Khanam, Z., & Ahsan, M. N. (2017). Evaluating the effectiveness of test driven development: advantages and pitfalls. *Int. J. Appl. Eng. Res*, 12(18), 7705-7716.

[22] Friansa, K., Haq, I. N., Santi, B. M., Kurniadi, D., Leksono, E., & Yulianto, B. (2017). Development of battery monitoring system in smart microgrid based on internet of things (IoT). *Procedia engineering*, 170, 482-487.

APPENDICES

APPENDIX A

```
1 while true
2     %cam.Resolution = '640x480';
3     %cam = webcam;
4     %cam = 'e2eSoft iVCam';
5     %img = snapshot(cam);
6     img = imread('CBJ6238.jpeg');
7
8     I = rgb2gray(img);
9     threshold = graythresh(I);
10    BW= ~imbinarize(I,threshold);
11    imshow(BW)
12
13    results = ocr(BW);
14    results.Text
15    %array = isempty(results.Text);
16
17    filename = 'result.xlsx';
18    xlswrite(filename,results.Text,1,'A1:K1');
19
20    pause(3)
21 end
22
```

APPENDIX B

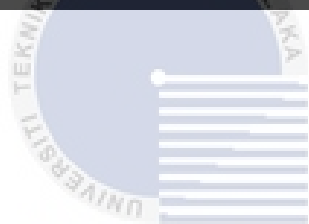
```
1 //console.time('test');
2
3 var mosca = require('mosca');
4 var mqtt = require('mqtt');
5 var client = mqtt.connect('mqtt://192.168.0.162');
6
7 const Influx = require('influx');
8 //creating database
9 const influx = new Influx.InfluxDB({
10   host: 'localhost',
11   database: 'Vehicle_check',
12   schema: [
13     {
14       measurement: 'ABC1234',
15       fields: { i:true },
16       tags: ['Owner', 'Veh_Type'] //parameters
17     }
18   ]
19 });
20
21 influx.getDatabaseNames().then(names =>{
22   if (names.indexOf('Vehicle_check') == -1 )
23     {return influx.createDatabase('Vehicle_check');}
24 }).catch(error => console.log({ error }));
25
26 //mqtt part
27 var settings = {
28   port:1883 //set up broker on local port 1883
29   }
30
31 var server = new mosca.Server(settings);
32
33 server.on('ready', function(){
34   console.log("ready");
35 });
36
37 //subscribe message from alpr.js
38 client.on('connect', function () {
39   client.subscribe('myTopic');
```

```
40 console.log("subscribed to alpr.js");
41 })
42 client.on('message', function (topic, message) {
43   context = message.toString();
44   New_message = JSON.parse(message);
45   //console.log(context);
46
47   influx.query(`select * from Vehicle_check..`${New_message}``)
48   .then(result => { var resultArray = JSON.stringify(result)
49     //console.log(resultArray);
50
51     var parsedData = JSON.parse(resultArray)
52     if (parsedData.length == 0)
53     {
54       console.log("Warning! The vehicle is not recorded in database.");
55     }
56     else
57     {
58       console.log("The owner is " + parsedData[0]["Owner"]+ ".");
59       console.log("The vehicle is registered as " + parsedData[0]["Veh_Type"]
60         + " for vehicle type. \r\n \r\n");
61     }
62   })
63   .catch(error => console.log(error));
64 })
65
66 //console.timeEnd('test');
67 اونیورسیتی تکنیکل ملیسیا ملاک
```


APPENDIX C

```
1 //console.time('test');
2
3 var xlsx = require('xlsx');
4 var mqtt = require('mqtt');
5 var client = mqtt.connect('mqtt://192.168.0.162');
6 client.on('connect', function () {
7   setInterval(function() {
8     var wb = xlsx.readFile('result.xlsx');
9     var first_sheet_name = wb.SheetNames[0];
10    var worksheet = wb.Sheets[first_sheet_name];
11
12    var address1 = 'A1' ;
13    var desired_cell1 = worksheet[address1];
14    var desired_value1 = (desired_cell1 ? desired_cell1.v : undefined);
15
16    var address2 = 'B1' ;
17    var desired_cell2 = worksheet[address2];
18    var desired_value2 = (desired_cell2 ? desired_cell2.v : undefined);
19
20    var address3 = 'C1' ;
21    var desired_cell3 = worksheet[address3];
22    var desired_value3 = (desired_cell3 ? desired_cell3.v : undefined);
23
24    var address5 = 'E1' ;
25    var desired_cell5 = worksheet[address5];
26    var desired_value5 = (desired_cell5 ? desired_cell5.v : undefined);
27
28    var address6 = 'F1' ;
29    var desired_cell6 = worksheet[address6];
30    var desired_value6 = (desired_cell6 ? desired_cell6.v : undefined);
31
32    var address7 = 'G1' ;
33    var desired_cell7 = worksheet[address7];
34    var desired_value7 = (desired_cell7 ? desired_cell7.v : undefined);
35
36    var address8 = 'H1' ;
37    var desired_cell8 = worksheet[address8];
38    var desired_value8 = (desired_cell8 ? desired_cell8.v : undefined);
39
```

```
27
28     var address6 = 'F1' ;
29     var desired_cell6 = worksheet[address6];
30     var desired_value6 = (desired_cell6 ? desired_cell6.v : undefined);
31
32     var address7 = 'G1' ;
33     var desired_cell7 = worksheet[address7];
34     var desired_value7 = (desired_cell7 ? desired_cell7.v : undefined);
35
36     var address8 = 'H1' ;
37     var desired_cell8 = worksheet[address8];
38     var desired_value8 = (desired_cell8 ? desired_cell8.v : undefined);
39
40     var string_final = (`${desired_value1}${desired_value2}${desired_value3}
41     ${desired_value5}${desired_value6}${desired_value7}${desired_value8}`);
42
43     client.publish('myTopic', string_final); //publisher
44     console.log(`${string_final} Sent`);
45
46
47     }, 8000);
48 });
49
50 //console.timeEnd('test');
51 |
```



UTeM

اونيورسيتي تيكنيكل مليسيا ملاك

UNIVERSITI TEKNIKAL MALAYSIA MELAKA