CONTACTLESS DELIVERY ROBOT

SYSC 4907 ENGINEERING PROJECT PROGRESS REPORT

WINTER 2022

GROUP MEMBERS:

ZHUOXIN MA	101093563
YUCHEN MIAO	101042926
LANG SUN	101059548
YICHEN XIAO	101094948

SUPERVISOR:

PROF. PETER X. LIU

DATE SUBMITTED:

JANUARY 21ST, 2022

Contents

INTRODUCTION

Based on the proposal report, the project is about designing and building an ideal prototype for a “contactless delivery robot”. This project aims at practicing robot design based on business needs and real-life scenario applications.

Because of the prolonged epidemic, the government carried out lock-down policies to reduce exposure at different stages. Many stores lay off staff or significantly reduce the number of employees, making it very hard to keep the original and steady customer load. Also, during this period, people are more and more used to a shopping mode called “curbside pickup” where we order the product online and wait in the parking spot outside the store for pickup. All of these stimulate the thinking about the application of robots in maintaining service inside and outside the store for delivery. The use of robots for curbside delivery can ease the situation where a single staff needs to handle both indoor and outdoor service, as well as balance the labor demand both inside and outside of the store. It can also reduce the waiting time for pickups and the risk of infection due to personal contact during delivery. [1]

THEORY

Based on what we discussed in the introduction, the robot is designed as a head of a short-length “train”. It starts with moving to a specific area waiting for loading. Usually, the waiting-for-loading area is something that can be “defined” and “told the robot” by the store. Once the products have been loaded into the cabin, the robot will have to pair the cabin number with the product code for precision delivery. After it is fully loaded, it can be waiting around the curbside pickup area. When the customer arrives at a pickup spot, they will send a confirmation message with the spot number to the store server, which is then sent as the delivery instruction to the robot. The robot will follow the delivery instruction and direct itself to the specific parking spot. To process the delivery and make sure the customer pickups his and only his products, the robot will ask the customer to input his product code, then find the matched box for pickup. Upon finish, the robot needs to send an identification message to the store server indicating which product has been delivered/ which cabin is empty. Then, if no new delivery instruction is received, the robot can move back to the loading area for new product loading in the empty cabin. Otherwise, head for the next delivery. Attached below, is the designed workflow for the contactless delivery robot. [1]

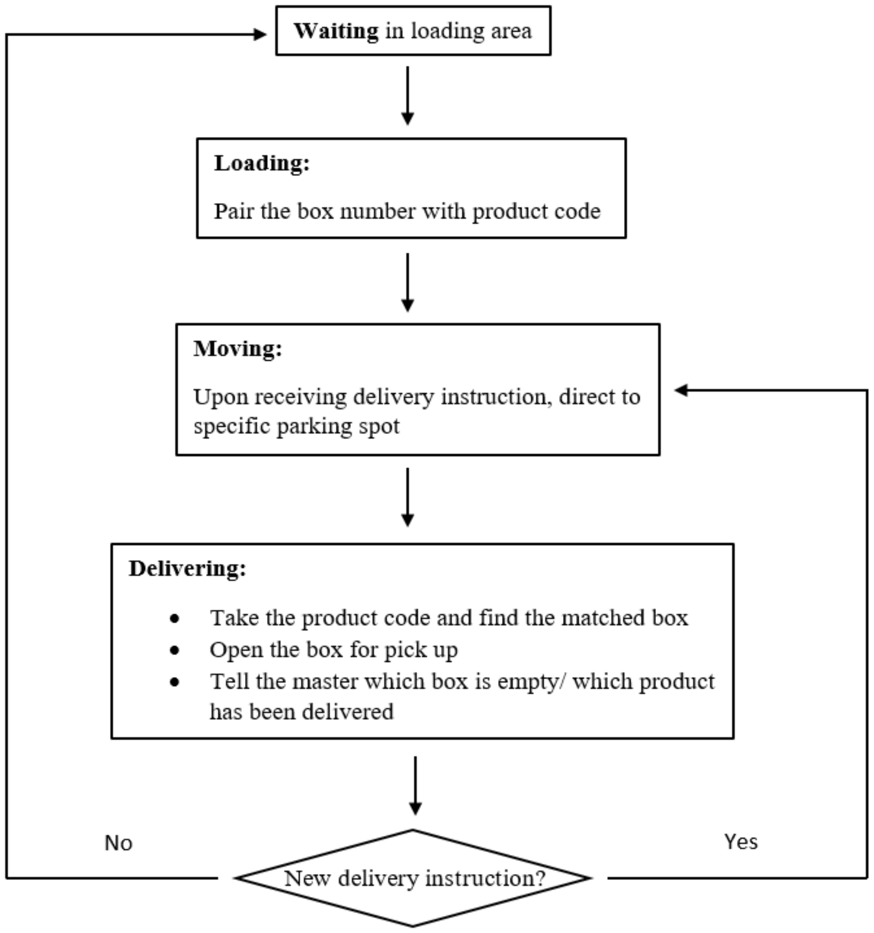

Waiting in loading area
Loading:
Pair the box number with product code
Moving:
Upon receiving delivery instruction, direct to specific parking spot
Delivering:
Take the product code and find the matched box
Open the box for pick up
Tell the master which box is empty/ which product has been delivered
New delivery instruction?
No
Yes

Figure 1. workflow chart

DESIGN PLAN

To achieve our goals, we decided to start the design from the most fundamental function of our robot: mobility by users' instructions in a particular condition. We don't go straight to design our robot but create a relatively small, simple, and extensible prototype. The prototype only needs to move under our manual control, such as RF and Bluetooth wireless remote, in a limited room condition like indoor places with less interference from temperature, humidity, and weather change. Arduino motherboard, extension board, motors, and some plugs about remote control will be used in this stage. After completing the prototype, we then design some routines and store them into its memory to move automatically along with multiple lines and directions. In this case, we only need to set a destination, and the “car” can drive there without any more instruction code. In this stage, ambient is also relatively ideal with less interference. After the plans mentioned above are finished, we can now move our robot outside by adding sensors like temperature, line tracking, ultrasonic sensor (to detect distances), etc., on the Arduino motherboard or extension board. Now the robot can analyze and adapt to the outdoor environment using sensors. We will still design a shield for the robot and choose environmentally friendly material. More than three alternatives for the material will be considered to optimize cost and endurances. The hardware part is almost completed except for some calibration on the routines and sensors. Then, we need to design a software user interface that collects customers’ park positions number and order numbers. An interface used by stores will also be designed so that store workers can enter park position numbers after load delivery on it. [1]

TECHNIQUES

At the current stage, we finished the assembly and testing for the motor part of the robot. The robot can now be controlled to move a certain path by typing some instruction codes on the computer so that the motor part can implement the target coding of automatic pathfinding.

HARDWARE PART

The following table contains the materials used at this stage of the project.

Name	Quantity
Keyestudio quick connectors Motor Drive Shield V2 (L298P)	1
TT Gear Motor (with Fix part)	4
Wheel	4
Acrylic board (25cm*20 cm)	1

Table 1. Current Product List

Motor Drive Shield

The figure below is the PINOUT diagram of the Keyestudio quick connectors Motor Drive Shield V2 used in this project. The motor drive shield is based on the L298 circuit, directly driving DC motors. The dimension of the shield is 66mm*55mm*18mm, with a weight of 25.2g. Logic part input voltage is DC 5V, and the operating range of driving part input voltage is between 6- 18 V. In addition, logic part working current is less than 36 mA, and driving part working current is less than 2A. Furthermore, its working temperature is between -25°C to +130°C, with maximum power dissipation of 25W when the temperature is 75°C [2].

The figure below is the PINOUT Diagram of the motor drive shield. Currently, four DC motor sockets are used to connect four TT gear motors, and four motor direction controls are applied to control the turning direction of motors using eight jumpers. The external power socket is also involved. A 6-cell AA battery holder with 6 AA batteries is connected to the power socket in the testing.

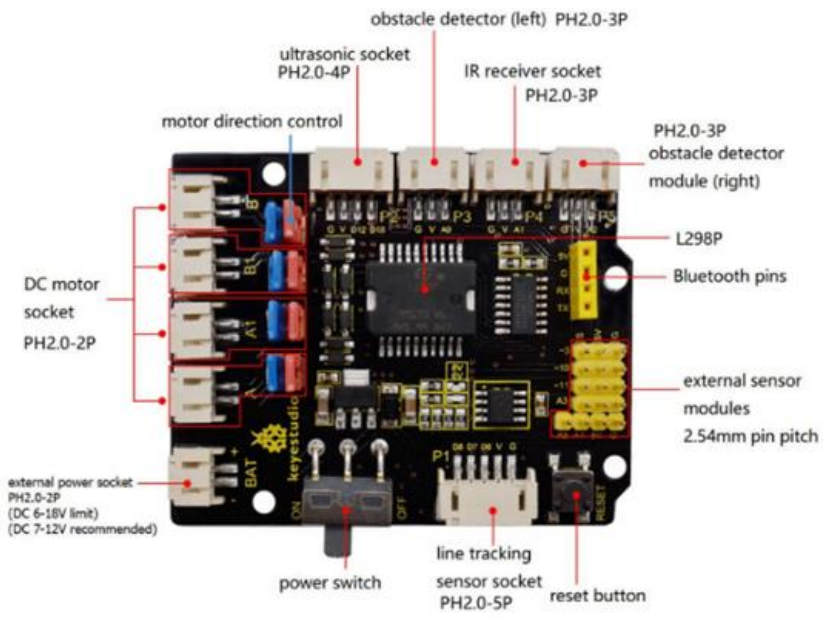

obstacle detector (left) PH2.0-3P
ultrasonic socket PH2.0-4P
IR receiver socket PH2.0-3P
motor direction control
PH2.0-3P obstacle detector module (right)
L298P
Bluetooth pins
DC motor socket PH2.0-2P
external sensor modules 2.54mm pin pitch
external power socket PH2.0-2P (DC 6-18V limit) (DC 7-12V recommended)
power switch
line tracking sensor socket PH2.0-5P
reset button

Figure 2. PINOUT Diagram of the Keyestudio Quick Connectors Motor Drive Shield V2 [2]

TT Gear Motors

Four TT gear motors are the main components of the motor part of the robot. The motor's operating voltage is 3 – 6 V DC, and the gear ratio is 1:48. The minimum speed at 6V is 200 +/- 10% RPM [3]. In Arduino IDE, motors' PWM control speed can be adjusted between 0 to 255.

Connections

Currently, the motor part of the robot is established. The motor drive shield is used to control four gear motors with wheels. The overview of the hardware motor part of the robot is shown in the figure below.

Each motor with welding wire is connected to a DC motor socket: A to lower left motor, A1 to the upper left motor, B1 to the upper right motor, and B to lower right motor. The jumpers near motor A and A1 are horizontal connections. In contrast, the jumpers near motor B and B1 are vertical connections, meaning that the turning direction of motors on the left side is opposite to the one of motors on the right side.

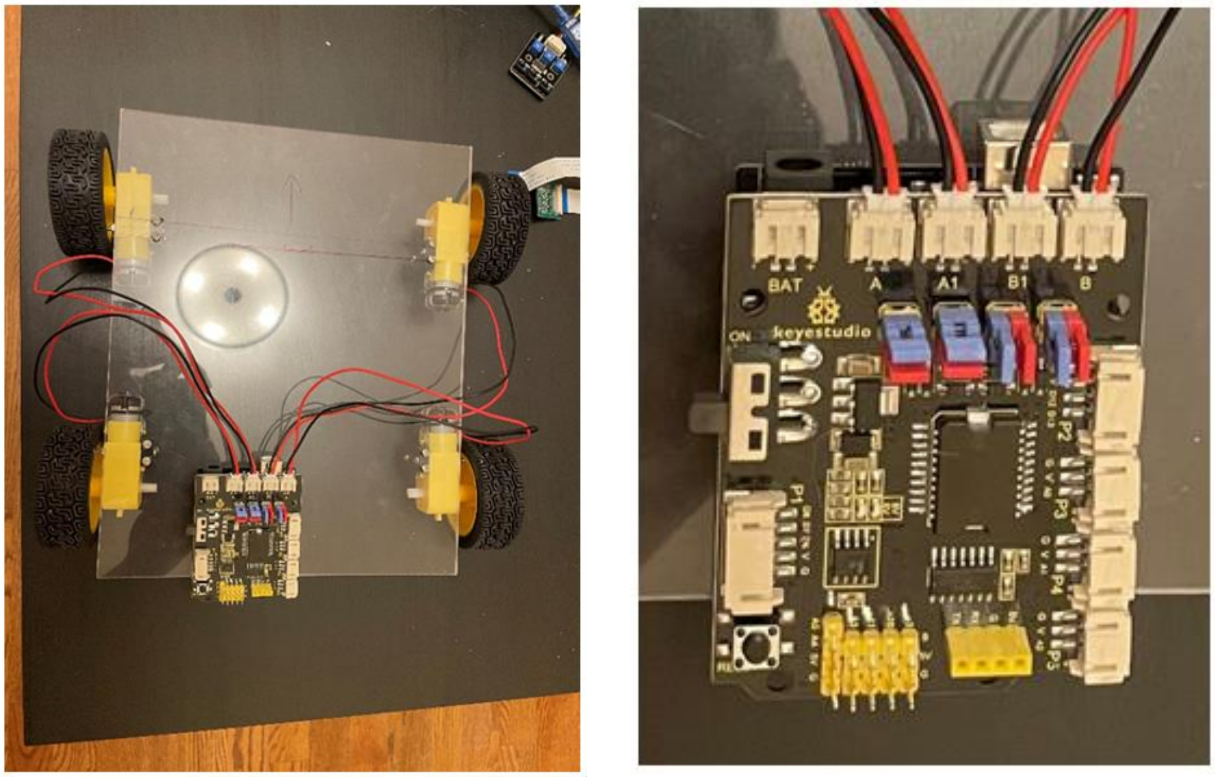


Figure 3. Overview of the Motor Part of the Robot

IMPLEMENTATION

For achieving the pathfinding function, the A* (A star) pathfinding algorithm is selected and programed onto the microcontroller. A* pathfinding is a variation of the graph traverser algorithm used by the Shakey project in the 1960s, which is the world’s first general-purpose mobile robot able to reason about its action. Many other pathfinding algorithms were developed based on this original algorithm. Compared to other pathfinding algorithms, A* takes the G (the distance between the current node and the start node) and H (estimated distance from the current node to the end node) into account, ranking the next nodes to move based on how promising the nodes are and start building the route from this. Thus, the A* avoids brutal search all the movable node and save computing powers. The path generated from A* is not necessarily the best, meaning the shortest, but it is the best path that can be found concerning the power efficiency and computing power of the computing unit. This project is currently using an Arduino Uno as the main control board, which contains an ATmega328p microcontroller with 16Mhz speed, 32KB memory, and 2KB RAM. Thus, it is not a very powerful platform to perform large-scale calculations and the power-saving feature of the A* algorithm is the best option and advantage to this project. The group is currently considering moving the pathfinding part to a more capable platform so the robot can operate on bigger maps, which requires a more powerful computing unit and more memory. However, as a demo prototype, this is sufficient.

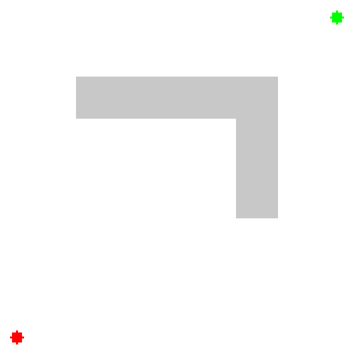
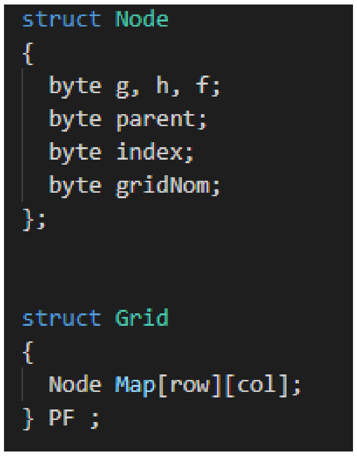
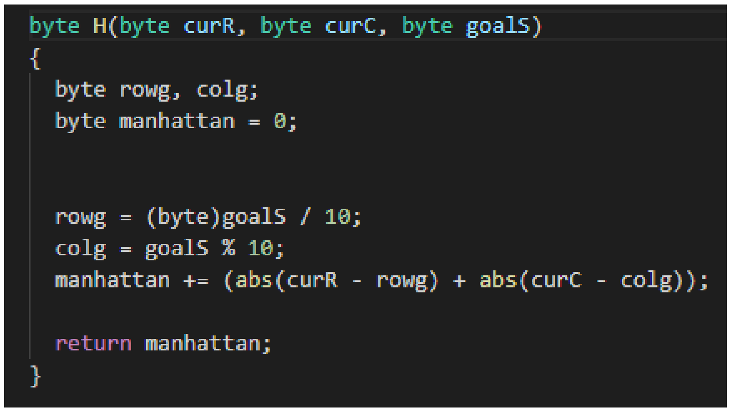


Figure 4. A* pathfinding algorithm prototype [4]

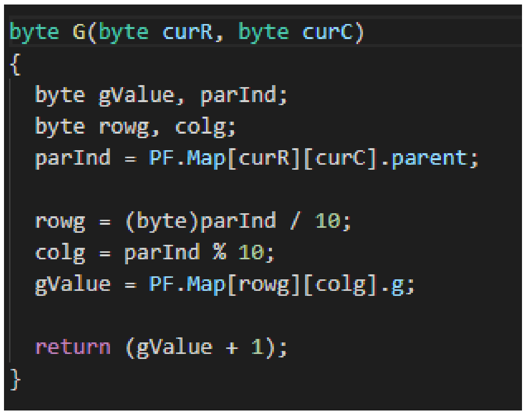
In our project, we first define our test map to be a 10*10 grid, which means a total of 100 available nodes. Then the Manhattan distance heuristic functions are used to calculate the H of the nodes. Also, the G is calculated.



```
struct Node
{
  byte g, h, f;
  byte parent;
  byte index;
  byte gridNom;
};


struct Grid
{
  Node Map[row][col];
} PF ;
```



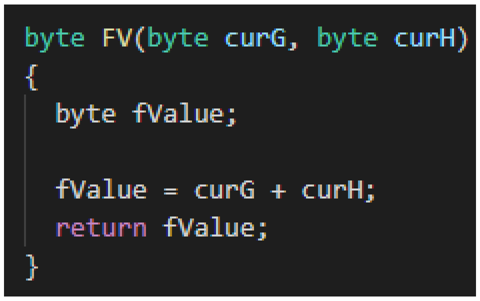
```
byte H(byte curR, byte curC, byte goalS)
{
  byte rowg, colg;
  byte manhattan = 0;


  rowg = (byte)goalS / 10;
  colg = goalS % 10;
  manhattan += (abs(curR - rowg) + abs(curC - colg));

  return manhattan;
}
```



```
byte G(byte curR, byte curC)
{
  byte gValue, parInd;
  byte rowg, colg;
  parInd = PF.Map[curR][curC].parent;

  rowg = (byte)parInd / 10;
  colg = parInd % 10;
  gValue = PF.Map[rowg][colg].g;

  return (gValue + 1);
}
```

Figure 5. sample code for the design

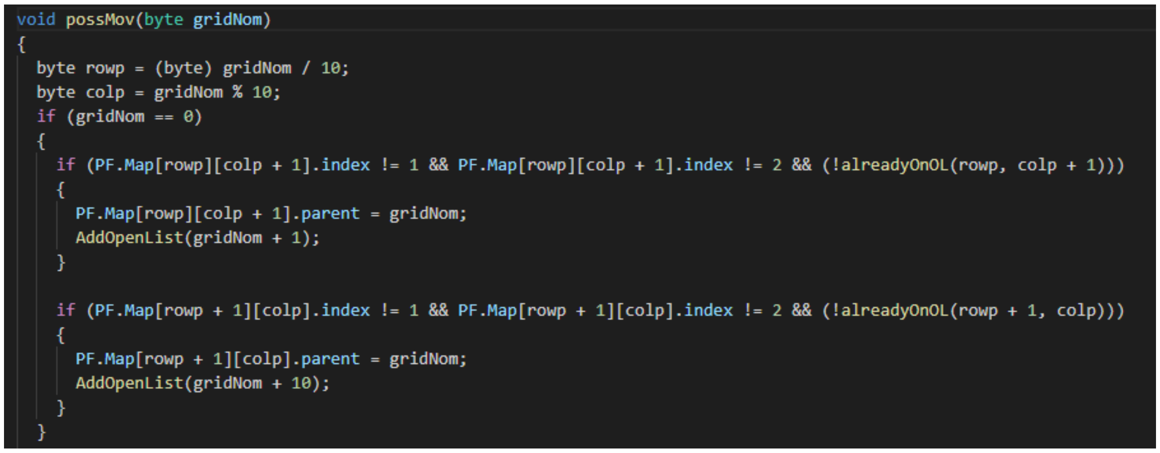
As stated previously, using H and G, the cost of the path taken can be calculated and weighted based on this.



```
byte FV(byte curG, byte curH)
{
    byte fValue;

    fValue = curG + curH;
    return fValue;
}
```

Figure 6. sample code for the design

After the user sets the goal and starting point. The algorithm starts to check the next possible node to move depending on the current node. The following example shows if the current location is at the node with a grid index of 0 (top left corner), then there will only be two possible nodes to move, which is the same for the other three corners. Similar and non-related code is omitted.

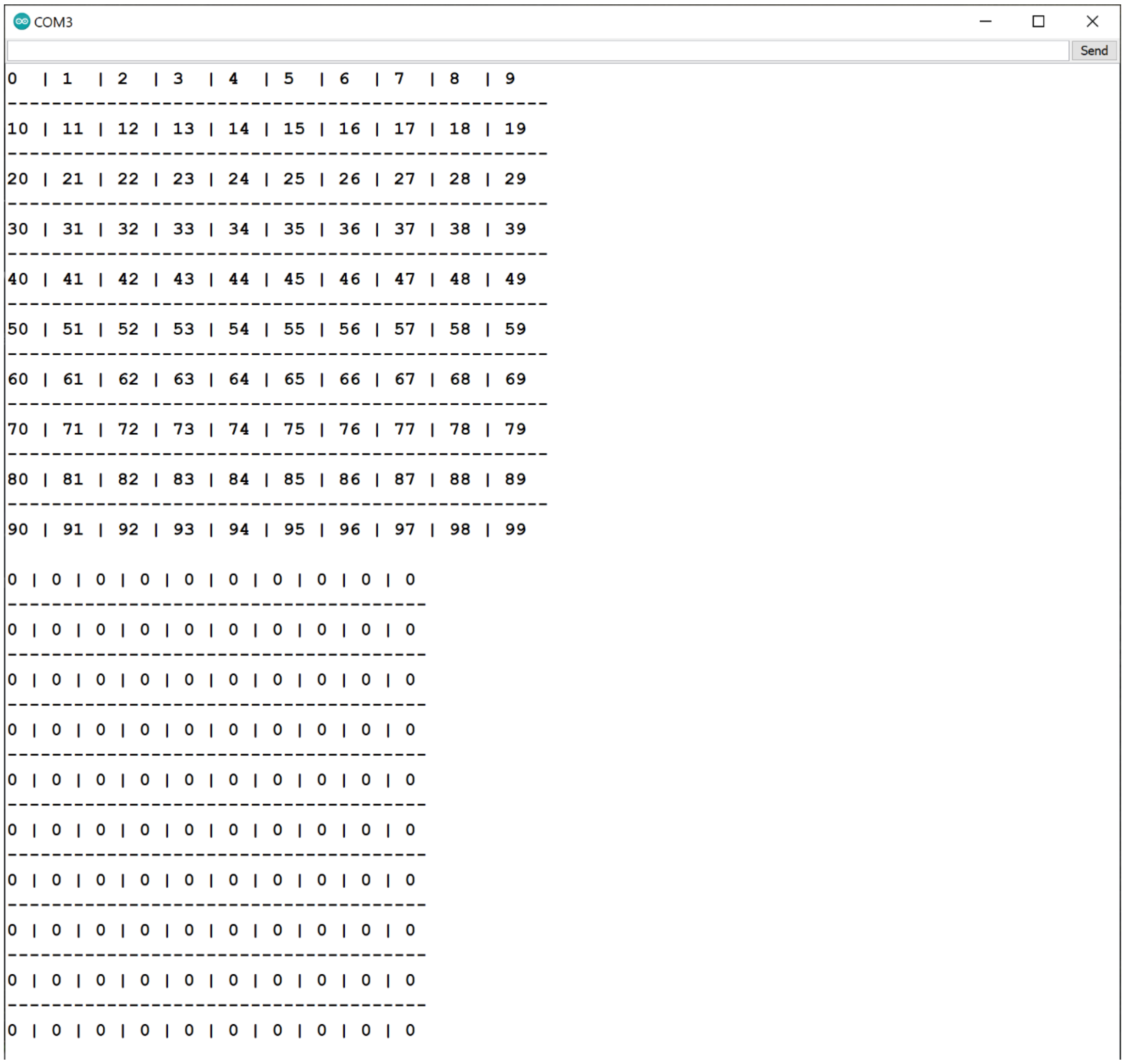


```
void possMov(byte gridNom)
{
  byte rowp = (byte) gridNom / 10;
  byte colp = gridNom % 10;
  if (gridNom == 0)
  {
    if (PF.Map[rowp][colp + 1].index != 1 && PF.Map[rowp][colp + 1].index != 2 && (!alreadyOnOL(rowp, colp + 1)))
    {
      PF.Map[rowp][colp + 1].parent = gridNom;
      AddOpenList(gridNom + 1);
    }

    if (PF.Map[rowp + 1][colp].index != 1 && PF.Map[rowp + 1][colp].index != 2 && (!alreadyOnOL(rowp + 1, colp)))
    {
      PF.Map[rowp + 1][colp].parent = gridNom;
      AddOpenList(gridNom + 10);
    }
  }
```

Figure 7. sample code for the design

Found all next possible nodes and put them into a list, the algorithm started to weight the nodes based on the G and H calculated and re-organize the list. Picking the most promising node as the current node and repeating the same process till the current node matches the goal node.

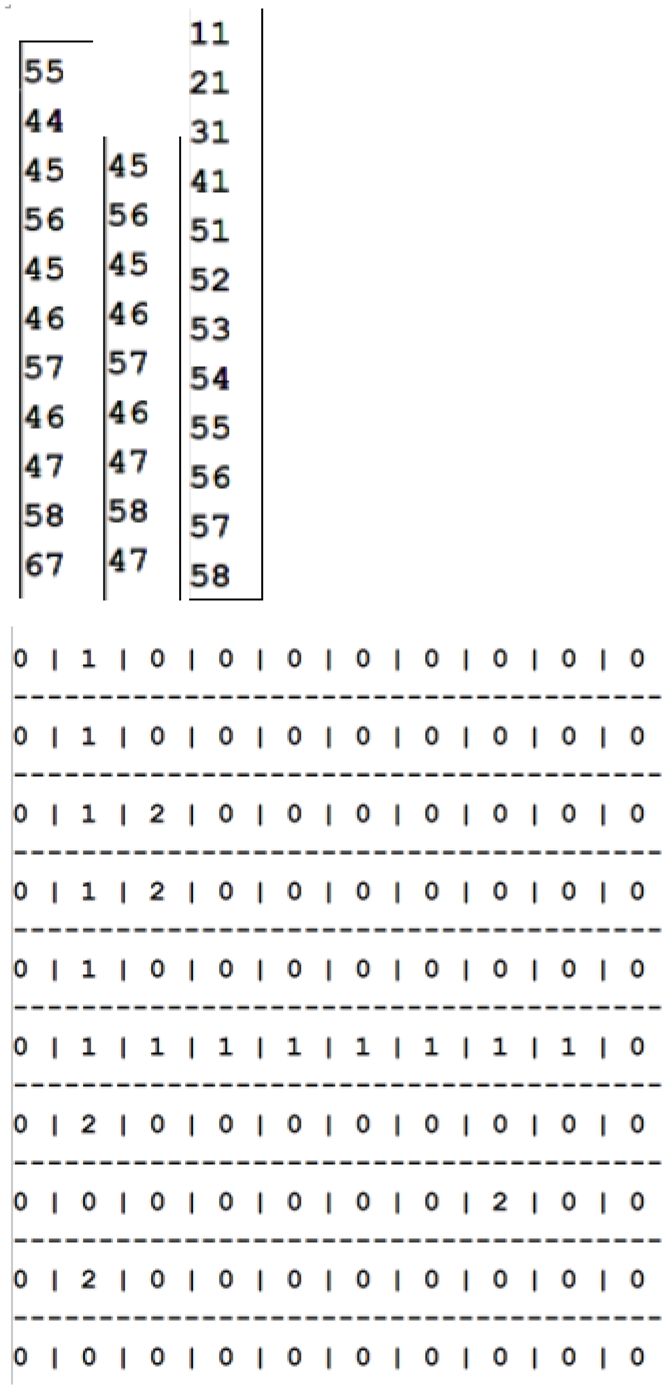
For instance, taking node (1) as starting point and node (58) as the goal node.



```
0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9
-------------------------------------------------------
10  | 11  | 12  | 13  | 14  | 15  | 16  | 17  | 18  | 19
-------------------------------------------------------
20  | 21  | 22  | 23  | 24  | 25  | 26  | 27  | 28  | 29
-------------------------------------------------------
30  | 31  | 32  | 33  | 34  | 35  | 36  | 37  | 38  | 39
-------------------------------------------------------
40  | 41  | 42  | 43  | 44  | 45  | 46  | 47  | 48  | 49
-------------------------------------------------------
50  | 51  | 52  | 53  | 54  | 55  | 56  | 57  | 58  | 59
-------------------------------------------------------
60  | 61  | 62  | 63  | 64  | 65  | 66  | 67  | 68  | 69
-------------------------------------------------------
70  | 71  | 72  | 73  | 74  | 75  | 76  | 77  | 78  | 79
-------------------------------------------------------
80  | 81  | 82  | 83  | 84  | 85  | 86  | 87  | 88  | 89
-------------------------------------------------------
90  | 91  | 92  | 93  | 94  | 95  | 96  | 97  | 98  | 99

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
-------------------------------------------
0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
-------------------------------------------
0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
-------------------------------------------
0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
-------------------------------------------
0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
-------------------------------------------
0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
-------------------------------------------
0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
-------------------------------------------
0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
-------------------------------------------
0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
-------------------------------------------
0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
```

Figure 8. sample test result of the function

Input 58 in the serial monitor to set the goal node. The algorithm will start to iterate following the process stated and update the list. The following figure shows the checked node and final selected path. 1 is the node selected to move. 2 is node check but forfeited. Thus, the final path from grid 1 to grid 58 is displayed in the figure.



```
55
44
45
56
45
46
57
46
47
58
67
```

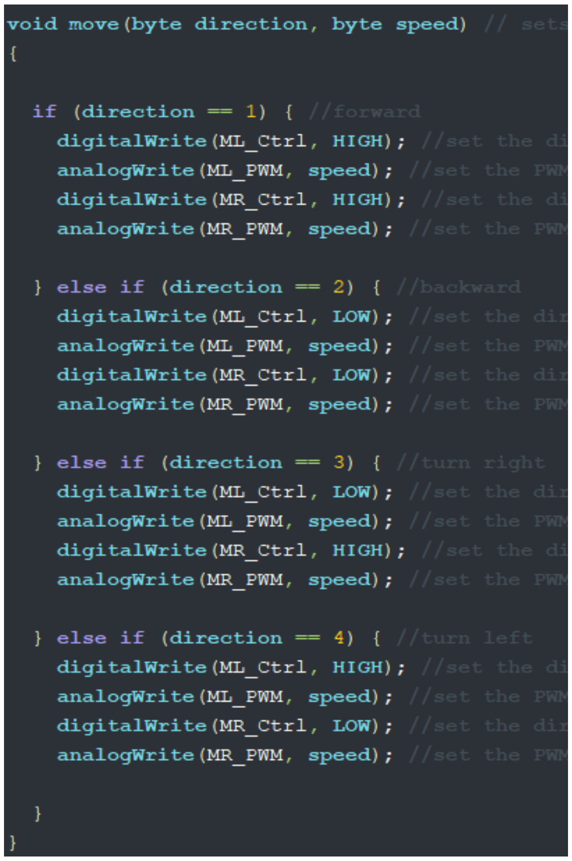
```
45
56
45
46
57
46
47
58
47
```

```
11
21
31
41
51
52
53
54
55
56
57
58
```

```
0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
---------------------------------------
0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
---------------------------------------
0 | 1 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0
---------------------------------------
0 | 1 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0
---------------------------------------
0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
---------------------------------------
0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0
---------------------------------------
0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
---------------------------------------
0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0
---------------------------------------
0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
---------------------------------------
0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
```

Figure 9. sample test result of the function

After the pathfinding process, the robot will self-direct itself to the designated node following the calculated path. As a simple prototype, it is only given four directions and turns ninety-degree angles. (Up, down, left, and right).



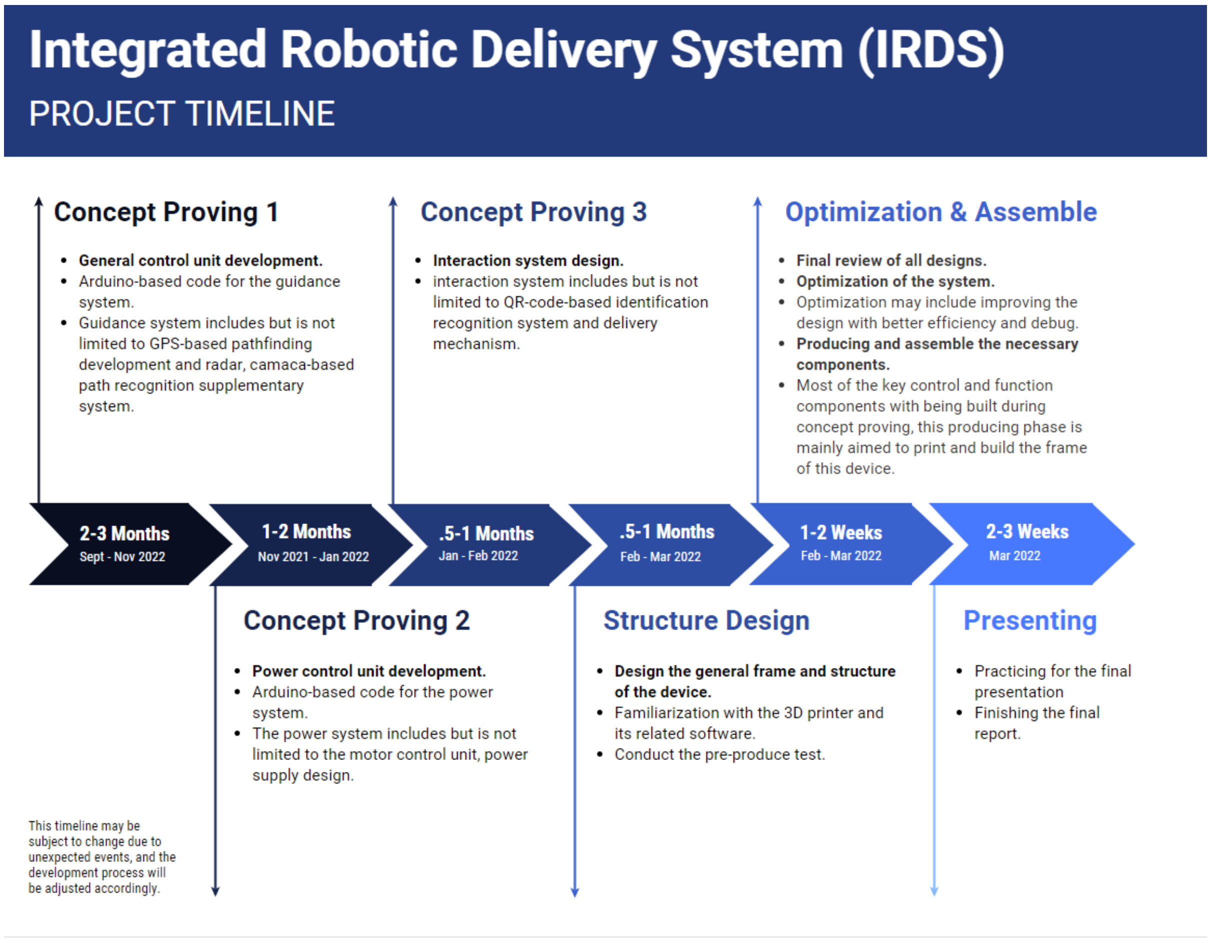
```
void move(byte direction, byte speed) // sets
{

  if (direction == 1) { //forward
    digitalWrite(ML_Ctrl, HIGH); //set the di
    analogWrite(ML_PWM, speed); //set the PWM
    digitalWrite(MR_Ctrl, HIGH); //set the di
    analogWrite(MR_PWM, speed); //set the PWM

  } else if (direction == 2) { //backward
    digitalWrite(ML_Ctrl, LOW); //set the dir
    analogWrite(ML_PWM, speed); //set the PWM
    digitalWrite(MR_Ctrl, LOW); //set the dir
    analogWrite(MR_PWM, speed); //set the PWM

  } else if (direction == 3) { //turn right
    digitalWrite(ML_Ctrl, LOW); //set the dir
    analogWrite(ML_PWM, speed); //set the PWM
    digitalWrite(MR_Ctrl, HIGH); //set the di
    analogWrite(MR_PWM, speed); //set the PWM

  } else if (direction == 4) { //turn left
    digitalWrite(ML_Ctrl, HIGH); //set the di
    analogWrite(ML_PWM, speed); //set the PWM
    digitalWrite(MR_Ctrl, LOW); //set the dir
    analogWrite(MR_PWM, speed); //set the PWM

  }
}
```

Figure 10. sample code for the design

based on the grid index stored in the list generated, representing the selected path, the robot will decide its direction based on the numerical relation between the current node and the next node.

TIMETABLE

According to our scheduled timeline, the progress has been kept on track and carried out smoothly. Referring to the above part of the report, the basic mechanical part and pathfinding function has been implemented. The robot can now navigate and move to a specified destination with an uploaded (10x10) map.



Integrated Robotic Delivery System (IRDS)

PROJECT TIMELINE

Concept Proving 1

- **General control unit development.**
- Arduino-based code for the guidance system.
- Guidance system includes but is not limited to GPS-based pathfinding development and radar, camaca-based path recognition supplementary system.

Concept Proving 3

- **Interaction system design.**
- interaction system includes but is not limited to QR-code-based identification recognition system and delivery mechanism.

Optimization & Assemble

- **Final review of all designs.**
- **Optimization of the system.**
- Optimization may include improving the design with better efficiency and debug.
- **Producing and assemble the necessary components.**
- Most of the key control and function components with being built during concept proving, this producing phase is mainly aimed to print and build the frame of this device.

2-3 Months
Sept - Nov 2022

1-2 Months
Nov 2021 - Jan 2022

.5-1 Months
Jan - Feb 2022

.5-1 Months
Feb - Mar 2022

1-2 Weeks
Feb - Mar 2022

2-3 Weeks
Mar 2022

Concept Proving 2

- **Power control unit development.**
- Arduino-based code for the power system.
- The power system includes but is not limited to the motor control unit, power supply design.

Structure Design

- **Design the general frame and structure of the device.**
- Familiarization with the 3D printer and its related software.
- Conduct the pre-produce test.

Presenting

- Practicing for the final presentation
- Finishing the final report.

This timeline may be subject to change due to unexpected events, and the development process will be adjusted accordingly.

Figure 11. Timeline & Milestones

To better refine the navigation function, we will need to test the robot with a more realistic map of a parking lot with several pickup parking spots. It is required that the robot can self-navigate to the destination with an instruction of the customer's parking spot.

With a complete "navigate-and-go" function, the rest of the work will be focused on the logic process of loading and delivery. It is assumed under our preset that the pickup service is a scheduled service so that at a specific time, the delivery volume of goods does not exceed 6.

Putting this into our design, we will set the robot to carry six cabins, each containing the product for one customer. Moreover, if the product is very large, for instance, a television or a sofa, human service is needed, which will be excluded from our design. Once the cabins are empty, the robot will automatically return to the loading area. For this case, we will write functions using Python to check if the cabins are empty, then call the "navigate-and-go" function with the instruction of the loading area address.

While loading products, a function will need to be implemented to match the product info with the cabin number and save the pairs of data into a data frame. While waiting for delivery instructions, the robot will pop up a window asking for a designated delivery parking spot. Once a customer arrives, and sends the parking spot number, the "navigate-and -go" function will be called to direct the robot to the customer.

Upon arriving at the delivery spot, the robot should receive the "arrived" signal and trigger another function that asks the customer to provide a QR code identification for the products. It will search the cabin number in the data frame with the product information provided. Then, send a signal to open the specific cabin.

RESULTS AND DISCUSSION (FUTURE PLAN)

For now, our Contactless Delivery Robot has achieved these functions:

Mobility: Now the robot could move forward or backward, do differential turning, and change its speed based on given commands.

Optimal routing selection based on A* algorithm: Now on a fixed map, the algorithm can avoid obstacles and find the closest path to a given destination.

Self-navigation: The robot can follow the path found by the algorithm, navigate itself to the destination.

When we were developing these functions, we encountered some problems, to solve them we redefined some of our original designs.

The first change is we decided to add a Raspberry Pi to the robot instead of only using Arduino boards because when we were developing the Optimal routing selection function, we found the

Arduino’s memory is too small to store a high-definition map, also Arduino’s computing speed is too slow. These problems would limit our robot’s field of use, so we decided to add a Raspberry Pi to the robot to handle all the computing and communicating work, letting the Arduino focus on mobility control.

The second change is we decided to add a touch screen to the robot and increase its interactivity. Since a Raspberry Pi was added, HDMI signal output becomes available. We decided to develop a GUI, which will ask clients to input their order details on the touch screen for identification, also the screen can display notifications to clients such as “Please pick up your item in container #1”. Adding a touch screen would make the human-computer interaction more intuitive.

The third change we want to do is to connect the robot to the Local Area Network via WIFI and receive data from a server. The server receives clients’ pickup request and automatically send clients’ parking lot number and order details to the robot, then the robot will find its way to the client. This function would make the management of large-scale robots much more efficient.

CONCLUSION

In conclusion, our development of the contactless delivery robot is progressing steadily. So far, we built the robot’s physical body with adequate mobility, but the accuracy on turning needs more calibration; We also added a Raspberry Pi to give the robot more computing power. With this extra computing power, we already completed the optimal routing and self-navigation function, adding an interactive user interface and network connection is our further target.

REFERENCE

[1] SYSC 4907 Proposal report. Contactless Delivery Robot.

[2] k. R. (keyestudio), "Keyestudio quick connectors Motor Drive Shield V2," [Online]. Available: https://www.dropbox.com/sh/258wg43g52r8msh/AAAR_I5CRR7dqaCrIEp6SpFGa?dl=0&preview=KS0435+Keyestudio+quick+connectors+Motor+Drive+Shield+V2.pdf.

[3] adafruit, [Online]. Available: https://www.adafruit.com/product/3777.

[4] pathfinding. [Online]. Available: https://en.wikipedia.org/wiki/A*_search_algorithm