

**DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING**

**EEE311 FINAL YEAR PROJECT**

*Navigation Positioning Control of*

*Automated Guided Vehicles (AGVS)*

**Final Thesis**

In Partial Fulfillment

of the Requirements for the Degree of

Bachelor of Engineering

|  |  |  |
| --- | --- | --- |
| Student Name | : | Haocheng Zhao |
| Student ID | : | 1406522 |
| Supervisor | : | Dr. Limin Yu |
| Co-Supervisor | : | Dr. Fei Ma |
| Assessor | : | Dr. Rui Lin |

Abstract

The Automated Guided Vehicle (AGV) becomes more important in manufacturing industrial, along with the development of Artificial Intelligence (AI). The multi-AGVs system is the popular topic in this area, and various solutions have been proposed. An distributed multi-AGVs system can help to manage the vehicles, which leads to the high efficiency and non-collision working circumstance. In a distributed system, the most important coding part including movement, communication, and navigation, should be programmed in the Arduino car. In this project, Arduino car with the ESP8266 wireless module is constructed and a control software written in python is completed successfully. Infrared sensors are used to achieve the navigation including following the black line and stop function. The ESP8266 uses TCP to communicate with the center control panel, and uses UDP to communicate with other cars. To regularize the cars’ position information, a series of communication code is defined in this scope. When testing the car in the project, the car can finish all the tasks as expected. This project provided a well managed starting point for students to investigate more complicated functions of the AGV and improve further the multi-AGV, AGV system.

**Keywords**: Distributed System, AGV, Arduino, ESP8266, Network Protocol, Python.

Acknowledgments

I would like to thank my supervisor and mentor Dr. Limin.Yu. She is a teacher from the EEE department, and she proposed this project to me.

I would like to thankmy mentor Dr. Fei Ma. He is a teacher from Mathematical Science department, and he provided the idea about the construction of distributed system.

I would also like to thank all my teachers in EEEdepartment and my class mates.

And I would like give my sincere thanks to XJTLU.

Contents

[Abstract ii](#_Toc57)

[Acknowledgments iii](#_Toc26463)

[Contents iv](#_Toc8563)

[List of Tables v](#_Toc19539)

[List of Figures vi](#_Toc2377)

[List of Codes vii](#_Toc32465)

[List of Acronyms viii](#_Toc10606)

[Chapter 1 Introduction 1](#_Toc5802)

[1.1 Motivation, Aims and Objective 1](#_Toc7081)

[1.2 Literature Review 1](#_Toc22181)

1)AGV......................................................................................................................1

2)Network Communication.....................................................................................2

3)Dijkstra’s Shortest Path Algorithm....................................................................4

[Chapter 2 Methodology and Results 5](#_Toc27941)

[2.1 Methodology 5](#_Toc10782)

1) Distributed Sytem Realization.............................................................................5

2) Model Construction.............................................................................................6

3) ESP8266 for Wireless Module) ........................................................................15

4)Algorithm Optimization for Collision Controlling.............................................19

5)Python GUI Platform Building...........................................................................19

[2.2 Results 21](#_Toc17829)

[Chapter 3 Conclusion and Future Work 24](#_Toc15527)

[3.1 Conclusion 24](#_Toc26193)

[3.2 Future Work 24](#_Toc3957)

[References 27](#_Toc27407)

[Appendix A. Arduino Code in Car 28](#_Toc14050)

[Appendix B. Arduino Code in ESP8266 34](#_Toc19575)

[Appendix C. Python Code in GUI Software 37](#_Toc8635)

List of Tables

[Table 1: Map Information table 1](#_Toc20987)2

[Table 2: Calculated Shortest Distance Table 1](#_Toc20987)4

Table 3: The Interaction Code between Car and ESP8266............................................... 17

List of Figures

[Figure 1.1.1: Four Layers in Network Communication 3](#_Toc20987)

[Figure 2.1.1: Differences between Distributes and Centralized Method 6](#_Toc20987)

Figure 2.1.2: Real Map Layout in Project........................................................................... 7

[Figure 2.1.3: Program Logic in Arduino Car 7](#_Toc15925)

[Figure 2.1.4: Infrared Sensors 9](#_Toc31537)

[Figure 2.1.5: Working Principle in Line Tracking 9](#_Toc13047)

[Figure 2.1.6: Real Case in Using Infrared Sensors 10](#_Toc25555)

[Figure 2.1.7: Layout of the Project 12](#_Toc16886)

[Figure 2.1.8: ESP8266 Wireless Module 15](#_Toc9038)

[Figure 2.1.9: Program Logic in ESP8266 16](#_Toc12058)

[Figure 2.1.10: Interface of Python Software 20](#_Toc5877)

Figure 2.2.1: Car Construction...........................................................................................21

Figure 2.2.2: Python GUI Software...................................................................................21

Figure 2.2.3: Display the Example 1.................................................................................22

Figure 2.2.4: Software in Example 1.................................................................................22

Figure 2.2.5: Display the Example 2.................................................................................23

Figure 2.2.6: Re-Calculate the Path in Example 3............................................................23

Figure 2.2.7: Software display of Example 3 ...................................................................23

Figure 3.2.1: Servo Working Process................................................................................26

List of Codes

[Code 1: Code of Motor Control 8](#_Toc20987)

[Code 2: Gather the Data from Infrared Sensors 1](#_Toc20987)0

[Code 3: Follow the Line based on Infrared Data 1](#_Toc15925)1

[Code 4: Type the Map Data to Arduino 1](#_Toc31537)3

[Code 5: Calculate the Shortest Distance 1](#_Toc13047)3

[Code 6: Calculate the Angle Car need to Turn 1](#_Toc25555)5

[Code 7: ESP8266 Receive Message from Car 17](#_Toc16886)

[Code 8: ESP8266 Receive Data from Computer 18](#_Toc9038)

[Code 9: ESP8266 Receive Data from Other Cars 18](#_Toc12058)

[Code 10: Arduino Calculate the Shortest Path 1](#_Toc5877)9

List of Acronyms

Term Initial components of the term

AI Artificial Intelligence

FEC Forward Error Correction

FET Field Effect Transistor

AGV Automated Guided Vehicle

TMS Traffic Management System

CD Coordination Diagram

SURF Speed-Up Robust Feature

TCP Transmission Control Protocol

UDP User Datagram Protocol

GUI Graphical User Interface

# Introduction

## Motivation, Aims and Objective

Recently, with the development of Artificial Intelligence, Automated Guided Vehicles (AGVs) become more important in manufacturing industries. According to Gotting [1] over 20,000 AGVs were used in industrial applications till 2000. AGVs not only increase the efficiency in factories and reduce the cost on workers, but are also used to simulate the transit and Automatic Drive. For the Automatic Drive cars, the whole procedure of car moving and path finding is controlled by themselves. It is hard to control a plenty of cars by a central computer. At this aspect, the distributed management proposed in the AGV management system is valuable for the industrial. Consequently, this project aims to build up a prototype of a multi-AGV system with Arduino, and use a distributed way to manage the AGVs. Each car could receive a task from client constantly, at the same time, cars could communicate with each other to avoid conflict, deadlock, and collision.

## Literature Review

1. **AGV**

AGV is the acronym for Automated Guided Vehicle. It is a portable robot and often used in industries factories to transport materials or product. Navigation, path decision and traffic control are three important parts. In navigation, most of the solutions are line following such as tape and laser, because of convenience and lower cost. Dr. Hans Moravec [2] invented and developed the vision guidance for AGV at Carnegie Mellon University, which use the camera to record features along the route. Path decision means the vehicle has to calculate and decide the path. Common solutions are frequency select, path select, and magnetic tape. Frequency select mode detects the different frequencies and decides on the best path. However, this solution is not easily expandable and expensive. Path select mode is based on pre-programmed path and compares the path with a calculated route, which is a rather simple and flexible than previous frequency select mode. Magnetic tape mode provides the path for AGV to follow and strips of the tape in various combinations. In addition, it can lead the AGV to change lane, speed up, slow down and stop. In real AGV applications, the project containing more than one vehicles require a none-collision working environment and an optimized transport times[3]. In their simulation of traffic management system (TMS) in an automated guided vehicle, they apply the Coordination Diagram (CD) to optimize the transport times based on collision control [3].

The most important part of the multi-AGVs project is management system. Chih-Lyang Hwang and Hsing Hao [4] build an RGB vision based line tracking system, which can detect the obstacles through the Speed-Up Robust Feature method (SURF) in 2017. The estimated distance with respect to the vehicle can execute the task of obstacle avoidance and target approach [4]. In his vehicle construction, car use four motors to drive and equipped a camera. In his platform building, the single vehicle tracking the aim and circle had been accomplished, but multi-AGVs working had not been finished.

Domingo Gonzalez and his teammates [5] proposed an optimized design of AGV for mixed type transportation in hospital environments. In their design, a AGV based cart in the hospital had been proposed, which can tow some equipment and be connected with other carts to increase the loading ability [5]. The biggest challenge they pointed out is a control system for managing all the vehicles in the hospital and it will improve the project a lot [5]. Two related work leads that system managing can be a significant issue in the multi-AGVs project.

1. **Network Communication**

In telecommunication, four layers displayed in Figure 1.1.1 are designed in a network protocol. First is “link”, which is responsible for accepting IP data and communicating on the local network link. Next, internet layer handles the communication between machines. Packets will be accepted from transport layer with an identification of receiving machine and encapsulated in IP datagrams. If the datagram is addressed to the local machine, the appropriate transport protocol for the packet is chosen. TCP and UDP are transport layers, which provide communication from application to another. The transport layer should ensure that data transmission without error. Transmission Control Protocol (TCP) is a reliable transmission, which requires three times “handshake” to confirm the connection of transmission. User Datagram Protocol (UDP) use a simple communication method to reduce the connection time, which leads to the unreliability of transmission. At the highest layer, the application layer based on TCP/IP internet provide a brief and streaming way to work. For example, online surfing and chat online are all belonging to this layer.

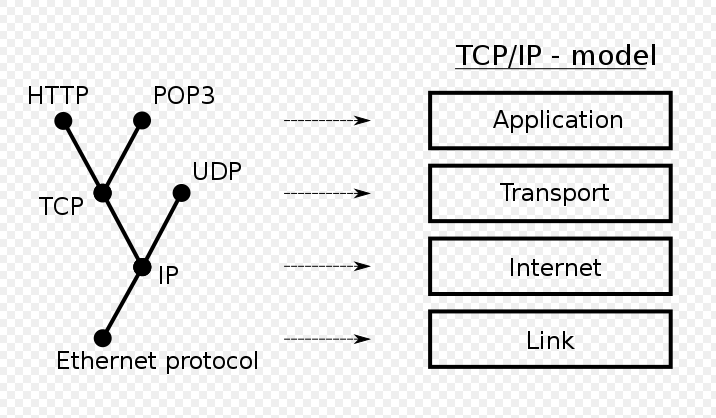


Figure 1.1.1: Four Layers in Network Communication

ESP8266 is suitable for AGV project because of the low cost and consumption. Herman Yuliandoko and his teammate [6] built a flooding detection system based on velocity and water level dam with ESP8266 for Indonesia. They point out ESP8266 is a low cost and low consumption, which can be integrated with internet network and send sensors’ data to the smartphone. They use the ultrasonic sensor to detect the water flood and report the data through wireless module. ESP8266 will use HTTP protocol to send and acquire the request to the web server [6].

TCP communication in ESP8266 is reliable and efficient. In Martin Kajan team design [7], they use Arduino Nano and ESP8266 send TCP messages. They re-programmed the ESP8266 and the vehicles will send the message to TCP server constantly. This research provides a new type of method to control wireless module by programming rather than its basic function. They figure out that the TCP connection is very reliable and efficiency [7].

1. **Dijkstra’s Shortest Path Algorithm**

Dijkstra shortest path algorithm is a basic solution to find the shortest path between nodes in a graph. Edsger W. Dijkstra [8] conceived this algorithm in 1956 to solve 2 problems in connection. First is to calculate the minimunm distance between nodes in graph, and another is find the path of it [8]. In AGV system, Dijkstra algorithm can be used for find the shortest path in the map.

The solution of pathfinding is based on the Dijkstra shortest path algorithm, and some condition requirement should be considered to optimized the algorithm. To optimize the path length and computational time of ocean surveying and exploration, Y.Singh [9] with his team use Dijkstra algorithm to finish the navigation work in ocean exploration. They apply the attractive potential to direct the vehicle towards the destination, which will tend to zero as boat approaches to destination [9]. However, they need to consider more on vehicle dynamics and environmental disturbances in grid map because the model they built has limitations for real circumstance.

# Methodology and Results

## Methodology

1. **Distributed System Realization**

In building distributed multi-AGV prototype, car plays a huge role. Different from centralized prototype, the center operation client can only send tasks to cars, and each car saved its own running data. In other words, this project, center client only need the ability to send network packages to cars, therefore cars should gather all information, calculate path and move.

Figure 2.1.1 shows the difference between distributed and centralized management. In a centralized system, AGV only executes the action after center computer calculating. The algorithm will simulate in the center computer by considering complex conditions. Additionally, send the action to each car. On a different way, cars only receive task, compare the authority with others and calculate an optimized shortest path. As the figure shows, distributed method reduce the processes when running the project. In a long-term, cars can receive the task by any devices rather than the center computer.

Consequently, in this project, well-constructed AGV model and programming can be two important parts. The following sections will discuss how to build an AGV model and details in programming.

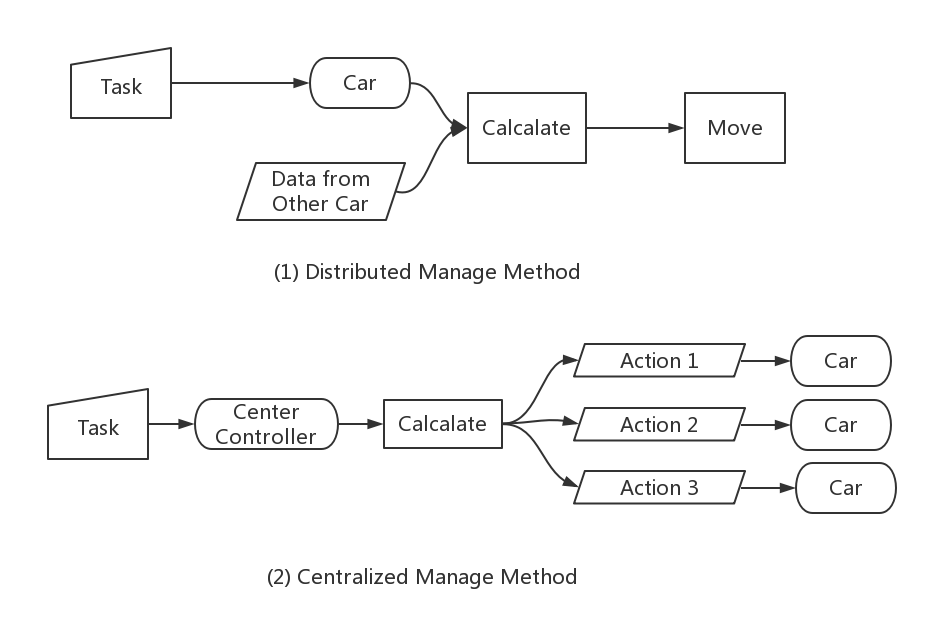


Figure 2.1.1: Differences between Distributes and Centralized Method

1. **Model Construction**

In this project, Arduino board is used because of its low price and simple usage. Arduino Uno board with the I/O expansion board plugged and add the motor module, wireless module and infrared sensors. A car will acquire task information by wireless module, and calculate the shortest path. Next, calculate the angle it needs to turn and do the line tracking. It can communicate with other cars, exchange information and avoid the collision.

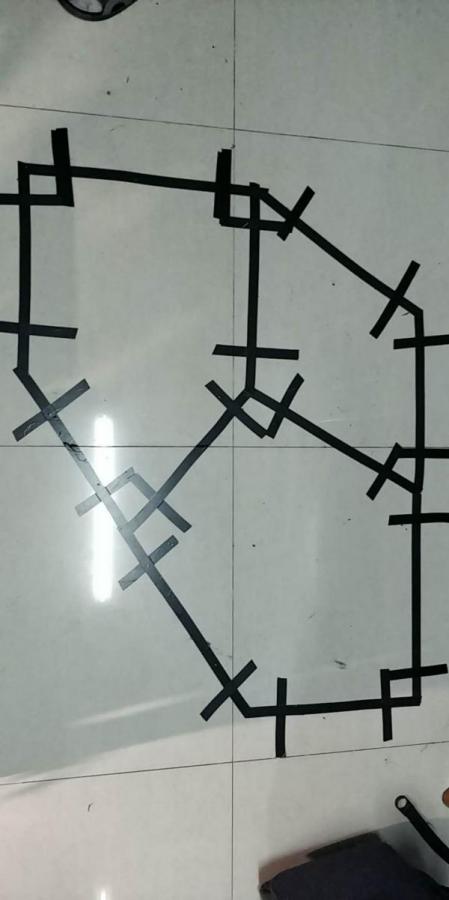


Figure 2.1.2: Real Map Layout in Project

Figure 2.1.2 shows the map layout of this project. 9 points are set in the map and connect the related point in the black line. Additionally, two ends of each line have a short black line perpendicular to connection line. When the car is in the connection line, it will follow the line and move. Once it meets the perpendicular line, the car will stop as finishing this line tracking. Whole program logic is shown in Figure 2.1.3. When the car start to work, Arduino will setup the sensors first. Then the car performs at a idle status, and waits for the wireless task. Once it obtain the task, the car will calculate the path and send data to ESP8266. When the car finish the task, it will return to idle status.

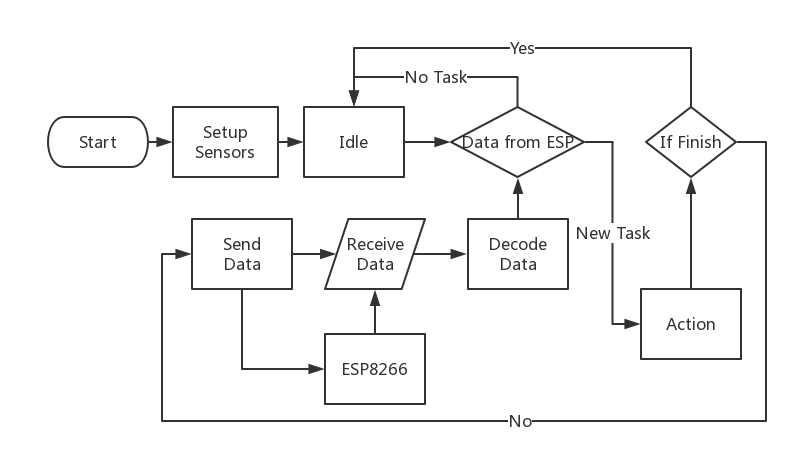
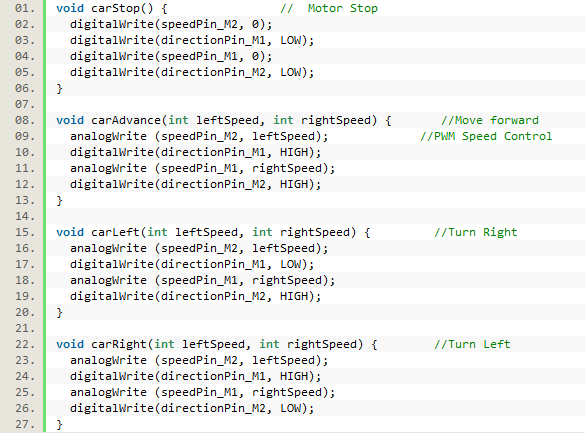


Figure 2.1.3: Program Logic in Arduino Car

* 1. **Motor**

Motor control is the basic part of car’s movement. Each wheel is controlled by two variables, one controls the direction, another controls the speed. Therefore, in this project, 4 Pins are used to control 2 wheels. When using the motor, just set the value to this Pin value as shown in Code 1.



Code 1: Code of Motor Control

* 1. **Line Tracking**

7 infrared sensors are used to detect black line for navigation. Figure 2.1.4 display the outlook of infrared sensor. When infrared detected the black, it will return value 0, or it will return value 1.



Figure 2.1.4: Infrared Sensors

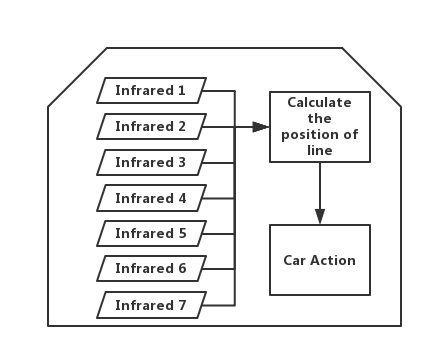
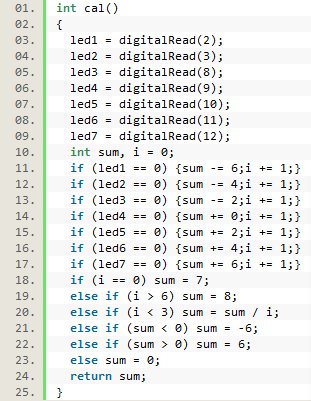


Figure 2.1.5: Working Principle in Line Tracking

Figure 2.1.5 shows the working principle in line tracking. At first, locate the 7 sensors in front of the car and place it as a line as shown in Code 2. Additionally, gather the return value and calculate the position of the black line.



Code 2: Gather the Data from Infrared Sensors

In normal cases, the car will change the speed of left or right to follow the black line. In abnormal case, all sensors detect the black line, which means the car has finished the navigation on this route. The black line is placed shown in Figure 2.1.6, and two types of line is located in a single path.

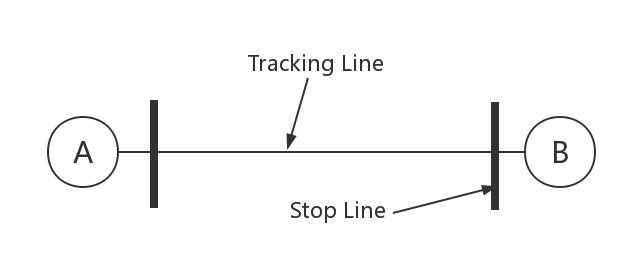
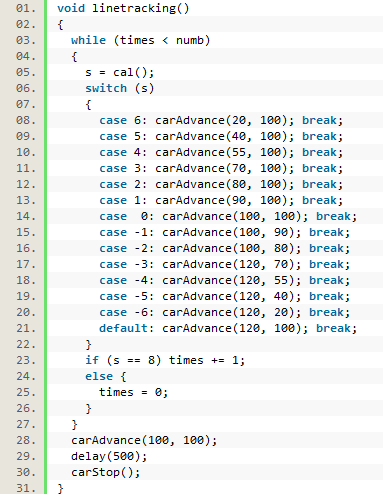


Figure 2.1.6: Real Case in Using Infrared Sensors

As shown in Code 3, based on different cases, the car will change the motor setting to achieve line tracking.



Code 3: Follow the Line based on Infrared Data

* 1. **Dijkstra Shortest Path for Routing**

Dijkstra algorithm is a basic and simple algorithm, which is suitable for this project at this stage. Figure 2.1.7 is the layout of this project.

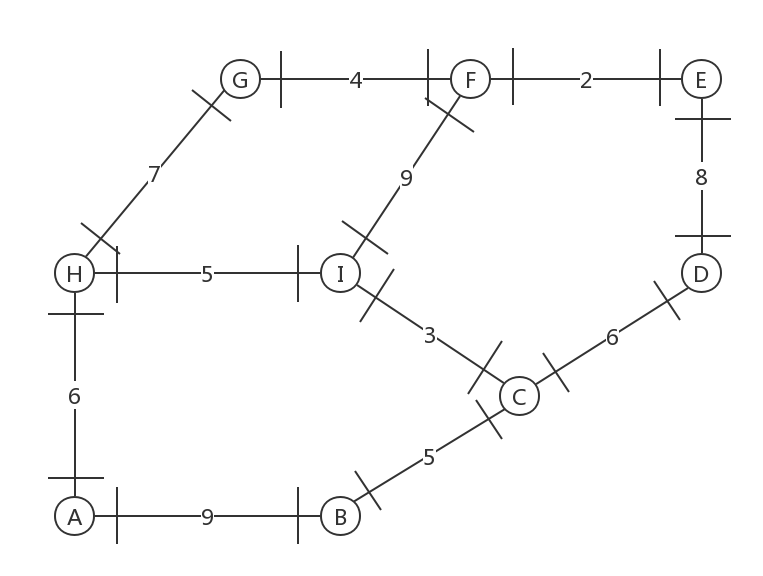
****

Figure 2.1.7: Layout of the Project

Firstly, transfer the map information to a table 1, and save it in the Arduino board. The table shows the connection relationship of each point. The car will use Code 4 to save the map in a 2-D array.

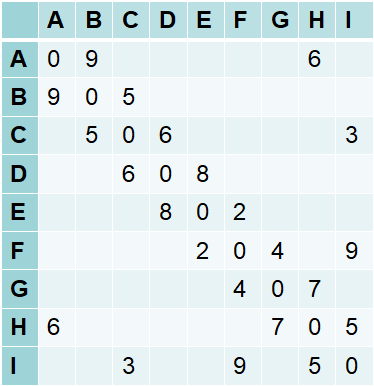
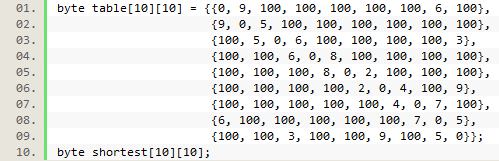
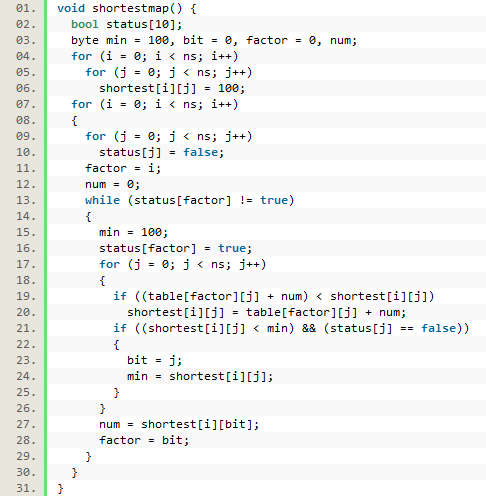


Table 1: Map Information table



Code 4: Type the Map Data to Arduino

Next, use an optimized enumeration method to calculate the shortest distance of each point shown in Code 5 and save it to table 2.



Code 5: Calculate the Shortest Distance

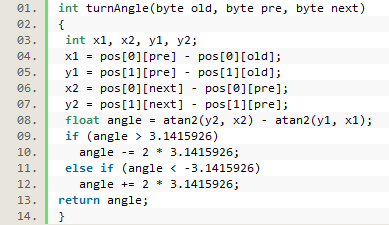


Table 2: Calculated Shortest Distance Table

Then, the shortest path can be found by searching the table 1 and table 2. People can easily find the shortest path from any point to any point. For example, a car needs to go from A to E. Though table 1, A is connected to B and H. Then, it can be found that shortest distance from B to E is 19 and from H to R is 13 at table 2. From table 1, A to B is 9 and A to H is 6. The whole distance go to B firstly is 28, and H is 19. Thus, the first point the car should bound to is H. The whole path can be found by using this algorithm.

* 1. **Turn Angle before Line Tracking**

When a task is sent to Arduino car, the car will calculate the optimized shortest path to the destination. Once the next point is decided, the car will calculate the direction and angle to which it needs to turn. Code 6 display the angle calculation in Arduino.



Code 6: Calculate the Angle Car need to Turn

Use the vector to calculate the angle.



“old”, “pre”, and “next” mean the position of the old point, present point, and next point.



1. **ESP8266 for Wireless Module**

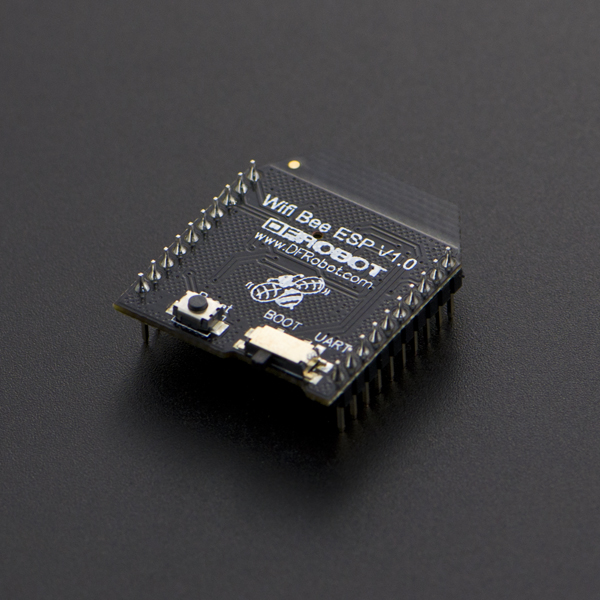


Figure 2.1.8: ESP8266 Wireless Module

ESP8266 wireless module shown in Figure 2.1.8 is selected in this project because of price and convenience. Three communication ways including serial port, TCP, and UDP, are used in the module.

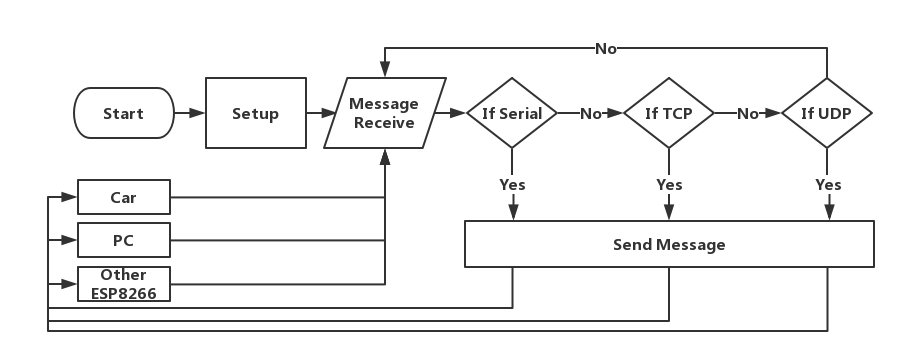


Figure 2.1.9: Program Logic in ESP8266

1. **Re-Program the ESP8266**

Initially, ESP8266 is compiled by the manufacturer and launch a series of instructions named “AT Instruction” to control the module. By sending instructions to the module, it will work as asked. However, the code compiled in ESP8266 is much more complex, and most of it this project has no need to use. Moreover, it lowers the stability of wireless module and increases the running pressure of Arduino. Therefore, in this section, the wireless module has been re-programmed. Programming logic is simple and shown in Figure 2.1.9, the ESP8266 will connect to the remote operation client TCP server and other cars’ UDP. When detected the message from the Serial port, UDP or TCP server, it will handle the information and transport it.

Moreover, it must set a new instruction principle between Arduino and ESP module. New instructions have 5 bits data including starting sign, ID, function, present point and next/aimed point., which is displayed in Table 3.

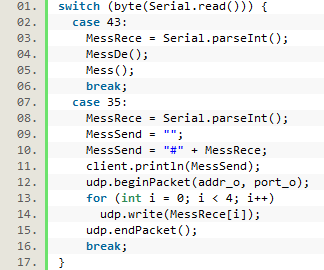
|  |  |  |
| --- | --- | --- |
| Starting Sign | ASC II | Meaning |
| # | 35 | Car information |
| + | 43 | Function from car |
| - | 45 | Function from ESP8266 |
| Function |  |  |
| 0 | 48 | The Car is idle/ESP do not get TCP Task |
| 1 | 49 | Getting task |
| 2 | 50 | Getting other cars’ information |

Table 3: The Interaction Code between Car and ESP8266

1. **Serial port**

The Serial port is the key communication tool to connect the modules and boards. The connection is reliable because it connected by Pin. Each module or board can print messages in a select serial port, and others can read the message form it.

Here Code 7 is an example of how to use the serial port.

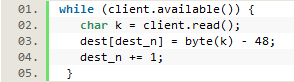


Code 7: ESP8266 Receive Message from Car

The car will send two kinds of the message to wireless module. One is information about it, another is sending a request. As the Table 3 presented, when a head message is “43” referring to “+”, the car is requested to obtain the task or other car information. If obtain “35” referring to “#”, the car is sending the position information and ESP8266 will send the same message to the remote controller or other cars.

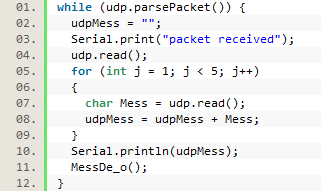
1. **TCP** **and UDP**

TCP connect the wireless module with the remote operation client through network IP and port. Code 8 is an example of how to use TCP communicate with remote operation client. In this example, ESP8266 is testing if receive a TCP server message which will send tasks. If so, save the task in an array.



Code 8: ESP8266 Receive Data from Computer

UDP is a fast way to enable every car to communicate with each other. The usage is same to TCP packet. Code 9 shows the wireless is searching available UDP packages, which will store the other cars’ position information.

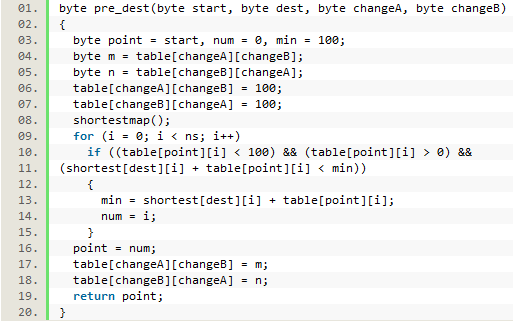


Code 9: ESP8266 Receive Data from Other Cars

1. **Algorithm Optimization for Collision Controlling**

In collision controlling, the strategy is to find a new route to follow. When two car is near, two cars will send the basic information to each other. The authority will decide which car should choose another route.

The code 10 in Arduino will calculate the next place and avoiding collision.



Code 10: Arduino Calculate the Shortest Path

Acquired another cars’ position information, the car will check the authority with others. Once the authority is low than others, the map will change and choose another path to follow.

1. **Python GUI Platform Building**

For a better performance in demonstration, Python is selected to build a TCP server, send the task to cars and display the real-time position of the car.

Python is kind of script language, which is easy to create a GUI program. Before using python to write a software, an open source platform software named OpenTCS has been considered. However, this software is complex and hard to start. That is the reason python is used in this project.

In this software program, TCP communication, Thread working, and Graphic User Interface are three main part. TCP function should bind an IP address and port and build a TCP server. Once a car connects to the server, it will display one of color to represent the car.

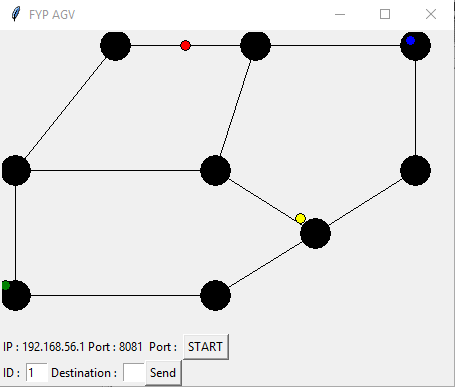


Figure 2.1.10: Interface of Python Software

The figure shows a basic layout of software, and it is consist of three part: map information, starting the server and sending task. In the figure, there are 4 different color dot which represents 4 different cars connecting to the server. The car will return the present position information and display in the map in various color. The controller can check if the car is going to the right place.

## 

## 2.2 Results

1. **Car and Software**

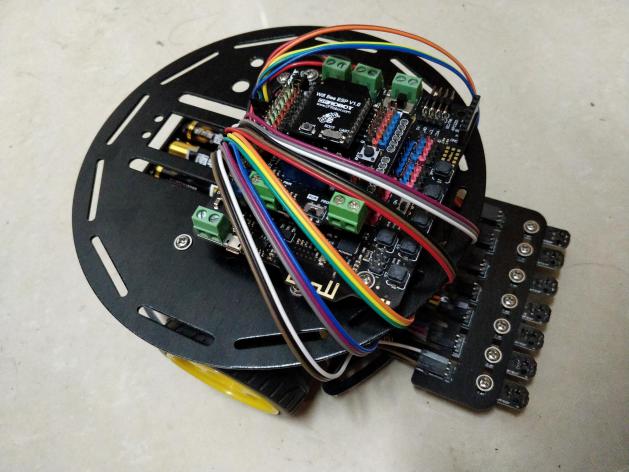


Figure 2.2.1: Car Construction

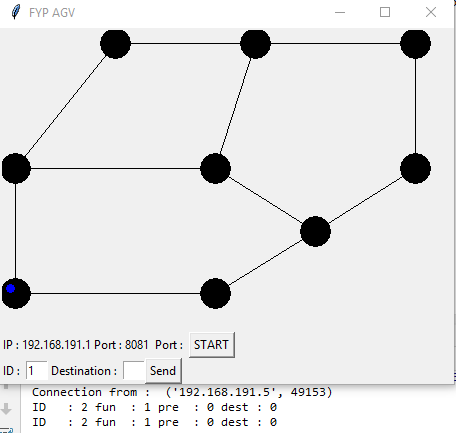


Figure 2.2.2: Python GUI Software

Figure 2.2.1 display the AGV vehivle in this project. In the control software, it is defined that number 0-8 represent the A-I points in Figure 2.1.7 and the car start from “point 0”.

1. **Example 1:Send Single Task to Car**

After sending the task to the car, the car will calculate the path and move. In this example, I send task “6” to the car, which means the car should go from “point 0” to “point 6”. The calculated shortest path is “0-7-6”. After finishing the task, car will stop at “point 6” as shown in Figure 2.2.3. The Car which represented by a “blue dot” will report its position to the control panel, and the software is shown in Figure 2.2.4.

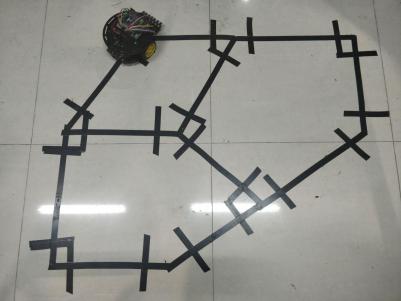
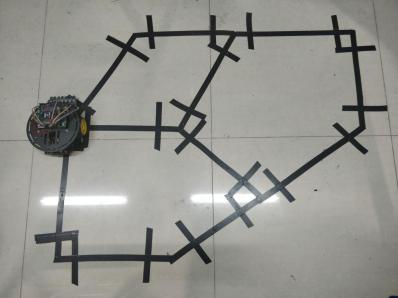
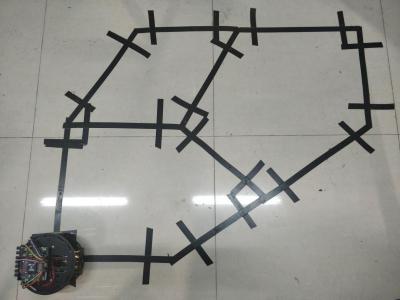


Figure 2.2.3: Display the Example 1

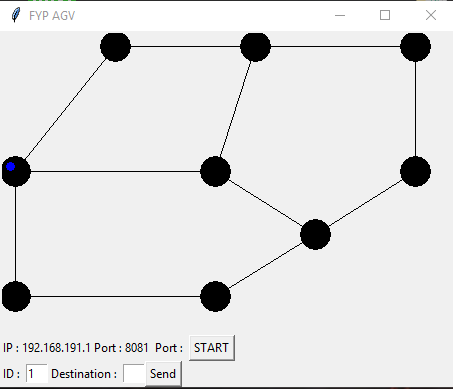
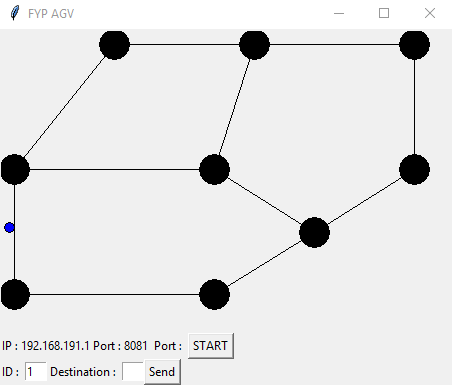
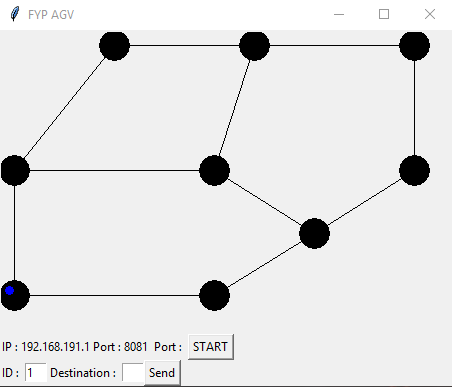
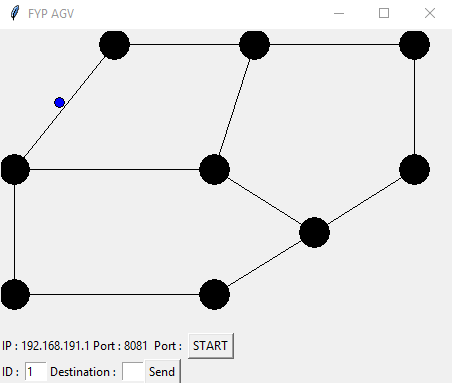
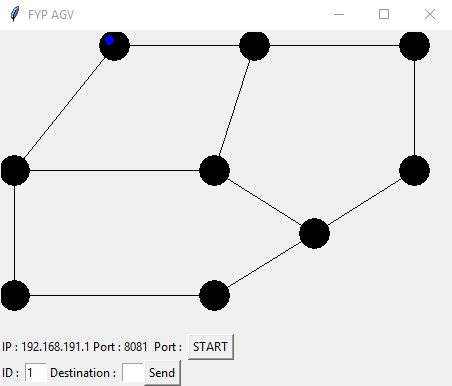
  

Figure 2.2.4: Software in Example 1

1. **Example 2: Send Several Task to the Car**

After receiving the task from control panel, the ESP8266 will save the task in an array. Once the car request the task, ESP8266 will send task in order. In this example, I send the task “6” and task “4”. Continue with the example 1, the car arrive the “point 6” and turn to “point 5”. Then as shown in Figure 2.2.5, the car go to “point 4”.

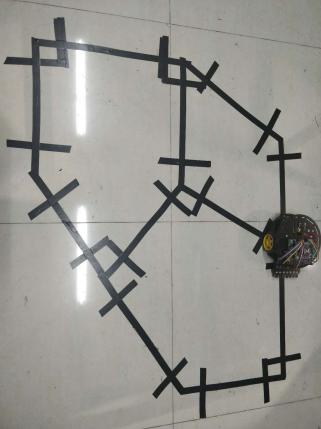
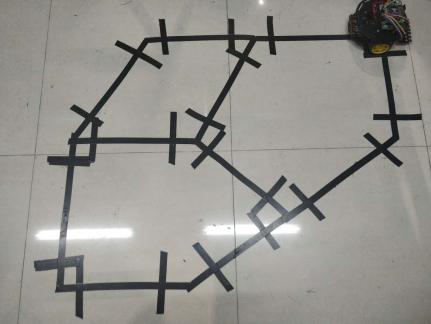
  

Figure 2.2.5: Display the Example 2

1. **Example 3: Another Car Block the Path “6-7”**

Because I only built one AGV vehicle, in multi-AGVs demonstration, I must use the software “TCP/UDP Netassist” to simulate another car. Software simulating another car send the UDP message to the car “#1267”, which is same code instruction as the Table 3. The simulated car has the ID “1” and is going from “point 6” to “point 7”. As shown in Figure 2.2.6, when the car arrive the “point 7”, the simulated car send the message to it. The car will calculate another path “7-8-5-6” and execute. Figure 2.2.7 display the software action when the car is acting as Figure 2.2.6. “Blue dot” represent the car, and “Red dot” represent the simulated car.

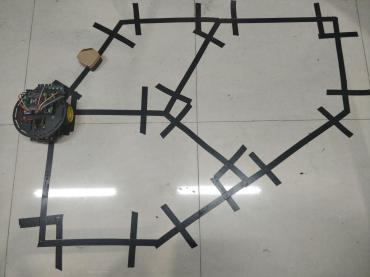
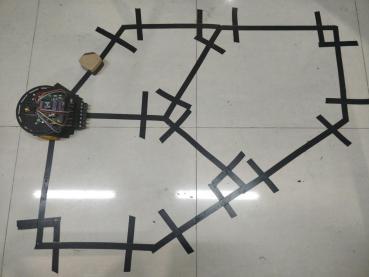
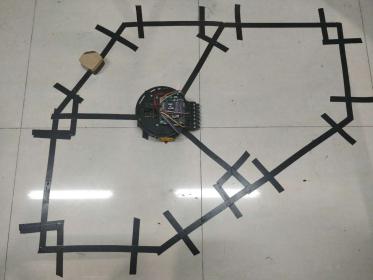
  

Figure 2.2.6: Re-Calculate the Path in Example 3

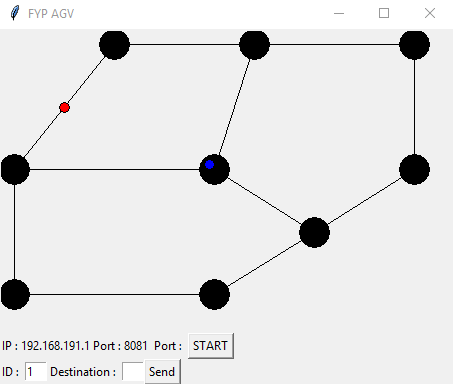
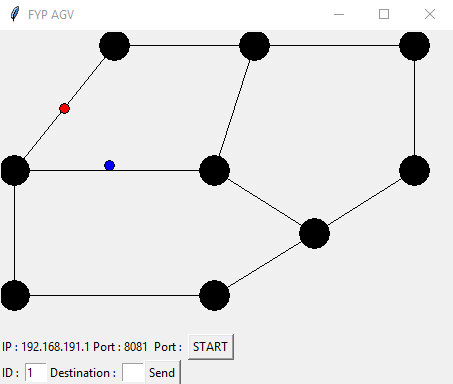
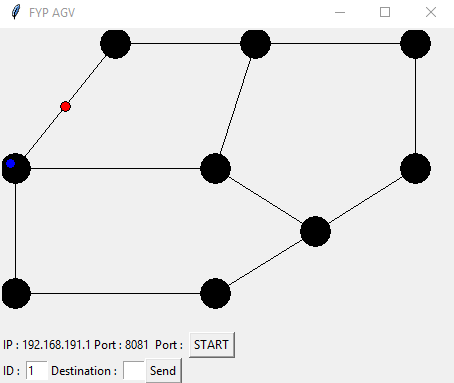


Figure 2.2.7: Software display of Example 3

# Conclusion and Future Work

## Conclusion

To conclude, in this project I constructed a AGV vehicle and designed a distributed system. I programmed the Arduino Uno board, ESP8266, and python control panel. In the construction of AGV car, infrared sensors help to follow the line and stop and the optimized Dijkstra shortest path algorithm is used to navigation. In the programming of wireless module, ESP8266 is used to communicate with control center and other cars. Python software is programmed to send task to cars and display the position of the AGVs. However, this project does not work perfectly in navigation part, and many aspects need to be improved in the future.

In my opinion, to finish this project, I learned lots of knowledge in Arduino, wireless network protocol, python programming, shortest path algorithm, and optimization. The more progress I finished in this project, there is more knowledge I need to learn. Finishing this final year project is not the end of research in this AGV area, but a new brand start of more challenged study.

## Future Work

Back to the evaluation of the whole project, the positioning method, optimized algorithm and batter did not work perfectly.

Firstly, the optimized routing algorithm to avoid the collision is not perfect enough. The algorithm optimized in this project only considers few kinds of the conditions in the collision, but the type of collisions is various. Lack of sufficient collision conditions considered leads to the occurrence of deadlocks in the algorithms. In the future work, this project should use some more complex model to optimize the algorithm. In this project, it costed much time on car construction and wireless ESP8266 learning, consequently, there was less time left for algorithm optimizing. In the future work, an ultrasonic can be added in every vehicle. When the car meets human or bottle, the ultrasonic sensor will detect the barrier and stop until the road is clear. On the other hands, when the calculation in the car is no response or the wireless is not working unexpectedly, the car still can avoid the collision. More specific design of algorithm or integrating the ultrasonic sensor may help the project a lot.

The battery problem can be solved by a stable power source and compass module. In this Arduino car, only 5 AA batteries are used, which cause the decrease ratio of voltage is large. In this project design, after calculating the next point of the map, the car need to calculate the angle based on the location of the point. Time delay method is used in this car, and the delay time is decided by angle. However, every time adjusting the delay time, the angle is different because the power of the car is not stable. Therefore, 5 AA batteries are not suitable for this design. In the future, more batteries should be used in this project or use a stable DC source such as the portable power supply. These two solutions aim to stabilize the working power of cars. On the other way, it can add a “compass module” to control the rotate. When a car is turning, the car will not stop the rotate until the compass module confirms the angle. In my point of view, this is the best solution for this kind of problem.

Next, the positioning method is hard to execute. The positioning method in this project is to detect a perpendicular black line, and it is not easy for the car to execute. After line tracking, if the car beyond the orbit, some infrared sensors may not detect the black line. Therefore, the car will not stop when it arrives the place. For improving the positioning, RFID (Radio Frequency Identification) can help to recognize the point on the map. Or the other positioning method like GPS can help to obtain the position of itself easier. The vehicle must know its destination. In my opinion, we can set a reference point and this point can be detected by the car. Add the ultrasonic or infrared sensor to detect the reference point. In the figure 3.2.1, before following the line, servo need to turn an angle to which the sensors can detect the reference point in the next place. Once the sensor detects the point, the car will stop.

Lastly, the python software only provides the basic operation of TCP server and display the map information. In the future, more functions can be added to this software. This idea comes up while demonstrating the project. Because I only builtone AGV car, I cannot test the whole distributed system. If I can spend more time on the python software, I can build the virtual AGV model in software. Then I can test the optimized algorithm easily. Python is a functional script program language like MATLAB, which provides many functions to do the network communication work and to build graphical user interface. If the software can be finished in the future, it can be used in industrial factories or academic research to develop the management of the multi-AGV system.

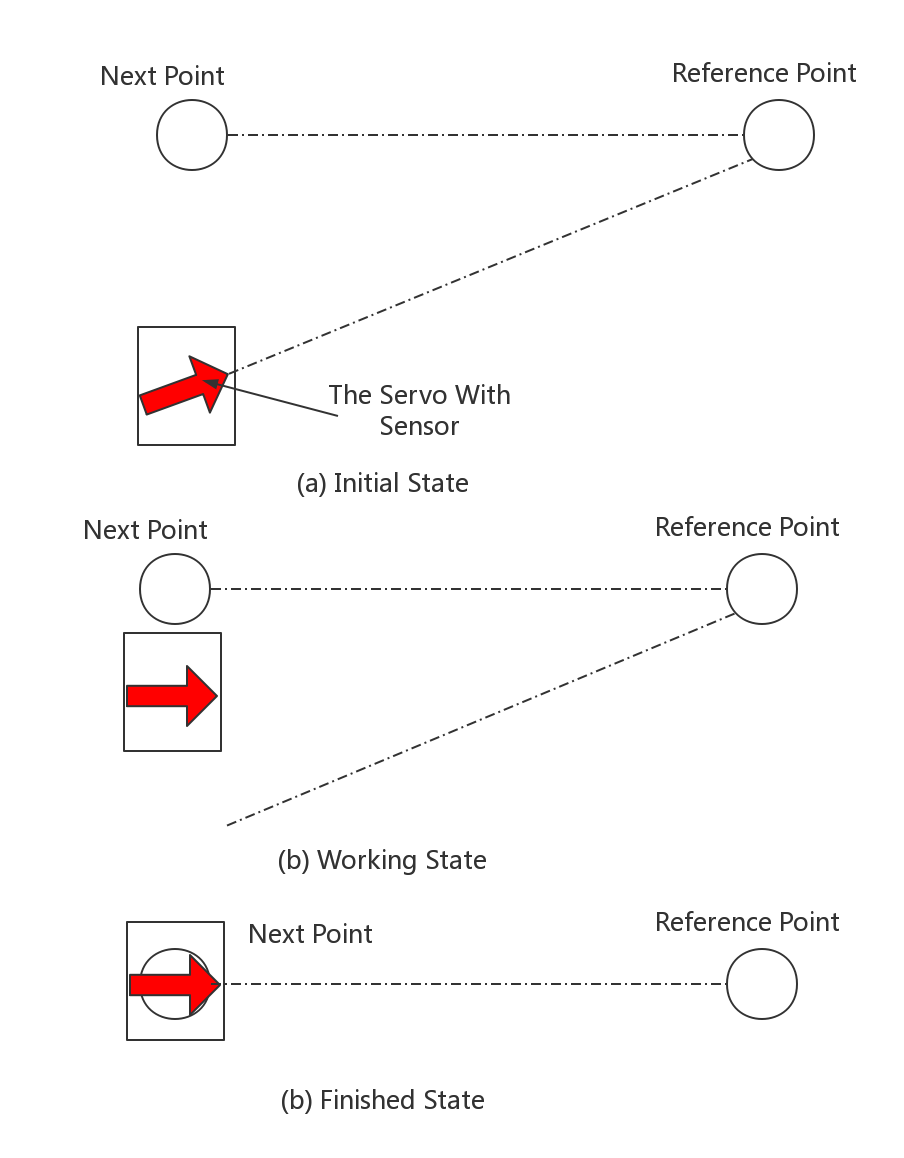


Figure 3.2.1: Servo Working Process

References

[1] H.H.Gotting, “Automation and steering of vehicles in ports.” Jan. 2000.

[2] H.P. Moravec, “Robust Navigation by Probabilistic Volumetric Sensing”, in Carnegie Mellon University, 30, Apr. 2002.

[3] O.Roberto, “Traffic Management of Automated Guided Vehicles in Flexible Manufacturing Systems”, 13, Apr. 2012.

[4] H.Chih-Lyang, H.Hsing-Hao, “Experimental validation of a car-like automated guided vehicle with trajectory tracking, obstacle avoidance, and target approach,” 29, Oct. 2017.

[5] D.Gonzalez, L.Romero\*, Maria del Mar Espinosa, Manuel Dominguez, “An optimization Design Proposal Of Automated Guided Vehicles For Mixed Type Transportation In Hospital environments,” in PLoS ONE, vol. 12 Issue 5, p1-22. 22p. 31, May. 2017.

[6] Yuliandoko, Herman, Subono, Wardhany, Vivien Arief, Pramono, Sholeh Hadi,Siwindarto, Ponco, “Design of flooding detection system based on velocity and water level DAM with ESP8266,” in 2nd ICITISEE, 396-401 Nov. 2017 .

1. Kajan. M, Sovcik.J, Duchon. F, Mrafko.L, Florek. M, Beno. P, “Sensoric subsystem of automated guided vehicle: TCP communication between SIMATIC S7 PLC and Arduino” in 2014 23rd RAAD , 2014, 1-6.
2. E.W.Dijkstra, “A Node on Two Problem in Connexion with Graphs,” in Numerische Mathematik, pp 269-271, 1959.

[9] Y.Singh, S.Sharma, R.Sutton, D.Hatton, “Towards use of Dijkstra Algorithm for Optimal Navigation of an Unmanned Surface Vehicle in a Real-Time Marine Environment with results from Artificial Potential Field”, in LCC:Transportation and communications, vol 12, Issue 1, pp 125-131, 2018.

Appendix A. Arduino Code in Car

#define numb 20

#define ns 9

int speedPin\_M1 **=** 5**;** //M1 Speed Control

int speedPin\_M2 **=** 6**;** //M2 Speed Control

int directionPin\_M1 **=** 4**;** //M1 Direction Control

int directionPin\_M2 **=** 7**;** //M1 Direction Control

byte table**[**10**][**10**]** **=** **{{**0**,** 9**,** 100**,** 100**,** 100**,** 100**,** 100**,** 6**,** 100**},**

**{**9**,** 0**,** 5**,** 100**,** 100**,** 100**,** 100**,** 100**,** 100**},**

**{**100**,** 5**,** 0**,** 6**,** 100**,** 100**,** 100**,** 100**,** 3**},**

**{**100**,** 100**,** 6**,** 0**,** 8**,** 100**,** 100**,** 100**,** 100**},**

**{**100**,** 100**,** 100**,** 8**,** 0**,** 2**,** 100**,** 100**,** 100**},**

**{**100**,** 100**,** 100**,** 100**,** 2**,** 0**,** 4**,** 100**,** 9**},**

**{**100**,** 100**,** 100**,** 100**,** 100**,** 4**,** 0**,** 7**,** 100**},**

**{**6**,** 100**,** 100**,** 100**,** 100**,** 100**,** 7**,** 0**,** 5**},**

**{**100**,** 100**,** 3**,** 100**,** 100**,** 9**,** 100**,** 5**,** 0**}};**

byte shortest**[**10**][**10**];**

byte pos **=** **{{**0**,** 5**,** 7.5**,** 10**,** 10**,** 6**,** 2.5**,** 0**,** 5**},** **{**10**,** 10**,** 7.5**,** 5**,** 0**,** 0**,** 0**,** 5**,** 5**}};**

int angle**;**

byte i**,** j **=** 0**,** times**,** error **=** 0**,** s **=** 0**;**

byte led1 **=** 0**,** led2 **=** 0**,** led3 **=** 0**,** led4 **=** 0**,** led5 **=** 0**,** led6 **=** 0**,** led7 **=** 0**;**

byte start **=** 0**,** dest **=** 0**;**

byte old **=** 1**,** pre **=** 0**,** next **=** 0**,** ID **=** 2**,** function **=** 1**;**

byte pre\_o**,** next\_o **=** 0**,** ID\_o **=** 1**,** function\_o **=** 0**;**

String MessSend **=** ""**,** MessRece **=** ""**;**

byte status **=** 9**;**

void setup**()** **{**

delay**(**2000**);**

Serial**.**begin**(**115200**);**

setup\_IR**();**

shortestmap**();**

**if** **(**Serial**.**read**()** **>=** 0**)** **{}**

delay**(**1000**);**

MessSend **=** ""**;**

MessSend **=** "#" **+** String**(**ID**)** **+** String**(**function**)** **+** String**(**pre**)** **+** String**(**next**);**

Serial**.**println**(**MessSend**);**

**}**

void loop**()** **{**

**switch** **(**status**)** **{**

**case** 0**:**

delay**(**500**);**

MessSd**();**

**switch** **(**function**)**

**{**

**case** 1**:** status **=** 1**;** **break;**

**case** 2**:** status **=** 2**;** **break;**

**case** 0**:** status **=** 9**;** **break;**

**}**

**break;**

**case** 1**:** //Dest

**if** **(**byte**(**Serial**.**read**())** **==** 45**)** **{**

delay**(**100**);**

MessRece **=** Serial**.**parseInt**();**

MessDe**();**

delay**(**500**);**

**if** **(**dest**==**9**)**

function **=** 1**;**

**else**

function **=** 2**;**

status **=** 0**;**

**}**

**break;**

**case** 2**:**

**if** **(**byte**(**Serial**.**read**())** **==** 45**)** **{**

MessRece **=** Serial**.**parseInt**();**

MessDe\_o**();**

Serial**.**println**(**MessRece**);**

delay**(**500**);**

status **=** 3**;**

**}**

**break;**

**case** 3**:** //calculate the route

next **=** pre\_dest**(**pre**,** dest**,** pre\_o**,** next\_o**);**

angle **=** turnAngle**(**old**,** pre**,** next**);**

**if** **(**angle **<=** 0**)**

**{**

carRight**(**150**,** 150**);**

angle **=** **-**angle**;**

**}**

**else**

carLeft**(**150**,** 150**);**

angle**=**int**(**angle**\*250);**

delay**(**angle**);**

Serial**.**println**(**angle**);**

MessSend **=** ""**;**

MessSend **=** "#" **+** String**(**ID**)** **+** String**(**function**)** **+** String**(**pre**)** **+** String**(**next**);**

Serial**.**println**(**MessSend**);**

status **=** 4**;**

**break;**

**case** 4**:** //line tracking

linetracking**();**

times **=** 0**;**

old **=** pre**;**

pre **=** next**;**

**if** **(**pre **==** dest**)**

status **=** 5**;**

**else** status **=** 0**;**

**break;**

**case** 5**:** //finish

Serial**.**println**(**"Finish!"**);**

status **=** 0**;**

function **=** 0**;**

**break;**

**default:**

delay**(**2000**);**

Serial**.**println**(**""**);**

Serial**.**println**(**"New Mession Comes"**);**

status **=** 0**;**MessSend **=** ""**;**

MessSend **=** "#" **+** String**(**ID**)** **+** String**(**function**)** **+** String**(**pre**)** **+** String**(**next**);**

Serial**.**println**(**MessSend**);**

function **=** 1**;**

**break;**

**}**

**}**

// Decode Task from ESP8266

void MessDe**()**

**{**

ID **=** byte**(**MessRece**[**0**])** **-** 48**;**

function **=** byte**(**MessRece**[**1**])** **-** 48**;**

pre **=** byte**(**MessRece**[**2**])** **-** 48**;**

dest **=** byte**(**MessRece**[**3**])** **-** 48**;**

**}**

// Decode Other car information from ESP8266

void MessDe\_o**()**

**{**

ID\_o **=** byte**(**MessRece**[**0**])** **-** 48**;**

function\_o **=** byte**(**MessRece**[**1**])** **-** 48**;**

pre\_o **=** byte**(**MessRece**[**2**])** **-** 48**;**

next\_o **=** byte**(**MessRece**[**3**])** **-** 48**;**

**}**

// Send the position information to ESP8266

void MessSd**()**

**{**

MessSend **=** ""**;**

MessSend **=** "+" **+** String**(**ID**)** **+** String**(**function**)** **+** String**(**pre**)** **+** String**(**dest**);**

Serial**.**println**(**MessSend**);**

**}**

// Setup the infrared sensors

void setup\_IR**()**

**{**

pinMode**(**2**,** INPUT**);**

pinMode**(**3**,** INPUT**);**

pinMode**(**8**,** INPUT**);**

pinMode**(**9**,** INPUT**);**

pinMode**(**10**,** INPUT**);**

pinMode**(**11**,** INPUT**);**

pinMode**(**12**,** INPUT**);**

**}**

//gather the return value of infrared sensors

void test**()**

**{**

led1 **=** digitalRead**(**2**);**

led2 **=** digitalRead**(**3**);**

led3 **=** digitalRead**(**8**);**

led4 **=** digitalRead**(**9**);**

led5 **=** digitalRead**(**10**);**

led6 **=** digitalRead**(**11**);**

led7 **=** digitalRead**(**12**);**

**}**

// calculate the position of black line

int cal**()**

**{**

test**();**

int sum**,** i **=** 0**;**

**if** **(**led1 **==** 0**)** **{**sum **-=** 6**;**i **+=** 1**;}**

**if** **(**led2 **==** 0**)** **{**sum **-=** 4**;**i **+=** 1**;}**

**if** **(**led3 **==** 0**)** **{**sum **-=** 2**;**i **+=** 1**;}**

**if** **(**led4 **==** 0**)** **{**sum **+=** 0**;**i **+=** 1**;}**

**if** **(**led5 **==** 0**)** **{**sum **+=** 2**;**i **+=** 1**;}**

**if** **(**led6 **==** 0**)** **{**sum **+=** 4**;**i **+=** 1**;}**

**if** **(**led7 **==** 0**)** **{**sum **+=** 6**;**i **+=** 1**;}**

**if** **(**i **==** 0**)** sum **=** 7**;**

**else** **if** **(**i **>** 6**)** sum **=** 8**;**

**else** **if** **(**i **<** 3**)** sum **=** sum **/** i**;**

**else** **if** **(**sum **<** 0**)** sum **=** **-**6**;**

**else** **if** **(**sum **>** 0**)** sum **=** 6**;**

**else** sum **=** 0**;**

**return** sum**;**

**}**

// Motor control

void carStop**()** **{** // Motor Stop

digitalWrite**(**speedPin\_M2**,** 0**);**

digitalWrite**(**directionPin\_M1**,** LOW**);**

digitalWrite**(**speedPin\_M1**,** 0**);**

digitalWrite**(**directionPin\_M2**,** LOW**);**

**}**

void carAdvance**(**int leftSpeed**,** int rightSpeed**)** **{** //Move forward

analogWrite **(**speedPin\_M2**,** leftSpeed**);** //PWM Speed Control

digitalWrite**(**directionPin\_M1**,** HIGH**);**

analogWrite **(**speedPin\_M1**,** rightSpeed**);**

digitalWrite**(**directionPin\_M2**,** HIGH**);**

**}**

void carLeft**(**int leftSpeed**,** int rightSpeed**)** **{** //Turn Right

analogWrite **(**speedPin\_M2**,** leftSpeed**);**

digitalWrite**(**directionPin\_M1**,** LOW**);**

analogWrite **(**speedPin\_M1**,** rightSpeed**);**

digitalWrite**(**directionPin\_M2**,** HIGH**);**

**}**

void carRight**(**int leftSpeed**,** int rightSpeed**)** **{** //Turn Left

analogWrite **(**speedPin\_M2**,** leftSpeed**);**

digitalWrite**(**directionPin\_M1**,** HIGH**);**

analogWrite **(**speedPin\_M1**,** rightSpeed**);**

digitalWrite**(**directionPin\_M2**,** LOW**);**

**}**

// Calculate the shortest distance table

void shortestmap**()** **{**

bool status**[**10**];**

byte min **=** 100**,** bit **=** 0**,** factor **=** 0**,** num**;**

**for** **(**i **=** 0**;** i **<** ns**;** i**++)**

**for** **(**j **=** 0**;** j **<** ns**;** j**++)**

shortest**[**i**][**j**]** **=** 100**;**

**for** **(**i **=** 0**;** i **<** ns**;** i**++)**

**{**

**for** **(**j **=** 0**;** j **<** ns**;** j**++)**

status**[**j**]** **=** **false;**

factor **=** i**;**

num **=** 0**;**

**while** **(**status**[**factor**]** **!=** **true)**

**{**

min **=** 100**;**

status**[**factor**]** **=** **true;**

**for** **(**j **=** 0**;** j **<** ns**;** j**++)**

**{**

**if** **((**table**[**factor**][**j**]** **+** num**)** **<** shortest**[**i**][**j**])**

shortest**[**i**][**j**]** **=** table**[**factor**][**j**]** **+** num**;**

**if** **((**shortest**[**i**][**j**]** **<** min**)** **&&** **(**status**[**j**]** **==** **false))**

**{**

bit **=** j**;**

min **=** shortest**[**i**][**j**];**

**}**

**}**

num **=** shortest**[**i**][**bit**];**

factor **=** bit**;**

**}**

**}**

**}**

// Calculate the next point of car

byte pre\_dest**(**byte start**,** byte dest**,** byte changeA**,** byte changeB**)**

**{**

byte point **=** start**,** num **=** 0**,** min **=** 100**;**

byte m **=** table**[**changeA**][**changeB**];**

byte n **=** table**[**changeB**][**changeA**];**

table**[**changeA**][**changeB**]** **=** 100**;**

table**[**changeB**][**changeA**]** **=** 100**;**

shortestmap**();**

**for** **(**i **=** 0**;** i **<** ns**;** i**++)**

**if** **((**table**[**point**][**i**]** **<** 100**)** **&&** **(**table**[**point**][**i**]** **>** 0**)** **&&** **(**shortest**[**dest**][**i**]** **+** table**[**point**][**i**]** **<** min**))**

**{**

min **=** shortest**[**dest**][**i**]** **+** table**[**point**][**i**];**

num **=** i**;**

**}**

point **=** num**;**

table**[**changeA**][**changeB**]** **=** m**;**

table**[**changeB**][**changeA**]** **=** n**;**

**return** point**;**

**}**

// calculate the angle it need to rotate

int turnAngle**(**byte old**,** byte pre**,** byte next**)**

**{**

int x1**,** x2**,** y1**,** y2**;**

x1 **=** pos**[**0**][**pre**]** **-** pos**[**0**][**old**];**

y1 **=** pos**[**1**][**pre**]** **-** pos**[**1**][**old**];**

x2 **=** pos**[**0**][**next**]** **-** pos**[**0**][**pre**];**

y2 **=** pos**[**1**][**next**]** **-** pos**[**1**][**pre**];**

float angle **=** atan2**(**y2**,** x2**)** **-** atan2**(**y1**,** x1**);**

**if** **(**angle **>** 3.1415926**)**

angle **-=** 2 **\*** 3.1415926**;**

**else** **if** **(**angle **<** **-**3.1415926**)**

angle **+=** 2 **\*** 3.1415926**;**

**return** angle**;**

**}**

// based on the position of black line, follow the line

// and stop at the perpendicular black line

void linetracking**()**

**{**

**while** **(**times **<** numb**)**

**{**

s **=** cal**();**

**switch** **(**s**)**

**{**

**case** 6**:** carAdvance**(**20**,** 100**);** **break;**

**case** 5**:** carAdvance**(**40**,** 100**);** **break;**

**case** 4**:** carAdvance**(**55**,** 100**);** **break;**

**case** 3**:** carAdvance**(**70**,** 100**);** **break;**

**case** 2**:** carAdvance**(**80**,** 100**);** **break;**

**case** 1**:** carAdvance**(**90**,** 100**);** **break;**

**case** 0**:** carAdvance**(**100**,** 100**);** **break;**

**case** **-**1**:** carAdvance**(**100**,** 90**);** **break;**

**case** **-**2**:** carAdvance**(**100**,** 80**);** **break;**

**case** **-**3**:** carAdvance**(**120**,** 70**);** **break;**

**case** **-**4**:** carAdvance**(**120**,** 55**);** **break;**

**case** **-**5**:** carAdvance**(**120**,** 40**);** **break;**

**case** **-**6**:** carAdvance**(**120**,** 20**);** **break;**

**default:** carAdvance**(**120**,** 100**);** **break;**

**}**

**if** **(**s **==** 8**)** times **+=** 1**;**

**else** **{**

times **=** 0**;**

**}**

**}**

carAdvance**(**100**,** 100**);**

delay**(**500**);**

carStop**();**

**}**

Appendix B. Arduino Code in ESP8266

#include "SoftwareSerial.h"

#include <ESP8266WiFi.h>

#include <WiFiClient.h>

#include <WiFiUdp.h>

const char **\***ssid **=** "test"**;**

const char **\***password **=** "19960101"**;**

const char **\***host **=** "192.168.191.1"**;**

const char **\***addr\_o **=** "192.168.191.1"**;**

unsigned int tcpPort **=** 8081**;**

unsigned int udpPort **=** 1001**;**

unsigned int port\_o **=** 1002**;**

WiFiUDP udp**;**

WiFiClient client**;**

byte pre **=** 0**,** next **=** 0**,** ID **=** 2**,** function **=** 1**;**

byte pre\_o **=** 9**,** next\_o **=** 9**,** ID\_o **=** 1**,** function\_o **=** 2**;**

String MessSend **=** ""**,** MessRece **=** ""**,** udpMess **=** ""**;**

byte dest\_n **=** 0**,** dest\_s **=** 0**,** status **=** 0**;**

byte dest**[**10**]** **=** **{**9**,** 9**,** 9**,** 9**,** 9**,** 9**,** 9**,** 9**,** 9**};**

void setup**()**

**{**

Serial**.**begin**(**115200**);**

delay**(**2000**);**

Serial**.**println**();**

Serial**.**println**();**

Serial**.**print**(**"Connecting to "**);**

Serial**.**println**(**ssid**);**

WiFi**.**begin**(**ssid**,** password**);**

// test if connected to the wifi

**while** **(**WiFi**.**status**()** **!=** WL\_CONNECTED**)**

**{**

delay**(**500**);**

Serial**.**print**(**"."**);**

**}**

Serial**.**println**(**""**);**

Serial**.**println**(**"WiFi connected"**);**

// test if connected to the TCP server

**while** **(!**client**.**connected**())**

**{**

**if** **(!**client**.**connect**(**host**,** tcpPort**))**

**{**

Serial**.**println**(**"connection...."**);**

//client.stop();

delay**(**500**);**

**}**

**}**

// print the basic network information of ESP8266

Serial**.**println**(**"Starting UDP"**);**

Serial**.**println**(**"IP address: "**);**

Serial**.**println**(**WiFi**.**localIP**());**

udp**.**begin**(**udpPort**);**

Serial**.**print**(**"Local port: "**);**

Serial**.**println**(**udp**.**localPort**());**

**}**

void loop**()**

**{**

//try to get UDP packet

**while** **(**udp**.**parsePacket**())** **{**

udpMess **=** ""**;**

Serial**.**print**(**"packet received"**);**

udp**.**read**();**

**for** **(**int j **=** 1**;** j **<** 5**;** j**++)**

**{**

char Mess **=** udp**.**read**();**

udpMess **=** udpMess **+** Mess**;**

**}**

Serial**.**println**(**udpMess**);**

MessDe\_o**();**

**}**

//try to get TCP Packet

**while** **(**client**.**available**())** **{**

char k **=** client**.**read**();**

dest**[**dest\_n**]** **=** byte**(**k**)** **-** 48**;**

dest\_n **+=** 1**;**

**}**

//try to get Serial port instruction

**switch** **(**byte**(**Serial**.**read**()))** **{**

**case** 43**:**

MessRece **=** Serial**.**parseInt**();**

MessDe**();**

Mess**();**

**break;**

**case** 35**:**

MessRece **=** Serial**.**parseInt**();**

MessSend **=** ""**;**

MessSend **=** "#" **+** MessRece**;**

client**.**println**(**MessSend**);**

udp**.**beginPacket**(**addr\_o**,** port\_o**);**

**for** **(**int i **=** 0**;** i **<** 4**;** i**++)**

udp**.**write**(**MessRece**[**i**]);**

udp**.**endPacket**();**

**break;**

**}**

**}**

// Messages communication with Arduino Uno

void Mess**()**

**{**

**switch** **(**function**)** **{**

**case** 1**:** MessSd**();** **break;**

**case** 2**:** MessSd\_o**();** **break;**

**case** 0**:** status **=** 0**;** **break;**

**}**

**}**

// Decode the Messages from Arduino

void MessDe**()**

**{**

ID **=** byte**(**MessRece**[**0**])** **-** 48**;**

function **=** byte**(**MessRece**[**1**])** **-** 48**;**

pre **=** byte**(**MessRece**[**2**])** **-** 48**;**

next **=** byte**(**MessRece**[**3**])** **-** 48**;**

**}**

// Decode the Messages from other car

void MessDe\_o**()**

**{**

ID\_o **=** byte**(**udpMess**[**0**])** **-** 48**;**

function\_o **=** byte**(**udpMess**[**1**])** **-** 48**;**

pre\_o **=** byte**(**udpMess**[**2**])** **-** 48**;**

next\_o **=** byte**(**udpMess**[**3**])** **-** 48**;**

**}**

// Send the Task to Arduino

void MessSd**()**

**{**

MessSend **=** ""**;**

MessSend **=** "-" **+** String**(**ID**)** **+** String**(**function**)** **+** String**(**pre**)** **+** String**(**dest**[**dest\_s**]);**

**if** **(**dest**[**dest\_s**]** **!=** 9**)**

dest\_s **+=** 1**;**

Serial**.**println**(**MessSend**);**

**}**

// Send the other car information to Arduino

void MessSd\_o**()**

**{**

MessSend **=** ""**;**

MessSend **=** "-" **+** String**(**ID\_o**)** **+** String**(**function\_o**)** **+** String**(**pre\_o**)** **+** String**(**next\_o**);**

Serial**.**println**(**MessSend**);**

**}**

Appendix C. Python Code in GUI Software

**from** socket **import** **\***

**import** threading

**from** tkinter **import** **\***

##define the TCP server IP##

host **=** '192.168.191.1'

port **=** 8081

buffsize **=** 2048

ADDR **=** **(**host**,** port**)**

point **=** **[**0**,** 1**,** 2**,** 3**,** 4**,** 5**]**

##define map display information##

setX **=** 40

setY **=** 25

index **=** 25

canvas\_width **=** 450

canvas\_height **=** 300

r **=** 15

pos **=** **[[**0**,** 5**,** 7.5**,** 10**,** 10**,** 6**,** 2.5**,** 0**,** 5**],** **[**10**,** 10**,** 7.5**,** 5**,** 0**,** 0**,** 0**,** 5**,** 5**]]**

table **=** **[[**0**,** 9**,** 100**,** 100**,** 100**,** 100**,** 100**,** 6**,** 100**],**

**[**9**,** 0**,** 5**,** 100**,** 100**,** 100**,** 100**,** 100**,** 100**],**

**[**100**,** 5**,** 0**,** 6**,** 100**,** 100**,** 100**,** 100**,** 3**],**

**[**100**,** 100**,** 6**,** 0**,** 8**,** 100**,** 100**,** 100**,** 100**],**

**[**100**,** 100**,** 100**,** 8**,** 0**,** 2**,** 100**,** 100**,** 100**],**

**[**100**,** 100**,** 100**,** 100**,** 2**,** 0**,** 4**,** 100**,** 9**],**

**[**100**,** 100**,** 100**,** 100**,** 100**,** 4**,** 0**,** 7**,** 100**],**

**[**6**,** 100**,** 100**,** 100**,** 100**,** 100**,** 7**,** 0**,** 5**],**

**[**100**,** 100**,** 3**,** 100**,** 100**,** 9**,** 100**,** 5**,** 0**]]**

pos1 **=** pos

m **=** 1

list **=** **[**'list'**]**

color **=** **[**'color'**,** 'red'**,** 'blue'**,** 'green'**,** 'yellow'**]**

##define the GUI software##

**class** **app(**object**):**

**def** \_\_init\_\_**(**self**):**

self**.**root **=** Tk**()**

self**.**root**.**title**(**"FYP AGV"**)**

self**.**map\_ **=** Canvas**(**self**.**root**,** width**=**canvas\_width**,** height**=**canvas\_height**)**

self**.**fm **=** Frame**(**self**.**root**)**

self**.**fmS **=** Frame**(**self**.**root**)**

self**.**fm1 **=** Frame**(**self**.**fm**)**

self**.**fm2 **=** Frame**(**self**.**fm**)**

self**.**var\_ID **=** IntVar**()**

self**.**var\_dest **=** StringVar**()**

self**.**var\_l\_S **=** StringVar**()**

##plot the map##

**for** i **in** range**(**0**,** 9**):**

pos1**[**0**][**i**]** **=** pos**[**0**][**i**]** **\*** setX **+** r

pos1**[**1**][**i**]** **=** pos**[**1**][**i**]** **\*** setY **+** r

self**.**map\_**.**create\_oval**(**pos**[**0**][**i**]** **-** r**,** pos**[**1**][**i**]** **-** r**,** pos**[**0**][**i**]** **+** r**,** pos**[**1**][**i**]** **+** r**,** fill**=**'black'**)**

**for** i **in** range**(**0**,** 9**):**

**for** j **in** range**(**0**,** 9**):**

**if** table**[**i**][**j**]** **<** 100 **and** table**[**i**][**j**]** **>** 0**:**

self**.**map\_**.**create\_line**(**pos**[**0**][**i**],** pos**[**1**][**i**],** pos**[**0**][**j**],** pos**[**1**][**j**])**

##assign the input area##

Label**(**self**.**fm1**,** text**=**'IP : ' **+** str**(**host**)** **+** ' ' **+** 'Port : ' **+** str**(**port**)).**pack**(**side**=**LEFT**)**

Label**(**self**.**fm1**,** text**=**'Port : '**).**pack**(**side**=**LEFT**)**

Button**(**self**.**fm1**,** text**=**'START'**,** command**=**self**.**buttonStart**).**pack**(**side**=**LEFT**)**

Label**(**self**.**fm2**,** text**=**'ID : '**).**pack**(**side**=**LEFT**)**

self**.**var\_ID**.**initialize**(**'1'**)**

Entry**(**self**.**fm2**,** width**=**3**,** textvariable**=**self**.**var\_ID**).**pack**(**side**=**LEFT**)**

Label**(**self**.**fm2**,** text**=**'Destination : '**).**pack**(**side**=**LEFT**)**

Entry**(**self**.**fm2**,** width**=**3**,** textvariable**=**self**.**var\_dest**).**pack**(**side**=**LEFT**)**

Button**(**self**.**fm2**,** text**=**'Send'**,** command**=**self**.**buttonSend**).**pack**(**side**=**LEFT**)**

Label**(**self**.**fmS**,** text**=**'Hello!'**,** textvariable**=**self**.**var\_l\_S**).**pack**()**

##define the layout of the software##

self**.**map\_**.**pack**(**side**=**TOP**)**

self**.**fm1**.**pack**(**side**=**TOP**,** fill**=**BOTH**,** expand**=**NO**)**

self**.**fm2**.**pack**(**side**=**TOP**,** fill**=**BOTH**,** expand**=**NO**)**

self**.**fm**.**pack**(**side**=**LEFT**)**

self**.**fmS**.**pack**(**side**=**LEFT**)**

##define the action when click the Start button##

**def** buttonStart**(**self**):**

**global** list

**global** m

**try:**

carServer **=** socket**(**AF\_INET**,** SOCK\_STREAM**)**

carServer**.**bind**(**ADDR**)**

carServer**.**listen**(**3**)**

cc**,** addr **=** carServer**.**accept**()**

**print(**"Connection from : "**,** addr**)**

car1 **=** carThread**(**cc**,** addr**)**

list**.**append**(**car1**)**

list**[**m**].**start**()**

m **=** m **+** 1

**except:**

**pass**

##define the action when click the Send button##

**def** buttonSend**(**self**):**

**global** list

**try:**

id **=** self**.**var\_ID**.**get**()**

dest **=** self**.**var\_dest**.**get**()**

**print(**int**(**id**),** int**(**dest**))**

list**[**id**].**cc**.**send**(**dest**.**encode**())**

**except:**

**pass**

self**.**var\_ID**.**initialize**(**'1'**)**

self**.**var\_dest**.**initialize**(**''**)**

##define the car in the software##

**class** **car\_info():**

**def** \_\_init\_\_**(**self**,** ID**,** fun**,** pre**,** next**,** dest**):**

self**.**ID **=** ID

self**.**fun **=** fun

self**.**pre **=** pre

self**.**next **=** next

self**.**dest **=** dest

**def** MessDe**(**self**,** Mess**):**

self**.**ID **=** Mess**[**1**]**

self**.**fun **=** Mess**[**2**]**

self**.**pre **=** Mess**[**3**]**

self**.**next **=** Mess**[**4**]**

**return**

**def** dest\_in**(**self**):**

**print(**'Plz enter a destination'**)**

pos **=** 0

**while** pos **in** point**:**

**if** pos **not** **in** point**:**

**print(**'Wrong Destination!'**)**

pos **=** input**(**'Plz enter:'**)**

**print(**'Destination is setting to : '**,** pos**)**

self**.**dest **=** pos

**return** self**.**dest

**def** print\_**(**self**):**

**print(**'ID :'**,** self**.**ID**,** 'fun :'**,** self**.**fun**,** 'pre :'**,** self**.**pre**,** 'dest :'**,** self**.**next**)**

##define the multi-AGV thread##

**class** **carThread(**threading**.**Thread**):**

**def** \_\_init\_\_**(**self**,** cc**,** addr**):**

threading**.**Thread**.**\_\_init\_\_**(**self**)**

self**.**flag **=** 0

self**.**cc **=** cc

self**.**addr **=** addr

self**.**info **=** car\_info**(**0**,** 0**,** 0**,** 0**,** 0**)**

**def** run**(**self**):**

**global** gui

**while** **True:**

data **=** self**.**cc**.**recv**(**buffsize**).**decode**()**

**if** data**[**0**]** **==** '#'**:**

self**.**info**.**MessDe**(**data**)**

self**.**id **=** self**.**info**.**ID

self**.**\_update**(**int**(**self**.**info**.**pre**),** int**(**self**.**info**.**next**),** int**(**self**.**info**.**ID**))**

**if** data**[**0**]** **==** '\*'**:**

pos\_ **=** self**.**info**.**dest\_in**()**

self**.**cc**.**send**(**pos\_**.**encode**())**

**if** data**[**0**]** **==** '$'**:**

self**.**flag **=** 1

**print(**'close'**)**

self**.**info**.**print\_**()**

**if** self**.**flag **==** 1**:**

**break**

self**.**cc**.**close**()**

##update the position of car##

**def** \_update**(**self**,** point1**,** point2**,** ID**):**

**global** gui

**try:**

gui**.**map\_**.**delete**(**self**.**s**)**

**except:**

**pass**

posx **=** **(**pos**[**0**][**point1**]** **+** pos**[**0**][**point2**])** **/** 2

posy **=** **(**pos**[**1**][**point1**]** **+** pos**[**1**][**point2**])** **/** 2

self**.**s **=** gui**.**map\_**.**create\_oval**(**posx **-** ID **\*** 5**,** posy **-** ID **\*** 5**,** posx **-** ID **\*** 5 **+** 10**,** posy **-** ID **\*** 5 **+** 10**,**

fill**=**color**[**ID**])**

**def** main**():**

**global** gui

gui **=** app**()**

mainloop**()**

**if** \_\_name\_\_ **==** "\_\_main\_\_"**:**

main**()**