Xi'an Jiaotong-Liverpool University

西交利物浦大學

Maze Solving Game-Based Learning Application Based on a Raspberry Pi-Controlled Robot

Author Zhiyong Liu

Module CSE305

Supervisor Dr. Jieming Ma

Date 13^{th} / April / 2019

Contents

Acknowledgement

Thanks to Dr. Jieming Ma for his guidance and support of this whole project. He provided all the equipments required for this project, which released the financial pressure. In each important step, he patiently provides timely information, which make me able to achieve the milestone in the development of this project. Working under his guidance equipped me with the necessary skills to make effective contributions to the research team, and laid the foundation for me to enter the next stage in my career. I would also like to thank Dr. Haining Liang for his professional feedback, which let me know how to improve in future academic research. In addition, I would like to thank Xi 'an Jiaotong-Liverpool University, China, which helps me to develop the key skills needed in my career path.

Abstract

As computer and mobile technologies advance, a amount of games for educational purposes have been used among learners of different levels, which is called game-based learning(GBL). It has become the best solution for soft skills learning when traditional learning is homiletic, expensive and difficult to implement. In this paper, a game-based learning application is developed for those people who wish to learning programming knowledge. The application allows user control a tracked mobile robot to get out of maze with their programming skills. The robot is built based on Raspberry Pi 3+ and configured with the Debian operating system. Running this application, tracked mobile robot can explore the maze and find a route out of the maze. In this application, Depth-First-Search algorithm is utilized, which is the core idea of algorithm to exit the maze. Furthermore, This application is modularized, which also supports users to modify the application for secondary development.

Key Words: game-based learning, maze solving, Depth-First-Search algorithm, tracked mobile robot.

Chapter 1

Introduction

Now more and more people are supposed to learn knowledge about programming. However, learning programming can sometimes become so boring that causes learners to give up halfway. As computer and mobile technologies advance, a large amount of games for educational purposes have been used among learners of different levels [1]. This form of learning is called game-based learning (GBL). It has become the best solution for soft skills learning when traditional learning tends to be homiletic, expensive and difficult to implement. Game-based learning aims to balance subjects with gameplay and players' ability to retain and apply subjects to the real world [2].

Games have been applied in education for a long time. In the middle ages, noblemen learned strategies of war by using ancient chess game. During the Civil War, volunteers from Rhode Island played *American Kriegsspiel*, which had originally been created in 1812 for training Prussian officers-of-war [3]. Until now, a wide range of game-based learning applications are used, such as exploring ancient history with video games, teaching empathy with video games [4]. Furthermore, one well-known example is the app *Playgrounds* developed by Apple [5]. This app provides abundant games for children to learn Swift interactive and fun. It drives children to solve the puzzles to master the basics using Swift. Compared with tradition way that facing with a programming textbook to study, children prefer to cost much time on playing and learning in the meantime through this way.

Maze solving problem has been researched for a long period in computer science. Computer hobbyists always enjoy finding new algorithms to solve the maze problem. The best-known rule for solving maze is the Wall Follower [6], which also known as either the right-hand rule or left-hand rule. It keeps one hand in contact with one wall of the maze, so that the solver doesn't get lost, and the solver can reach to another exit if it exists. Other algorithms such as Recursive algorithm [7], Pledge algorithm [8], and Trémaux's algorithm [9], were invented specially to deal with more special mazes, and each of them has their own strengths and weaknesses.

A maze can also be viewed as a tree or graph, therefore, some algorithms used in graph theory also have the ability to solve the maze solving problem [10]. One of them is Depth-First Search algorithm (DFS), which is used to traverse the tree or graph data structure. The algorithm starts at the root node and explores each branch as far as possible before backtracking [11].

In this project, a game-based learning application is developed for those people who wish to learning programming knowledge. It allows learners to utilize programming knowledge to control a tracked mobile robot to get out of maze. Furthermore, this application is also simple enough for learners to conduct secondary development and learners can utilize their own programming skills to design algorithms on this application for maze solving. The major component of the application is a tracked mobile robot, which uses a raspberry pi as the control core. It consists of one electronic compass module, one infrared sensors, and three ultrasonic sensors. The robot is also pre-configured with a maze solving program. It is developed in python language and an algorithm based on DFS algorithm designed in this project is used, which can enable the robot to get out of real maze successfully.

This dissertation is divided into several parts. Chapter 2 gives the related background about the software and hardware, as well as the technology used in this application. Chapter 3 indicates the design of this project. Chapter 4 shows the implementation of design introduced in Chapter 3. Chapter 5 gives the experiments of this maze solving game-based application and analyzes the results. Chapter 6 and Chapter 7 mention learning points and professional issues of this project respectively.

Chapter 2

Background

2.1 Tracked Mobile Robot

The tracked mobile robot is 300 millimeters long, 230 millimeters wide, 280 millimeters high. It has two crawler wheels and configured with two infrared sensors and three ultrasonic sensors in this project. The infrared sensors are used to detect the maze exit where the color of the ground is different from that of other places. The ultrasonic sensors are used to detect the obstacle.

The body of the robot is made of steel plates with some holes on them. These holes support the other elements of robot stick to the body. Furthermore, two electric units are attached to the

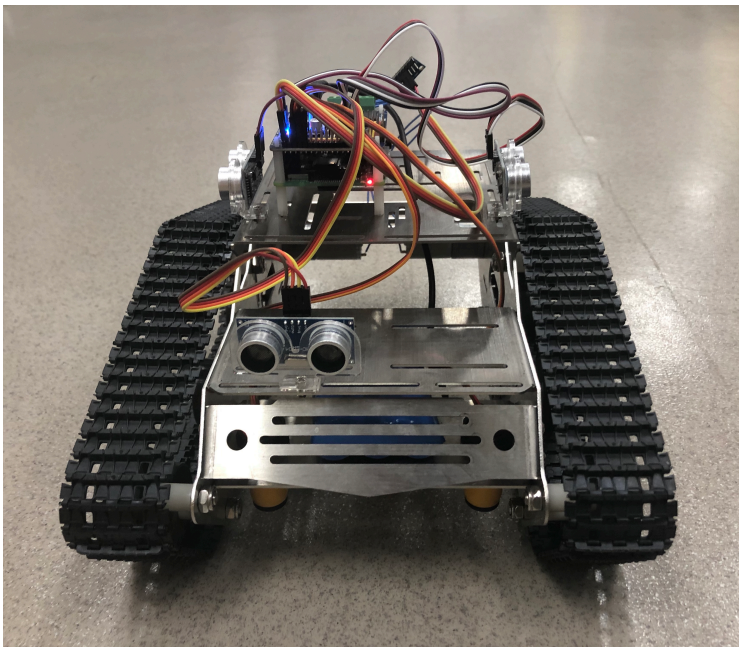

Fig. 2.1: Tracked Robot

body, which allows robot to move, such as move forward and turn. The robot uses one Raspberry Pi 3 Model B Plus Rev 1.3 as its processor. Raspberry Pi is powered by a 2200mAh, 12v, 8A lithium battery pack, its board output voltage is 5v and sensors are connected and powered with the dupont threads connected to robot. The raspberry pi on the robot supports remote connection

application to control [12]. Through the connection, researcher can upload the code to raspberry and run related code to make the robot perform actions accordingly.

2.2 Secure Shell Protocol

SSH is a cryptographic network protocol based on application layer in the seven-layer computer networking OSI model [13]. It is a reliable protocol aimed to provide security for logining remotely and other network services at present. Many applications provide remote login and execution through command-line. Furthermore, any service in networking can be made secure with SSH [14].

SSH uses a secure channel over an insecure network and adapts the the client-server architecture. The are two major versions in SSH protocol, referred to as SSH-1 and SSH-2 [15]. Because SSH is a protocol built on application layer, the standard TCP (Transmission Control Protocol) port for SSH is 22. SSH is generally used in Unix-like operating systems, but it can also be used in other types of systems, such as Microsoft Windows [16].

In this project, researcher use terminal on the remote mac to control the Raspberry Pi in the same local area network. Fig. 2.2 shows the procedure for a remote connection to Raspberry Pi on the tracked robot. The command ssh pi@192.168.1.1 is used to connect the remote host,

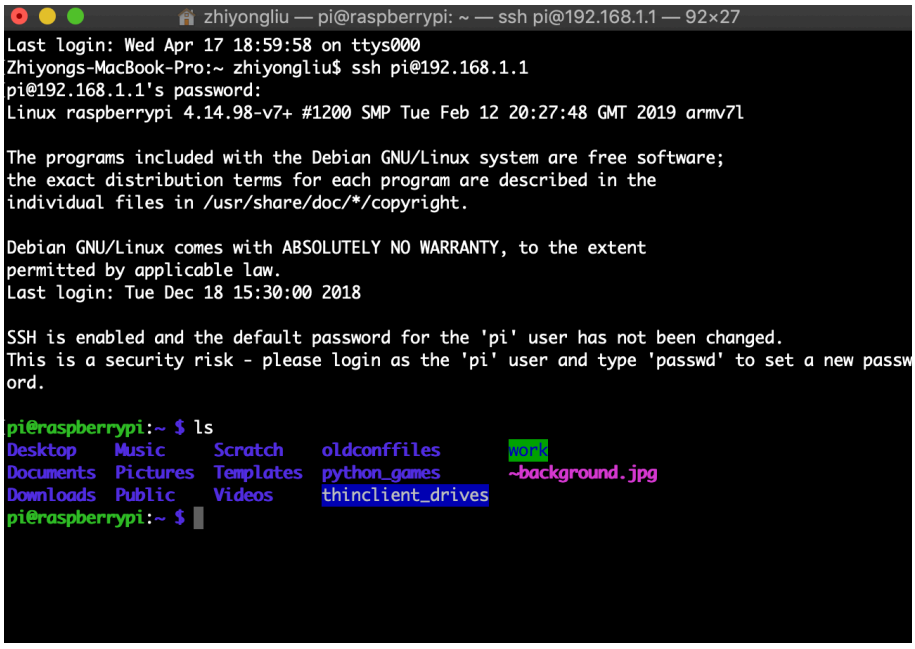

Fig. 2.2: SSH

referred to the Raspberry Pi in this project. the first parameter *pi* referred to username used to login in, and the second parameter is the IP address of the remote host, referred to *192.168.1.1* here. Then the command ls shows the files in the current directory. Fig. 2.2 denotes the remote connection is performed well and researcher successfully performs remote execution on Raspberry

Pi. Then, depending on Linux system knowledge, researcher can use Linux commands to operate the Raspbian system on Raspberry Pi.

2.3 Raspberry Pi 3 B+

The Raspberry Pi is one kind of small single-board computers. They are developed by the Raspberry Pi Foundation in the United Kingdom, which aims to facilitate education of basic computer science in schools and in developing countries. Its hardware has evolved over several versions, such as Model A, Model A+, Model B, Model B+ and Zero. The features of these versions differ in memory capacity and peripheral-device support.

In this project, the version of Raspberry Pi used is Raspberry Pi 3 B+. It has 1GB LPDDR2 SDRAM, and supports 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE [17]. The Raspberry Pi 3 B+ has 4 USB 2.0 ports are provided and it has a Micro SD port which aims to load the operating system and store data. The Raspberry Pi in this project used an 8G micro SD card, it is formatted by the *SD Card Formatter*, which is a program that provides instant access to all formats about memory cards [18]. Then, Researcher download the Raspbian

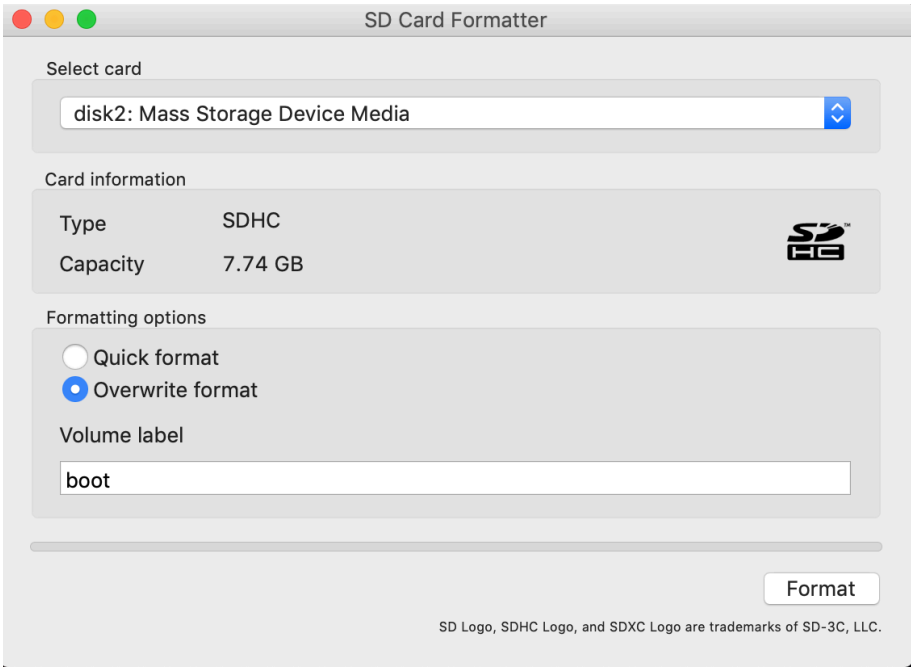

Fig. 2.3: The Interface of SD Formatter

system image from the Raspberry pie website official website. The system image is burned into SD card by the tool *BalenaEtcher*, which is an image burn tool on the mac platform. The interface of SD Card Formatter and BalenaEtcher are shown as Fig. 2.3 and Fig. 2.4. When the burned SD card is plugged into Raspberry Pi and the power is turn on, the raspberry pi starts to load the operating system and becomes available for researcher to use.

The Raspberry Pi 3 B+ has extended 40-pin GPIO header for developers to use. The pinout

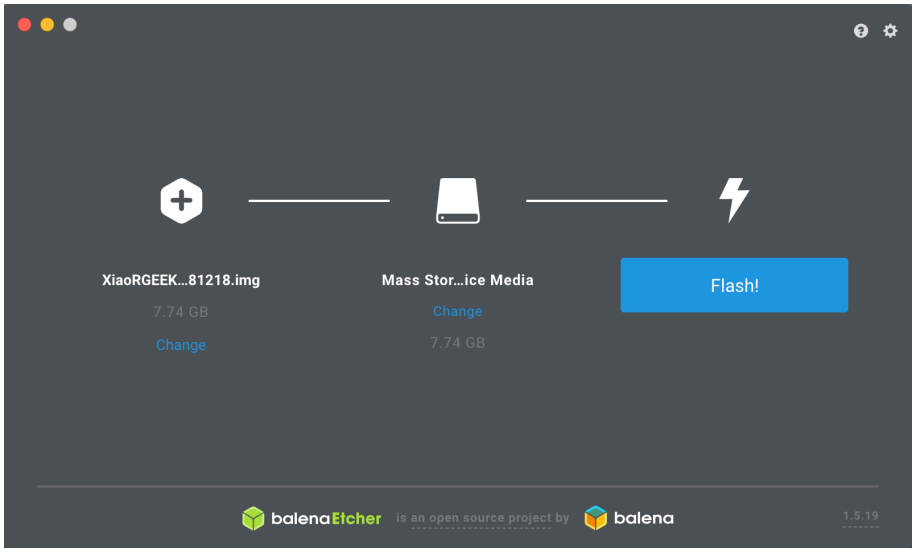

Fig. 2.4: The Interface of BalenaEtcher

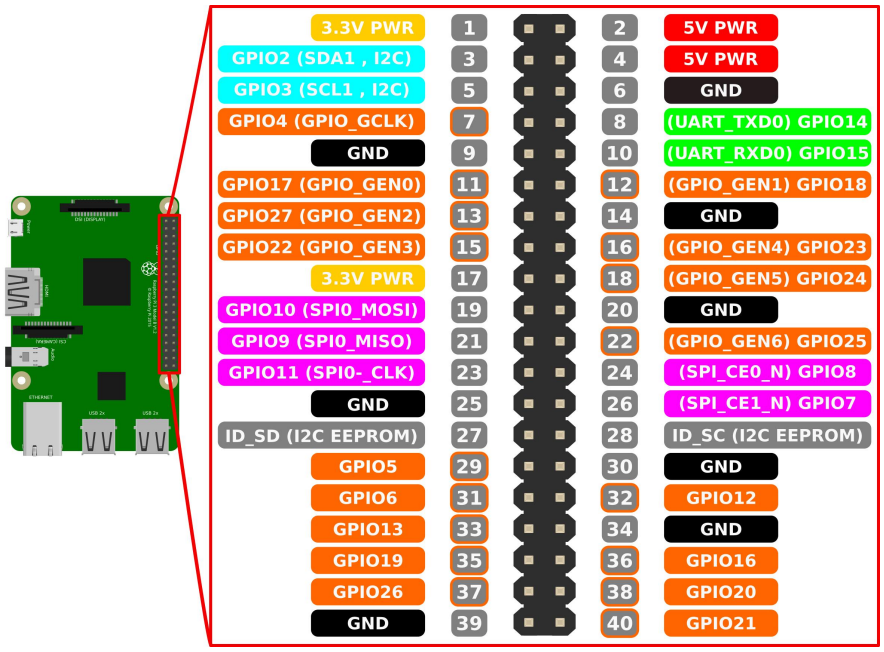

Fig. 2.5: The Pinout of Raspberry Pi 3 B+

is shown as Fig. 2.5. GPIO means universal input/output ports, which are pins that can be used to output high and low levels or to read in the state of a pin, referred to high or low. GPIO is a relatively concept. Users can interact with the hardware through these GPIO pts, control the hardware work (such as LED, buzzer,etc) and read the hardware status signal (such as interrupt signal).

GPIO can be used in three modes:

1. Input
2. Output
3. URAT Interface

Input is the default mode, in which the Raspberry Pi get the input from the connected device via GPIO. In the output mode, it delivers data to the connected device. Furthermore, GPIO can also be configured as an URAT interface, which allow user to define custom advertising packets [19].

2.4 Sensor

2.4.1 HC-SR04 Ultrasonic ranging module

Fig. 2.6: HC-SR04

HC-SR04 is an Ultrasonic ranging module, which provides the function of 2cm to 400 cm non-contact measurement. The ranging accuracy is able to reach to 3mm and its effectual angle can less than 15°. The modules include receiver, control circuit and ultrasonic transmitters. The basic principle of its work is:

1. Using IO trigger for at least 10us high level signal
2. The Module automatically sends eight 40 kHz and detect whether there is a pulse signal back.

3. IF the signal back, through high level, time of high output IO duration is the time from sending ultrasonic to returning.

4. Test distance = (high level time * velocity of sound (340M/S) / 2

The wire connecting direct is shown as following:

- 5V Supply
- Trigger Pulse Input
- Echo Pulse Output
- 0V Ground

The specification of HC-SR04 is shown in Table 2.1.

Table 2.1: The specification of HC-SR04 Ultrasonic ranging module

Working Voltage	DC 5V
Working Current	15mA
Working Frequency	40Hz
Max Range	4m
Min Range	2cm
Measuring Angle	15 degree
Trigger Input Signal	10μS TTL pulse
Echo Output Signal	Input TTL lever signal and the range in proportion
Dimension	45 * 20 * 15mm

2.4.2 E18-D80NK Infrared Distance Ranging Sensor

E18-D80NK is a infrared sensor with a long distance detection and has less interference by visible light. It adjusted to sense objects over a range of 3-80 cm. It is very cheap, and simple to use. It consists of an infrared transmitter and receiver pair in a module. The infrared transmitter emits infrared light, and the infrared receiver detects the infrared light reflected by the object [20].

The principle of transmitter and receiver are shown in Fig. 2.8 and Fig. 2.9. The specification of E18-D80NK Infrared Distance Ranging Sensor is shown in Table 2.2.

Fig. 2.7: E18-D80NK Infrared Distance Ranging Sensor

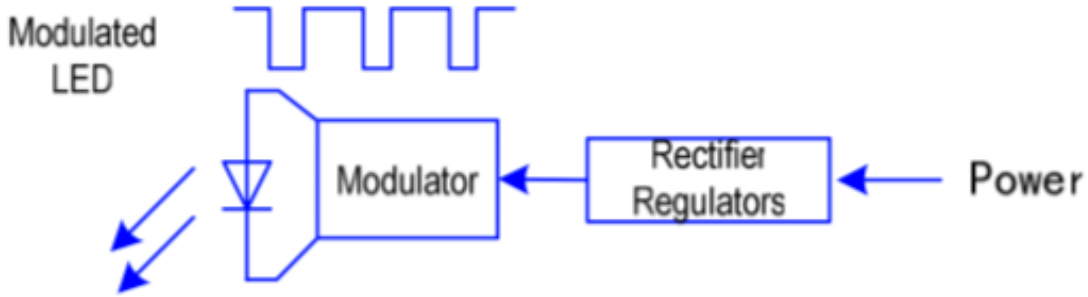

Fig. 2.8: transmitter

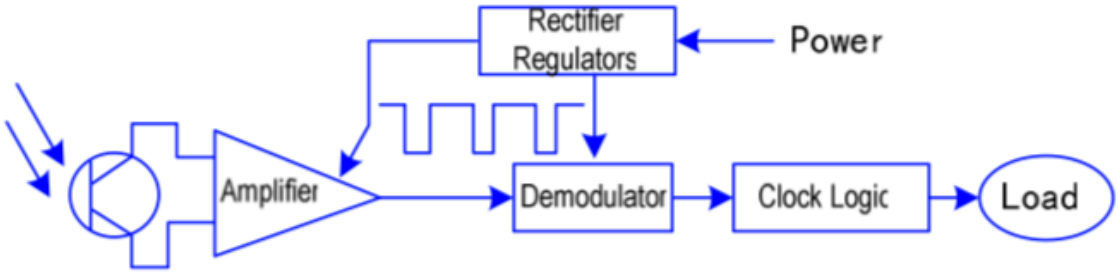

Fig. 2.9: receiver

Table 2.2: The specification of E18-D80NK Infrared Distance Ranging Sensor

Sensing range	3-80cm adjustable
Sensing object	Translucency, opaque
Working Voltage	DC5V
Working Current	100mA
Output operation	Normally open(O)
Output	DC three-wire system(NPN)
Diameter	18mm
Length	45mm
Guard mode	Reverse polarity protection
Ambient temperature	-25-70°C

In this project, the purpose of infrared sensor is to detect the destination of maze, whose ground color is black. The other area of maze is set to another color. The principle of detection is that the infrared sensor utilizes light reflection. Ordinary colors will reflect light black, but black is the strongest color to absorb light so that there is no set signal. With the infrared sensors, the robot detects the arrival of the endpoint of maze by detecting the presence or absence of reflected light.

2.5 Depth-First-Search Algorithm

Depth-first search (DFS) is algorithm designed to traverse or search graph or tree data structures. It was invented by *John Edward Hopcroft* and *Robert Endre Tarjan* and shared the Turing award in 1986 [21]. Depth-first search is a canonical algorithm used in graph theory. For a graph, it can generate the correspondent topological sorting table.

The main principle of algorithm is that the traverse starts at the root node in the tree or graph (selecting arbitrary node as the root node when algorithm is used in graph). Then the exploration moves far as possible along each branch. Only after one branch is explored, exploration backtracks and performs on another branch [22].

The pseudocode of Depth-First-Search Algorithm is shown as following.

Input: A graph G and a vertex n of G.

Output: All reachable vertices from n are marked as discovered.

The first thing to choose a node to start and the adjacent nodes of this node are pushed into the stack S. Then a node is popped from stack S and the algorithm chooses a node to visit next.

Algorithm 1 Depth-First-Search algorithm

```
1:  procedure DFS(G, n)
2:      S.push(n)                                   ▷ Inserting s in stack
3:      label n as visited
4:      while S is non-empty do
5:          v = S.top()
6:          S.pop()
7:          for all neighbours x of v in graph G do
8:              if x is not visted then
9:                  S.push(x)
10:                 label x as visited
11:             end if
12:         end for
13:     end while
14: end procedure
```

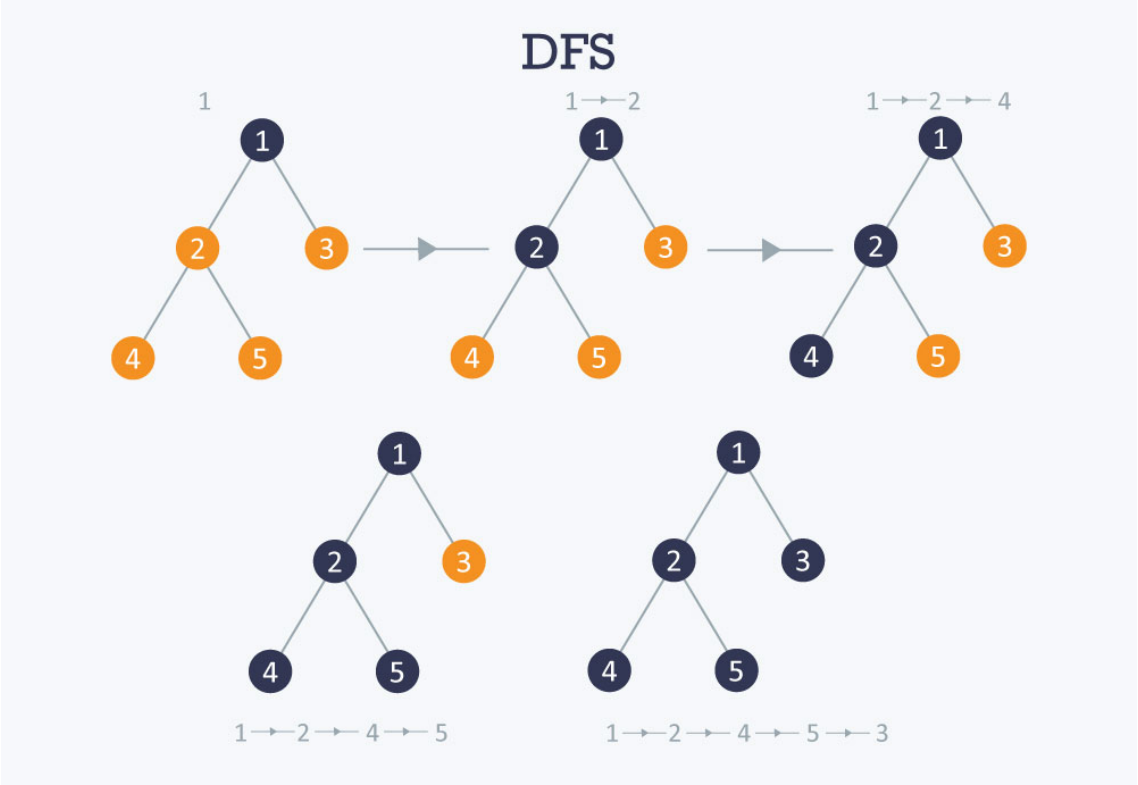

Fig. 2.10: The process of DFS

Besides, all its adjacent nodes are pushed into the stack S. This process is repeated until the stack S is empty. The nodes that are visited must be labeled, which can provide it from visiting the same node again.

Fig. 2.12 shows how DFS works in a simple tree. DFS algorithm starts at the node 1. It visits the node 1 and marks it as visited. Then, node 2 and node 3 are adjacent to node 1. The algorithm finds neither node 2 and node 3 are marked. It selects node 2 to visit and mark. At the node 2, it finds two nodes adjacent to it are not marked. Therefore, it chooses node 4 to visit and mark. After visiting the node 4, there are no nodes adjacent to node 4 which can be visited. It backtracks to the last node, node 2. At the node 2, it finds that the node 5 is adjacent to node 2 but not marked. Therefore, this algorithm then visits node 5 and marks it. In the same way, there are no unmarked nodes adjacent to node 5, so it backtracks to node 2. Because node 2 also does not has unmarked adjacent nodes in the current time, the algorithm continue to backtrack to node 1. At the node 1, there is one unmarked adjacent node left, which is node 3. Therefore, node 3 is chosen to visit and mark. By this point, all nodes in this tree have been visited.

2.6 General Purpose Input/Output

GPIO stands for General Purpose Input/Output. It is a type of uncommitted digital signal pin on an integrated circuit or an electronic circuit board. Its function is not specific and controlled by user at run time. They can be used to output high or low levels. Furthermore, user can use them to read the status of pin, referred to high or low level [23].

To control GPIO, the convenient method is to use some related library. For example, Debian operating system in Raspberry Pi has integrated the RPI.GPIO package. This package provides a class to control the GPIO on a Raspberry Pi. It allows developer use code to access GPIO, such as reading status, control output [24].

In python script on Raspberry Pi, the following statement can import the library RPI.GPIO.

```
1 import RPi.GPIO as GPIO
```

Once imported, the functions of GPIO module can be used. In RPI, two types of GPIO pin numbering scheme are supported, which are BOARD and BCM [25]. The GPIO BOARD scheme specifies that the pins are referred by the number of the pin. these are numbers printed on the board which shown inside the red rectangle in Fig. 2.11 and Fig. 2.12. The GPIO BCM scheme specifies that the pins are referred by the ”Broadcom SOC channel” number [26], these are the numbers after ”GPIO” outside of red rectangle in Fig. 2.12 and Fig. 2.13 [27].

However, the BCM numbers are different between versions of Raspberry Pi. For example, Fig. 2.11 and Fig. 2.12 shows the pin number of Revision 1.0 and 2.0 of Raspberry Pi 1 Model B. The GPIO 0 pin in Revision 1.0 is corresponding to GPIO 2 pin in Revision 2.0. These two pins are in same position on board but are numbered differently in BCM scheme varies to different versions. For different versions of the raspberry pie, the script files written may not be generic. Therefore, if more than one Raspberry Pi are used in a project, it may be safer to use the BOARD scheme.

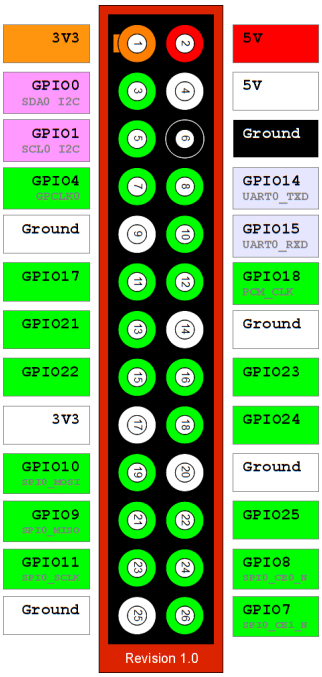

Fig. 2.11: Pi 1 Model B Revision 1.0

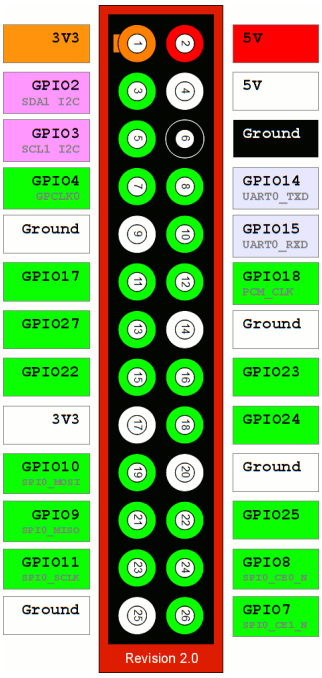

Fig. 2.12: Pi 1 Model B Revision 2.0

In python script, the numbering scheme can be indicated by the method 'GPIO.setmode()'.

Set numbering scheme to BOARD scheme:

```
# Set numbering scheme to BOARD scheme
GPIO.setmode(GPIO.BOARD)
```

Set numbering scheme to BCM scheme:

```
# Set numbering scheme to BCM scheme
GPIO.setmode(GPIO.BCM)
```

Before using a pin, this pin needs to be set as input or output. The python code which config a pin is shown as follow (The channel refers to the number of pin):

```
# Configue the pin as input.
GPIO.setup(channel, GPIO.IN)

# Configue the pin as output.
GPIO.setup(channel, GPIO.OUT)

# Set the default value for the pin.
GPIO.setup(channel, GPIO.OUT, initial=GPIO.HIGH)
```

To light an LED, or to power a device, what to do is to give current and voltage to that device. This step is extremely simple and just set the output state of the related pin connected to that device as following python code. The state can be set to 0, GPIO.LOW, False, 1, GPIO.HIGH or True.

```
# Set the output state of pin.
GPIO.output(channel, state)
```

We also often need to read the input state of the pin:

```
# Read the input state of pin.
GPIO.input(channel)
```

The following code is to use GPIO on Raspberry Pi to light up a LED. In this example, led is connected to the pin 11 on Raspberry Pi through dupont wire.

```
# import RPI.GPIO
import RPi.GPIO as GPIO
import time

# Set the numbering scheme
RPi.GPIO.setmode(GPIO.BOARD)

# Set the pin number
channel = 11

# Set pin 11 as output scheme
RPi.GPIO.setup(11, RPi.GPIO.OUT)

while True
    # Set the state of the pin to a high level and the LED lights up
    GPIO.output(channel, 1)
    # Program sleep 1 second, let the LED light for 1 second
    time.sleep(1)
    # Set the pin state to low and the LED will go off
    GPIO.output(channel, 0)
```

```
21          # Program sleep 1 second, let the LED off for 1 second
22          time.sleep(1)
23
24      # Release all the resources
25      GPIO.cleanup()
```

The above example makes the led light for one second and being off for another second, which runs again and agin. This led is connected to pin 11 on Raspberry Pi and its state is controlled by the output level of pin 11.

Chapter 3

Design

3.1 Design Methodology

This development of this project adopts incremental model. The product is designed, implemented and tested incrementally, until the project is completed. The product consists of multiple components, each of which is designed and built separately. The increments refer to a series of releases, and each increment will provides more functionalities to users [28]. After the first increment, a version which can be used by the users is delivered. Then, the plan for next increment is developed based on user feedback. This process will continues until the complete product is delivered [29].

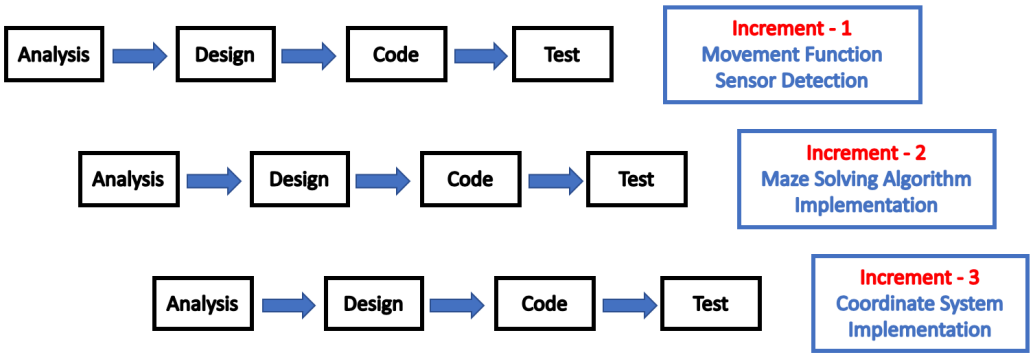

Fig. 3.1: Increment model

The general process of increment model is shown in Fig. 3.1. The Increment-1 about project delivered are movement function and sensor detection function. After the first increment delivered, robot can perform actions such as move forward, turn right, turn left and turn around. Furthermore, the ultrasonic sensors and infrared sensors configured on robot can be performed well. Robot can use ultrasonic sensors to detect available directions to move. It can also use infrared sensors to detect the color of ground, aimed to determine if robot reaches the end. The Increment-2 about project delivered is maze solving algorithm implementation. After the second increment delivered, robot implements a maze solving algorithm and can explore the maze. It performs actions according to situations in maze by the maze solving algorithm and get out of maze successfully.

The Increment-3 about project delivered is coordinate system function. After the third increment delivered, a coordinate system is built. Robot calculates its position dynamically during the process of movement. The path of robot can be recorded and the path from start to end can be output after getting out of maze.

3.2 Data Structure

3.2.1 Stack

The maze solving algorithm developed in this project uses stack to store the current path of robot and record the coordinate positions of crosses. Stack is chosen because its property of First-In Last-Out (FILO) [30]. FILO refers that the first element put in the stack will be the last element to be popped. With this property, it is convenient for the robot to return to the place which it has visited. For example, robot wants to return back to the position it visited last time. What need to do is to pop the top element in the path stack and the returned position is just the position visited last time. When the robot reaches the destination, the position list popped by the stack is exactly the path from end position to start position.

Fig. 3.2: Path stack

3.2.2 Tree Structure

The principle of tree structure is utilized in this project. The whole maze can be analogized to a tree. For example, the cross refers to the position in maze which has more than two directions to move. This can be analogized to the node which have multiple branches in tree structure. In this situation, the start position is analogized to the root node in tree structure. The position which does not belongs to cross in maze can be analogize to the point of edge in tree. The Fig. 3.3

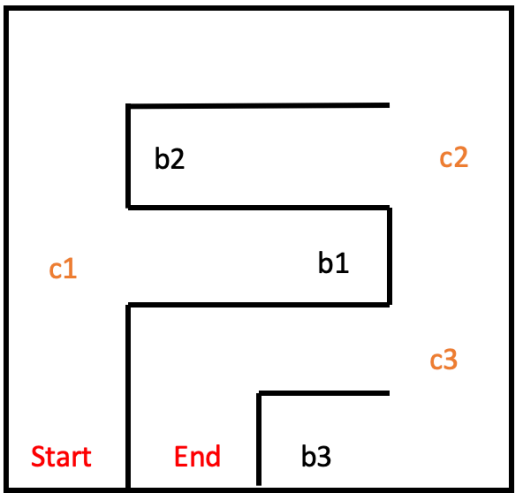

Fig. 3.3: Simple maze

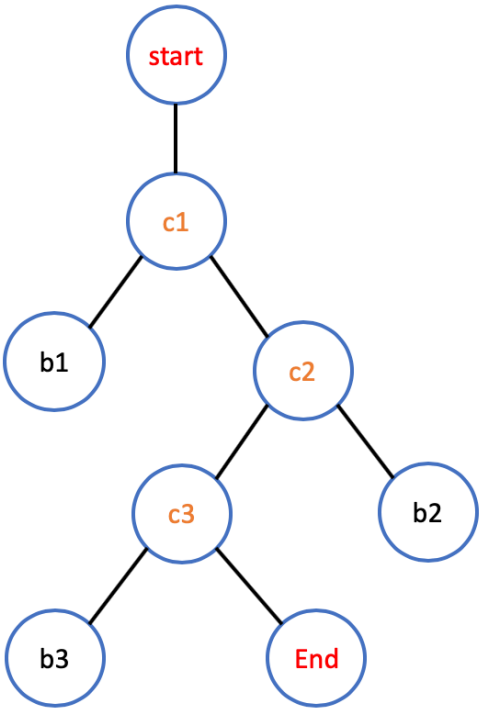

Fig. 3.4: Tree

shows a simple maze and the Fig. 3.4 is the tree structure analogize to this maze. The cross c1, c2, c3 in maze are corresponding to node c1, c2, c3 in tree. The blind alley b1, b2, b3 in maze are corresponding to node b1, b2, b3 in tree. By this method, the process of robot exploring maze is transformed into the process of traverse tree structure, and the algorithms to traverse tree structure can be utilized in maze solving problem.

3.3 Coordinate System

In this project, the maze explored by robot is a real environment, not a computer simulated environment. Therefore, a coordinate system is needed to be built up. This coordinate system can assign each position a coordinate. Meanwhile, the robot is also assigned a coordinate. When robot is exploring the maze, its coordinate is calculated dynamically. Therefore, every position which robot arrived are recorded in the form of coordinates. After robot arriving at the end, a path from start to end can be output in the form of coordinates.

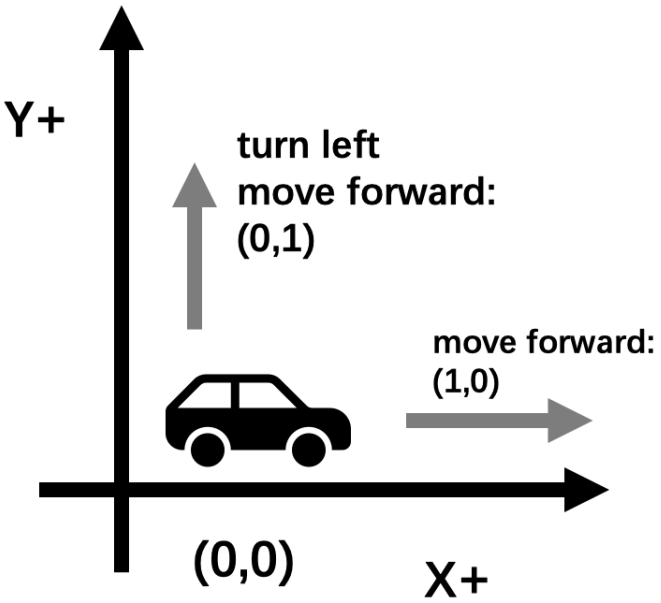

Fig. 3.5: Coordinate System

In this project, Cartesian coordinates is selected [31], and initial direction of robot is set to X positive axis. The start is set to (0,0). In Fig. 3.5, because the current direction is X+, if robot moves forward, its coordinate turn into (1,0). However, if robot turns left first, its current direction turns into Y+ from X+. At this time, robot then moves forward and its coordinate turn into (0,1) rather than (1,0).

Furthermore, some additional problem needs to be considered. For example, A mechanism is required to calculate and record the change of direction. In the following part in this section, a direction calculation mechanism designed in this project is introduced. In the process of exploring

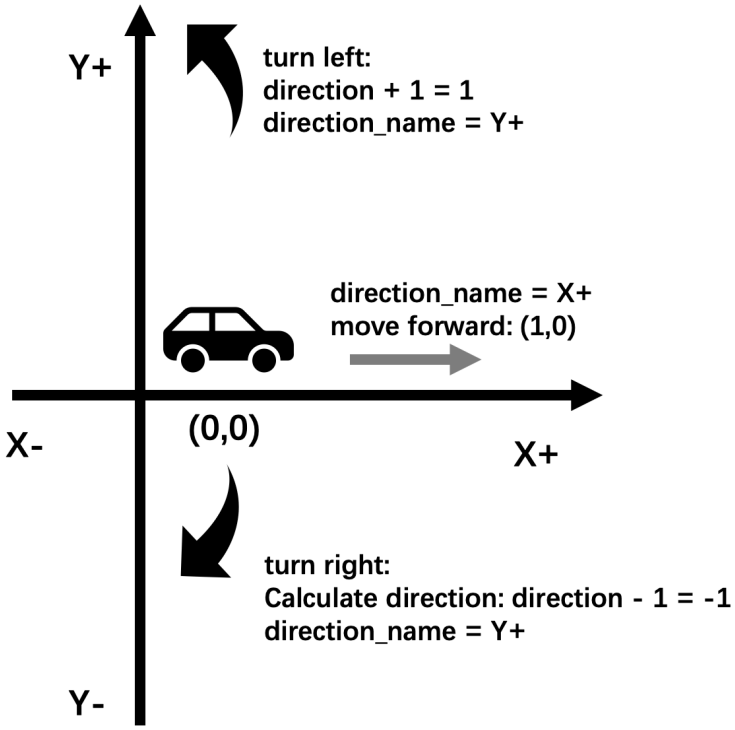

Fig. 3.6: Direction Calculation Mechanism

maze, robot could be in four possible directions, which are X+, Y+, X-, Y-.

In this project, two variables are used:

- **direction**: The range of $direction$ is [-4,4].
  $direction$ variable is used to calculate the current of direction of robot. When robot turns left, the value of $direction$ adds 1. On the contrary, when robot turns right, the value of $direction$ minus 1.

- **direction_name**: The value of $direction\_name$: X+, Y+, X-, Y-
  $direction\_name$ is used to calculate the direction of robot by the value of $direction$. For example, if the current value of $direction$ is 1, the value of $direction_name$ is set to Y+, and the coordinate calculation of robot is performed correspondingly according to the value of $direction\_name$.

In this designed direction calculation mechanism, each direction corresponds to a numerical value, which refers to $direction$.

- 0 $\rightarrow$ X+
- -2, 2 $\rightarrow$ X-
- 1, -3 $\rightarrow$ Y+

- -1, 3 $\rightarrow$ Y-
- -4, 4 resets to 0:
  When $direction$ equals to -4 or 4, it means robot has rotated 360 degrees accumulatively. Therefore, the direction reset to 0 (initial direction of robot).

The whole procedures of calculating direction is summarized as:

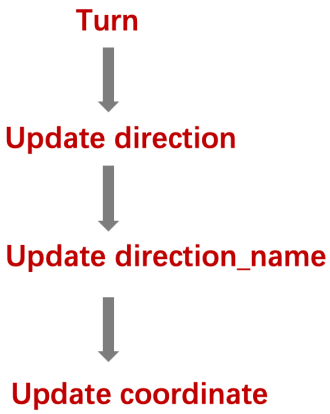

Fig. 3.7: Direction Calculation Mechanism

We will use example in Fig. 3.6 to show the operation process of this mechanism. In Fig. 3.6, the current coordinate of robot is (0,0). Its $direction$ is 0 and $direction\_name$ is X+.

At this time, robot turns left $\rightarrow$ $direction + 1$, which is equal to 1. Because $direction = 1$, then $direction\_name = Y+$, When robot moves forward in the direction of Y+. Due to $direction\_name = Y+$, this mechanism detects the value of $direction\_name$ and adds 1 to the value of Y. Therefore, the coordinate of robot turns into (0,1) from (0,0).

Through this coordinate calculate mechanism, the coordinate of robot is calculated and updated dynamically and accurately.

3.4 Maze Solving Algorithm

The designed maze solving algorithm (MSA) in this project use the recursive model [32]. In each recursive step, it will detect if current position is end. If current position is end, this algorithm executes completely. If current position is not end, robot will perform actions according to the type of current position. If it is a cross, this position is recorded in cross stack and marked as a cross. Then, robot will choose one available direction to move.

If current position is not a cross, there is only one possible available direction in the left, front, right. Robot chooses the sole direction to move. If all these three directions are not available to move, robot chooses the opposite direction to move and turn around.

If current position is a cross, it means there are at least two directions available to move in the left, front, right. Robot only chooses one of these available directions to move. The priority

principle of direction selection is left, front and right.

After choosing the moving direction, the new position is taken as the argument and call the MSA again. In this process, MSA will guide robot to explore as deep as it can until the current path of exploration is not available. MSA will explore another path from last cross. The whole process is repeated until the robot reaches the end.

The pseudocode of Maze Solving Algorithm (MSA) is shown below:

Algorithm 2 Maze Solving Algorithm

```
1: procedure MSA(position)
2:     current_path.push(position)          ▷ Inserting current postion into current_path
3:     if position is end then
4:         return current_path
5:     else
6:         if position is a cross then
7:             if postion in cross_stack then
8:                 pop current_path to the last occurrence of position
9:             else
10:                push postion in cross_stack
11:            end if
12:            new_position = available_position      ▷ new position search order: left, front,
    right, or turn around
13:            move to new_position
14:        end if
15:    end if
16: end procedure
```

The Fig. 3.8 shows the flow chart of MSA, which denotes the process of Maze Solving Algorithm. There is one most important part in this algorithm. This algorithm uses a stack to store crosses while robot is traversing the maze. The cross refers to the position which has more than one direction available to move. When robot moves to a new position, the robot configured with this algorithm will first test whether this position is a cross. In these three directions: left hand, front and right hand, if there is one more direction for robot to move, this position is regarded as a cross. Then this position is checked whether it exists in the stack cross. If this position does not exist in the stack cross, this position is regarded as a new cross and pushed into the cross stack, which recorded that the robot has reached this position once.

However, if this position exists in the stack cross, it means that robot has reached this position once in the previous time. Then, the stack current_path will be popped to the last occurrence of this position. It is because the path from the last time robot got to this position to this time is redundant, which is not regarded as one part of the final path from start to end.

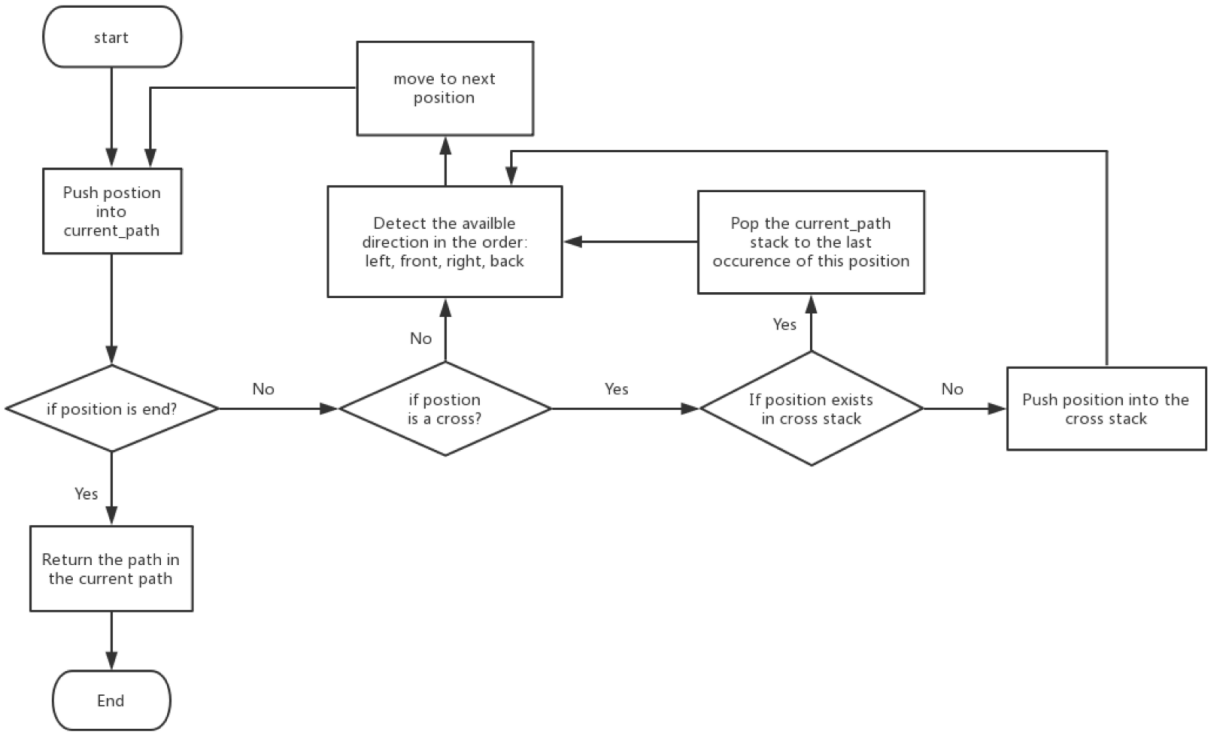

Fig. 3.8: Flow chart of MSA

In the Fig. 3.9, it shows an example that how MSA used in a maze. The MSA can obtain the path from start to end. The red circle with the number denotes the cross encountered when robot is traversing the maze. Due to reaching the same position for multiple times, the path denoted by purple line has been popped from the stack current_path. Therefore, the purple part is not regard as the part of the final path. The blue path denotes the final path from start to end generated by MSA.

The following steps are the process of robot exploring the maze in Fig. 3.6 using MSA.

1. Starts
2. Keep moving
3. Arrives at the **cross 1**
4. Detect current position: cross $\rightarrow$ visit **first time**, push into cross stack $\rightarrow$ two directions available: **font, right**
5. According to priority principle: left $>$ front $>$ right $\rightarrow$ selects moving direction: **front**
6. Keep moving
7. Arrives at the **cross 2**
8. Detect current position: cross $\rightarrow$ visit **first time**, push into cross stack $\rightarrow$ two directions available: **left, front**

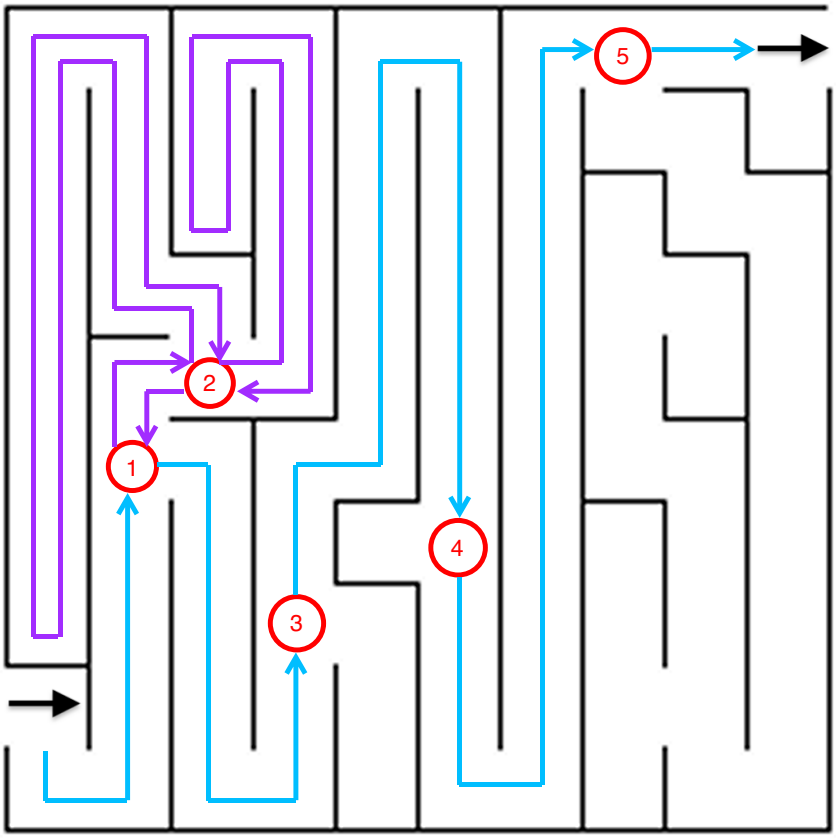

Fig. 3.9: Use MSA in a simple maze

9. According to priority principle: left > front > right → selects moving direction: **left**
10. Keep moving
11. No available way, turn around
12. Keep moving
13. Arrives at the **cross 2**
14. Detect current position: cross → visited **before**, pop path stack to last occurrence → two directions available: **left, right**
15. According to priority principle: left > front > right → selects moving direction: **left**
16. Keep moving
17. No available way, turn around
18. Keep moving
19. Arrives at the **cross 2**
20. Detect current position: cross → visited **before**, pop path stack to last occurrence → two directions available: **front, right**
21. According to priority principle: left > front > right → selects moving direction: **front**
22. Keep moving
23. Arrives at the **cross 1**
24. Detect current position: cross → visited **before**, pop path stack to last occurrence → two directions available: **front, left**
25. According to priority principle: left > front > right → selects moving direction: **left**
26. Keep moving
27. Arrives at the **cross 3**
28. Detect current position: cross → visit **first time**, push into cross stack → two directions available: **front, right**
29. According to priority principle: left > front > right → selects moving direction: **front**
30. Keep moving
31. Arrives at the **cross 4**

32. Detect current position: cross → visit **first time**, push into cross stack → two directions available: **front, right**

33. According to priority principle: left > front > right → selects moving direction: **front**

34. Keep moving

35. Arrives at the **cross 5**

36. Detect current position: cross → visit **first time**, push into cross stack → two directions available: **front, right**

37. According to priority principle: left > front > right → selects moving direction: **front**

38. Keep moving

39. Arrives at the end

40. Detects current position: **end**

41. Terminate

Chapter 4

Implementation

To implement the expected goals of this project, a Raspberry Pi 3 B+ is used in the development. The Raspbian is configured with python environment in advance, which allows python script to operate. Additionally, ultrasonic sensors (HC-SR04) were used to implement the wall detection. Infrared sensors (E18-D80NK) were used to implement the ground color detection. The controlling program was written in python and runs in Raspbian.

All sensor functions are implemented in the *Sensor.py*. In *Sensor.py*, we first import some module required and set the GPIO pin numbering to BCM.

```
1  # coding:utf-8
2  from socket import *
3  from time import ctime
4  import binascii
5  import RPi.GPIO as GPIO
6  import time
7  import threading
8
9  GPIO.setmode(GPIO.BCM)
10 GPIO.setwarnings(False)
```

The implementation of corresponding functions of sensors are introduced in the following sections.

4.1 Obstacle Detection

As traversing the maze, robot needs to have the ability to detect the existence of wall. In this project, three HC-SR04 Ultrasonic ranging modules were used. The installation illustration is shown in Fig. 4.1. These three ultrasonic sensors were installed in the different sides of robot. In Fig. 4.1, the black solid arrow denotes the moving direction of robot. One ultrasonic sensor is installed on the front side. Another one is installed on the left side and the last one is installed on the right side. The positions of these three ultrasonic sensors are marked by red rectangle in real graph in Fig. 4.1.

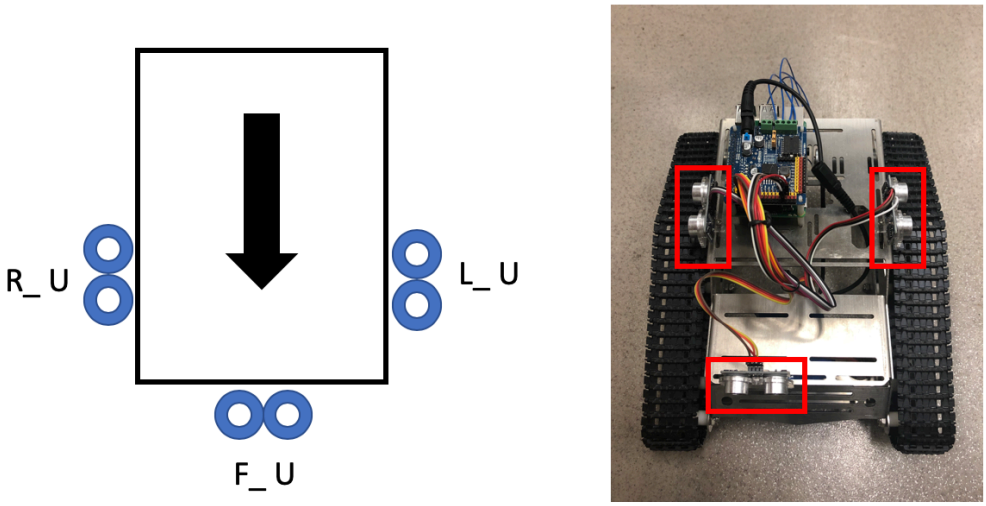

Fig. 4.1: Ultrasonic Sensors Installation

4.1.1 Hardware Setup

The HC-SR04 ultrasonic sensor has four pins, which are GND (ground), ECHO, TRIG, VCC. The ”GND” pin was connected to ”GND” pin on the Raspberry Pi. The ”VCC” pin was connected to ”5V” pin on the Raspberry Pi. However, the TRIG and ECHO of these ultrasonic sensors were connected to different pins on the Raspberry Pi:

- Left Ultrasonic Sensor: ECHO → pin 7, TRIG → pin 5
- Front Ultrasonic Sensor: ECHO → pin 4, TRIG → pin 17
- Right Ultrasonic Sensor: ECHO → pin 23, TRIG → pin 22

4.1.2 Ranging Principle

In this project, we let ultrasonic transmitter sends an ultrasonic wave in a certain direction. The timing starts as this ultrasonic wave was transmitted. The ultrasonic wave returns back as it encountered an obstacle. The ultrasonic receiver immediately stop timing when it receives the reflection. The velocity of ultrasonic propagation in the air is v, and according to the time difference Δt between transmitting and receiving. we can calculate the distance S from the launch point to the obstacle using the following formula.

$$S = v \times \Delta t/2$$

4.1.3 Code Implementation

The wall detection function implementation was programmed with the following Python code. The following code is the distance detection function of front side ultrasonic sensor.

```
"""
Front ultrasonic module code
"""

FRONT_ECHO = 4  # Echo
FRONT_TRIG = 17  # Trig

GPIO.setup(FRONT_TRIG, GPIO.OUT, initial=GPIO.LOW)  # Ultrasonic module
    transmitter setting
GPIO.setup(FRONT_ECHO, GPIO.IN, pull_up_down=GPIO.PUD_UP)  # Ultrasonic module
    receiver setting

def get_front_distance():
    time.sleep(0.05)
    GPIO.output(FRONT_TRIG, GPIO.HIGH)
    time.sleep(0.000015)
    GPIO.output(FRONT_TRIG, GPIO.LOW)
    while not GPIO.input(FRONT_ECHO):
        pass
    t1 = time.time()
    while GPIO.input(FRONT_ECHO):
        pass
    t2 = time.time()
    time.sleep(0.1)
    return (t2 - t1) * 340 / 2 * 100

def detect_front():
    dis_send = int(get_front_distance())
    if dis_send < 20:
        print('Distance from the front: %d cm' % dis_send)
        return False
    else:
        print("Front: no obstacle detected")
        return True
```

The $get\_front\_distance()$ function is used to calculate the distance between front-side ultrasonic sensor and obstacle. It is called by the $detect\_front()$ function. In the body of $detect\_front()$ function, an obstacle judgement is implemented. If the distance returned by $detect\_front()$ is less than 20 cm, it is recognized that there is obstacle in the front. Otherwise, it is recognized that there is no obstacle detected in the front.

4.2 Ground Color Detection

In this project, we only set the ground color of end in maze as black. Therefore, as robot arrives at the end and detects that the color of ground is black, it is able to know that the end has been arrived and terminate the program. Two E18-D80NK infrared sensors were used to implement the ground color detection. The infrared sensors installation illustration is shown in Fig. 4.2. Both two infrared sensors are installed on the font of robot. The positions of these two infrared sensors are marked by red rectangle in real graph in Fig. 4.2.

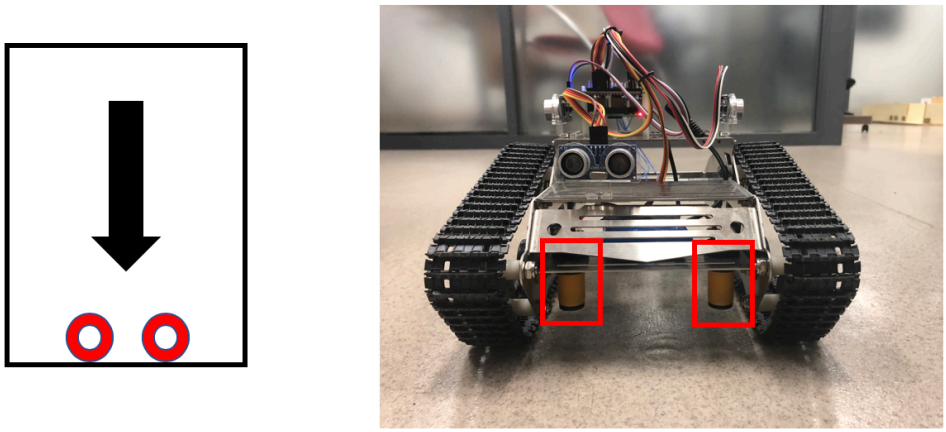

Fig. 4.2: Infrared Sensors Installation

4.2.1 Hardware Setup

The E18-D80NK infrared sensor has three pins, which are GND (ground), ”+5V” and ”DIGITAL OUTPUT”. The ”GND” pin was connected to ”GND” pin on the Raspberry Pi. The ”+5V” pin was connected to ”5V” pin on the Raspberry Pi. Then, ”DIGITAL OUTPUT” is connected to functional pins. The left infrared sensor is connected to pin 27 and the right one is connected to pin 18.

4.2.2 Detection Principle

The principle of infrared sensors is to use light reflection. Infrared sensor launches the infrared ray to the ground and then receives the reflected light. Because normal colors can reflect the light back, and black is the color that absorbs the most lights. Therefore, when the ground color of current position is black, the infrared ray is absorbed, and the receiver of sensor cannot detect the reflected signal. Through this mechanism, the black color is detected.

4.2.3 Code Implementation

Similarly, in order to make use of Input / Output pin on the Raspberry Pi. RPI.GPIO is imported into the code. Furthermore, another modules such as time, binascii [33], threading were also used.

The following code is the black detection function of these two infrared sensors.

```
"""
Infrared module code
"""

"""
Infrared sensor pin number
"""
IR_R = 18  # right infrared sensor
```

```
IR_L = 27  # left infrared sensor

# Infrared initializes as input and pulls up
GPIO.setup(IR_R, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(IR_L, GPIO.IN, pull_up_down=GPIO.PUD_UP)

def detect_end():
    if (GPIO.input(IR_L) == True)&(GPIO.input(IR_R) == True):      # detect
    black on both sides.
        return True
    else:
        return False
```

IR_L refers to the pin 27 on the Raspberry Pi and IR_R refers to the pin 18 on the Raspberry Pi. If the input of IR_L and IR_R are high, refered to $True$, it denotes that these infrareds detected the black color.

4.3 Motor Movement Implementation

The function of motor movement is implemented in the sensor.py. The corresponding pin numbers connected on Raspberry Pi are 13, 20, 19, 16, 21, 26. The following code is the movement implementation of robot.

```

"""
Motor code
"""

# Definition of motor drive interface
ENA = 13
ENB = 20
IN1 = 19
IN2 = 16
IN3 = 21
IN4 = 26

# Initialize motor to LOW
GPIO.setup(ENA, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(IN1, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(IN2, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(ENB, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(IN3, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(IN4, GPIO.OUT, initial=GPIO.LOW)

"""
Motor function definition
"""

def motor_forward():
    # print 'motor forward'
    GPIO.output(ENA, True)
    GPIO.output(ENB, True)
    GPIO.output(IN1, True)
    GPIO.output(IN2, False)
    GPIO.output(IN3, True)
    GPIO.output(IN4, False)

```

```
35 def motor_backward():
36     # print 'motor_backward'
37     GPIO.output(ENA, True)
38     GPIO.output(ENB, True)
39     GPIO.output(IN1, False)
40     GPIO.output(IN2, True)
41     GPIO.output(IN3, False)
42     GPIO.output(IN4, True)
43
44 def motor_turn_left():
45     # print 'motor_turn_left'
46     GPIO.output(ENA, True)
47     GPIO.output(ENB, True)
48     GPIO.output(IN1, True)
49     GPIO.output(IN2, False)
50     GPIO.output(IN3, False)
51     GPIO.output(IN4, True)
52
53 def motor_turn_right():
54     # print 'motor_turn_right'
55     GPIO.output(ENA, True)
56     GPIO.output(ENB, True)
57     GPIO.output(IN1, False)
58     GPIO.output(IN2, True)
59     GPIO.output(IN3, True)
60     GPIO.output(IN4, False)
61
62 def motor_stop():
63     # print 'motor_stop'
64     GPIO.output(ENA, False)
65     GPIO.output(ENB, False)
66     GPIO.output(IN1, False)
67     GPIO.output(IN2, False)
68     GPIO.output(IN3, False)
69     GPIO.output(IN4, False)
```

Four related functions were defined. The function $motor\_forward$ make both motors to rotate forward at the same speed, which make robot move forward. In the same way, the function $motor\_backward$ make both motors to rotate backward at the same speed, which make robot move backward.

Because the direction of motors is fixed, which towards front. Therefore, to implement Steering function, I utilized the turning mode of tank to make the car realize the turning function. Taking advantage of the speed difference between the two wheels, when the speed of one wheel is lower than that of the other wheel, the robot will deflector to the side with the lower speed. Therefore, we control the rotation speed of two wheels to make robot turn by controlling the output voltage of two pins, ENA and ENB.

In this project, we fixed the equal rotation speed on both sides of the robot, and only change the rotation direction of the motors on both sides. When turning left or right, we only need to set the rotation direction of the wheels on the left or right side to turn backward, so that the robot can turn in place. By this method, the function $motor\_turn\_left$ implements the turn left operation and the function $motor\_turn\_right$ implements the turn right operation.

4.4 Maze Solving Implementation

The core codes involved in maze solving algorithm are implemented in *maze_coordinate.py* file. In this file, four global variables are defined, and their purpose are listed following:

- **current_path**: The stack to store the current path of robot.
- **cross**: The stack to store the coordinate of crosses robot visited.
- **start**: The coordinate of start.
- **robot**: The robot object.

Furthermore, three classes were implemented, which are Stack class, Position class, Robot class. The detailed implementation is shown in the following part.

The core part is the Maze Solving Algorithm implementation, which is implemented as the go_maze function in *maze_coordinate.py*.

4.4.1 Stack Class

There is no stack data structure in Python language. Therefore, in this project, the stack is realized by self-programming. Its support operations are listed below:

- **push**($item$):
  push $item$ into the stack.
- **pop**():
  Retrieve and remove the top item of stack.
- **peek**():
  Retrieve, but does not remove, the top item of stack.
- **is_empty**():
  Return whether the stack is empty.
- **size**():
  Return the number of items.
- **show**():
  Show the information of all items.
- **check_in**($item$):
  Check if $item$ exists in the stack.

The detailed implementation is shown in *Stack.py* in appendix.

4.4.2 Position Class

The position class is used to simulate the coordinate in maze. It two instance attributes:

- **x**: x-axis coordinate
- **y**: y-axis coordinate

The instance functions are listed:

- **is_equal(**$position$**)**: check if $position$ to self.position.
- **print_position()**: print the information of coordinate.

The detailed implementation is shown in *Position.py* in appendix.

4.4.3 Robot Class

The robot class simulate the action of robots in maze. In the start of this program, one robot instance is created. The robot instance has related operations such as turning, moving, detecting obstacle and detecting end. The instance of this class has three attributes. The instance attributes and functions are listed below.

Instance Attributes

- **position**: the coordinate of robot
- **direction_name**: the direction_name of robot
- **direction**: the direction of robot

Instance Functions

- **set_direction_name()**:
  receives the input and set the initial value of $direction\_name$.
- **set_direction()**:
  according to the input, set the value of $direction$ variable.
- **detect_left()**:
  detect the left side of robot.
- **detect_front()**:
  detect the front side of robot.
- **detect_right()**:
  detect the right side of robot.

- **move_forward()**:
  move forward.

- **turn_right()**:
  turn right.

- **turn_left()**:
  turn left.

The detailed implementation is shown in *Robot.py* in appendix.

4.4.4 MSA Implementation

The Maze Solving Algorithm (MSA) is the core part of this project. It implements the algorithm introduced in 3.4. The implementation of this algorithm is programmed in *MazeCoordinate.py*.

```
1
2  def go_maze(position):
3  """
4  Maze solving algorithm main body
5  :param position: current position
6  :return:
7  """
8  s
9  print "current position: (", position.x, ",", position.y, ")"
10
11 """
12 Pushes the current location into the current_path stack
13 """
14 current_path.push(position)
15
16 """
17 Returns the current location from the recorded path in current_path stack
18 """
19 current_position = current_path.peek()
20
21 if test_end():      # Check if the current location is the end using infrared
       sensors
22     print("Reach the end")
23     return
24 else:
25     if check_cross():   # Check whether current position is a cross using
       ultrasonic sensors
26         """
27         If it is a cross, check if this cross has ever been reached
28         """
29         if repeat_check(current_position):
30             """
31             If the cross has been reached once, pop the stack to the place
32             where the cross last appeared in cross stack
33             """
34             pop_cross(current_path, current_position)
35         else:
36
37             """
38             If the cross has not been reached, this cross is recorded into the
       cross stack
```

```
39                 """
40                 cross.push(current_position)
41
42         perform_action(current_position)
43         next_position = Position(robot.position.x, robot.position.y)
44         go_maze(next_position)
```

4.4.5 Algorithm Function

This part implements the functions called in MSA main body, which are aimed to implement some special operations. The functions below are both global function defined in *MazeCoordinate.py* file and can be called globally in *MazeCoordinate.py*.

- **init()**:
  Init the whole program, such as $current\_path$, $cross$, $start$, $robot$

- **check_cross()**:
  Check if the current location is a cross with ultrasonic sensors.

- **pop_cross(stack, position)**:
  Pop the stack to the place where the last cross occurred.

- **repeat_check(position)**:
  Check if the cross has been arrived before.

- **perform_action(position)**
  Robot performs actions after moving to new position, it will call the $perform\_cross\_action()$ or $perform\_not\_cross\_action()$.

- **perform_cross_action()**:
  Robot performs related actions when the current position is a cross.

- **perform_not_cross_action()**:
  Robot performs related actions when the current position is not a cross.

- **test_end()**:
  Detect if the current position is the end.

- **output()**:
  Output the information, such as the path from start to end.

The detailed implementations of above functions are in *MazeCoordinate.py* shown in appendix.

Chapter 5

Evaluation

5.1 Test Environment

To validate if the robot can succeed in solving maze problem, A lot of experiments are carried out on an actual maze in indoor environment. The structure of actual maze is shown in Fig. 5.1. The red line marks the beginning and the black ground area marks the end. The tracked mobile robot

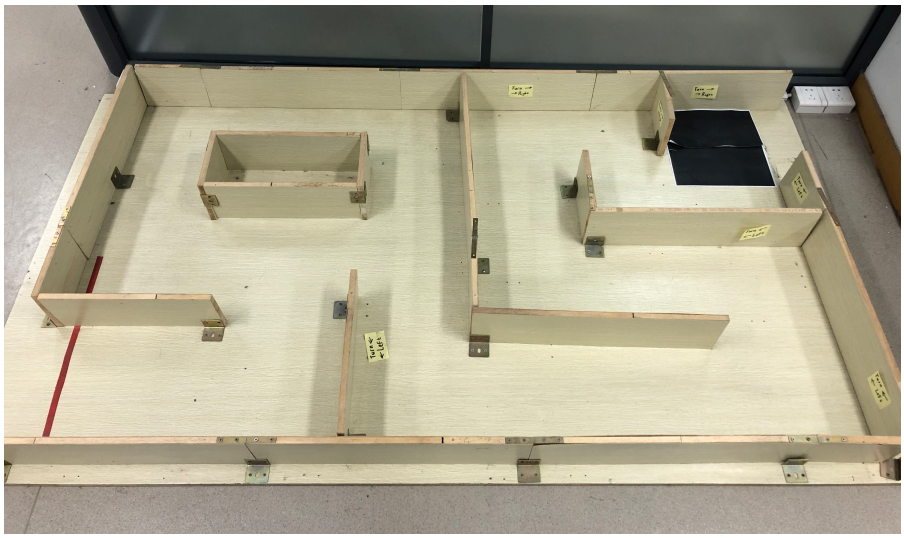

Fig. 5.1: Test Maze

starts at the red line and traverse the maze by the program developed in this project. The timing starts at the starting of robot and terminates when robot arrived at the end. The correctness of output refers to whether the output path in the experiment is equal to the output path in the theory. For the actual maze in Fig. 5.1, the theoretical output path oughts to be the sequence shown below. (The (0,0) denotes the start and (5,3) denotes the end.)

(0 , 0), (1 , 0), (1 , 1), (2 , 1), (2 , 0), (3 , 0), (4 , 0), (5 , 0), (5 , 1), (4 , 1), (3 , 1), (3 , 2), (3 , 3), (4 , 3), (4 , 2), (5 , 2), (5 , 3)

5.2 Test Results

In this test, the initial direction of robot is set to four kind of values, which are X+, X-, Y+, Y-. For each direction setting situation, the experiments are performed 10 times. In each experiment, the time cost of going out of maze and output correctness are recorded.

The results of these experiments are shown in Table 5.1.

Table 5.1: The results of experiments

Direction_Name Setting	Test Times	Average Time Cost	Output Correctness Rate
X+	10	1:05.89	0.9
X-	10	1:04.63	1
Y+	10	1:05.26	1
Y-	10	1:06.53	0.9
Total	40	1:05.58	0.95

The experiments are carried out 40 times in total. In each experiment, the robot successfully got out of the maze, which proves the effectiveness of the algorithm and program. The average time cost of these experiments is 1:05.58. The output Correctness rate is 0.85 but not 1 in total. The total output correctness rate denotes that the output paths returned of 2 in 40 experiments, are not same with the theoretical output path listed in 5.1.

The reason for this correctness is the limitation of actual the maze environment. The maze was too tight for the size of the tracked mobile robot. Therefore, in these two experiments whose output path were not correct, the robot got stuck at certain corners and tried many times before it succeeded. Because it tried to turn multiple times in one same corner, the program running in robot thought that it has turned multiple corners. Therefore, the directions of robot changed multiple times rather than only once time, which caused the error occurred in the calculation of coordinates. This kind of error occurred occasionally in the experiments.

5.3 User Feedback

The maze solving robot in this project is a game-based learning application. Its additional objective is supporting the secondary development, which allows the users to learn through this maze solving game.

Therefore, the questionnaire shown in Table 5.2 were distributed to 20 users, and the feedbacks were collected. The satisfaction degree is divided into five levels with increasing order from

dissatisfied to satisfied: 1, 2, 3, 4, 5. The feedbacks of these 20 users are shown in Table 5.3. (The numbers in cell are number of people who selected this option)

Table 5.2: Questionnaire

Question	1	2	3	4	5
Code Style & Clarity					
Game Play Degree					
Willing to Play					
Helpful for Learning Programming					

Table 5.3: Feedback

Question	1	2	3	4	5
Code Style & Clarity	0	0	0	4	16
Game Play Degree	0	0	5	7	8
Willing to Play	0	0	3	8	9
Helpful for Learning Programming	0	0	0	3	17

The feedbacks show that the application developed in this project follows the good standard, which means that it is simple for users to conduct the secondary development. Additionally, almost all of the respondents think this maze solving game-based learning application is quite useful for learning programming. However, the other two aspects are not very impeccable, which are the degree of game play and willingness to play. Perhaps the reason is that the maze solving game is not quite novel, and not as appealing as digital computer games in game market.

Chapter 6

Learning Points

Working on this project has given me a better understanding of how to conduct research and design new approaches to solving problems in the field of computer science and technology. In this part, the useful things that I have learned from this project are introduced.

The robot of the project needs to interact with the environment and act accordingly. In order to use ultrasonic sensors and infrared sensors, I obtained some professional knowledge in electronic engineering. I am now able to use python code to control and read output of the electronic modules. Furthermore, as the program running environment is Raspbian, which is a UNIX-like platform. I learned some useful knowledge about Linux instructions. I am able to program through terminal in the UNIX-like platform. The programming language used in this project is python. In order to make this project smoothly, I enhanced my python programming skills as well as be proficient in object-oriented programming skills such as classes and methods. Additionally, a coordinate system in this project is required, I learned the knowledge about cartesian coordinate system, and designed a dynamic coordinate system in real combining with my own mathematical knowledge. To solve the maze problem, an algorithm for the actual maze needs to be designed. This made me work on more advanced data structures and more efficient algorithms.

As the time is constrained, I need to manage my project properly so that it can be completed before the deadline. I learned the project manage skills in software development. I used increment model in software development. The project was divided into multiple components. Each component is designed and built separately. The whole project is designed, implemented and tested incrementally. With this learned skill, I can implement every component on time during the development of this project.

Chapter 7

Professional Issues

I confirm that this project obey the Code of Practice and Code of Conduct issued by British Computer Society, and I believe that the whole project is designed to serve the public interest. I guarantee that the whole project is completed by myself. Furthermore, this project is believed to have the ability to benefit other people. In the process of carrying out this project, I have followed the ”Key IT practices” stated in the code of practices.

Chapter 8

Conclusion

This project produces a maze solving game-based learning application based on Raspberry Pi-controlled robot. Equipped with ultrasonic sensors and infrared sensors, the robot was programmed with python language and achieves the ability to go out of real maze. The DFS algorithm is used as the basic principle to traverse the maze. Furthermore, combining the MSA (Maze Solving Algorithm) and dynamic coordinate system designed in this project, the robot has the ability to obtain a path solution from start to end in real maze and the path solution is returned in the form of a sequence of coordinates. Besides, the project is developed in good code style, and also does well in encapsulation and modularity. Therefore, this project is quite convenient for students to carry out the secondary development for teaching purposes. In future, some work can be done to improve the accuracy of robot, and additional functionalities for other kinds of games can be implemented in the robot.

Reference

[1] J.-N. Proulx, M. Romero, and S. Arnab, "Learning Mechanics and Game Mechanics Under the Perspective of Self-Determination Theory to Foster Motivation in Digital Game Based Learning," *arXiv.org*, no. 1, pp. 81–97, May 2018.

[2] D. Ifenthaler, D. Eseryel, and X. Ge, *Assessment in Game-Based Learning*, ser. Foundations, Innovations, and Perspectives. Springer Science & Business Media, Jun. 2012.

[3] P. Felicia, *Game-Based Learning*, ser. Challenges and Opportunities. Cambridge Scholars Publishing, Jun. 2014.

[4] M. Prensky, *Digital Game-Based Learning*. Paragon House, Mar. 2007.

[5] A. Tveit, T. Morland, and T. B. Røst, "DeepLearningKit - an GPU Optimized Deep Learning Framework for Apple's iOS, OS X and tvOS developed in Metal and Swift." *CoRR*, 2016.

[6] S.-I. Ao, B. B. Rieger, and S.-S. Chen, *Advances in Computational Algorithms and Data Analysis*. Springer Science & Business Media, Sep. 2008.

[7] H. W. Bullen IV and P. Ranjan, "Chaotic Transitions in Wall Following Robots," *arXiv.org*, Aug. 2009.

[8] R. Klein and T. Kamphans, "Pledge's Algorithm - How to Escape from a Dark Maze." *Algorithms Unplugged*, no. Chapter 8, pp. 69–75, 2011.

[9] "Solving a Maze," in *Building Robots with LEGO Mindstorms NXT*. Syngress, Jan. 2007, pp. 327–348.

[10] R. Tiwari, *Intelligent Planning for Mobile Robotics: Algorithmic Approaches*, ser. Algorithmic Approaches. IGI Global, Sep. 2012.

[11] A. Brændeland, "Depth-first search in split-by-edges trees," *arXiv.org*, May 2015.

[12] E. Upton and G. Halfacree, *Raspberry Pi User Guide*. John Wiley & Sons, Aug. 2016.

[13] A. G. Blank, *TCP/IP Foundations*. John Wiley & Sons, Feb. 2006.

[14] D. A. S. Anbalagan, "Secure Shell (SSH) - Public Key Authentication over Hypertext Transfer Protocol (HTTP)." *CoRR*, 2015.

[15] D. Barrett, R. Silverman, and R. Byrnes, *SSH, The Secure Shell*, ser. The Definitive Guide. ”O’Reilly Media, Inc.”, May 2005.

[16] R. Cowart and B. Knittel, *Using Microsoft Windows XP Professional Edition*. Que Pub, 2003.

[17] E. Upton and G. Halfacree, *Raspberry Pi User Guide*. John Wiley & Sons, Sep. 2014.

[18] G. Hart-Davis, *Deploying Raspberry Pi in the Classroom*. Apress, Dec. 2016.

[19] S. Monk, *Programming the Raspberry Pi: Getting Started with Python*. McGraw Hill Professional, Nov. 2012.

[20] B. Aubakir, Z. Kappasov, and A. Shintemirov, “Practical Realization of the Self-Balancing Robot Using Infrared Sensors,” *arXiv.org*, Mar. 2015.

[21] J. Jeuring and E. Meijer, *Advanced Functional Programming*, ser. First International Spring School on Advanced Functional Programming Techniques, Bastad, Sweden, May 24 - 30, 1995. Tutorial Text. Springer Science & Business Media, May 1995.

[22] A. Gibbons, *Algorithmic Graph Theory*. Cambridge University Press, Jun. 1985.

[23] J. Sun, J. Y. 0011, W. X. Zheng, and S. Li, “GPIO-Based Robust Control of Nonlinear Uncertain Systems Under Time-Varying Disturbance With Application to DC-DC Converter.” *IEEE Trans. on Circuits and Systems*, vol. 63, no. 11, pp. 1074–1078, 2016.

[24] W. Gay, *Advanced Raspberry Pi*, ser. Raspbian Linux and GPIO Integration. Apress, Oct. 2018.

[25] A. Kurniawan, *Raspberry Pi I/O Programming Using Python*. PE Press.

[26] S. Shah, *Learning Raspberry Pi*. Packt Publishing Ltd, Apr. 2015.

[27] T. Hoffstadt and J. Maas, “Sensorless BCM control for a bidirectional flyback-converter.” *IECON*, pp. 3600–3605, 2016.

[28] M. Bell, *Incremental Software Architecture*, ser. A Method for Saving Failing IT Implementations. John Wiley & Sons, Jan. 2016.

[29] G. Bergmann, Á. H. 0001, I. Ráth, D. Varró, A. Balogh, Z. B. 0003, and A. Ökrös, “Incremental Evaluation of Model Queries over EMF Models.” *MoDELS*, vol. 6394, no. 3, pp. 76–90, 2010.

[30] C. Roser and M. Nakano, “Guidelines for the Selection of FIFO Lanes and Supermarkets for Kanban-Based Pull Systems - When to Use a FIFO and When to Use a Supermarket.” *APMS*, vol. 460, no. Chapter 33, pp. 282–289, 2015.

[31] M. L. González-Martínez, L. Bonnet, P. Larrégaray, J. C. Rayez, and J. Rubayo-Soneira, "From angle-action to Cartesian coordinates: A key transformation for molecular dynamics," *arXiv.org*, no. 11, p. 114103, Nov. 2008.

[32] B. Khoussainov, A. Nies, and R. A. Shore, "Recursive models of theories with few models," 1995.

[33] F. Lundh, *Python Standard Library*. "O'Reilly Media, Inc.", 2001.

Chapter 9

Appendix

9.1 Sensor.py

```
# coding:utf-8
from socket import *
from time import ctime
import binascii
import RPi.GPIO as GPIO
import time
import threading

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)


"""
Motor code
"""

# Definition of motor drive interface
ENA = 13
ENB = 20
IN1 = 19
IN2 = 16
IN3 = 21
IN4 = 26

# Initialize motor to LOW
GPIO.setup(ENA, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(IN1, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(IN2, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(ENB, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(IN3, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(IN4, GPIO.OUT, initial=GPIO.LOW)


"""
Motor function definition
"""


def motor_forward():
    # print 'motor forward'
    GPIO.output(ENA, True)
```

```
42      GPIO.output(ENB, True)
43      GPIO.output(IN1, True)
44      GPIO.output(IN2, False)
45      GPIO.output(IN3, True)
46      GPIO.output(IN4, False)
47
48
49  def motor_backward():
50      # print 'motor_backward'
51      GPIO.output(ENA, True)
52      GPIO.output(ENB, True)
53      GPIO.output(IN1, False)
54      GPIO.output(IN2, True)
55      GPIO.output(IN3, False)
56      GPIO.output(IN4, True)
57
58
59  def motor_turn_left():
60      # print 'motor_turn_left'
61      GPIO.output(ENA, True)
62      GPIO.output(ENB, True)
63      GPIO.output(IN1, True)
64      GPIO.output(IN2, False)
65      GPIO.output(IN3, False)
66      GPIO.output(IN4, True)
67
68
69  def motor_turn_right():
70      # print 'motor_turn_right'
71      GPIO.output(ENA, True)
72      GPIO.output(ENB, True)
73      GPIO.output(IN1, False)
74      GPIO.output(IN2, True)
75      GPIO.output(IN3, True)
76      GPIO.output(IN4, False)
77
78
79  def motor_stop():
80      # print 'motor_stop'
81      GPIO.output(ENA, False)
82      GPIO.output(ENB, False)
83      GPIO.output(IN1, False)
84      GPIO.output(IN2, False)
85      GPIO.output(IN3, False)
86      GPIO.output(IN4, False)
87
88
89  """
90  Front ultrasonic module code
91  """
92
93  FRONT_ECHO = 4  # Echo
94  FRONT_TRIG = 17  # Trig
95
96
97  GPIO.setup(FRONT_TRIG, GPIO.OUT, initial=GPIO.LOW)  # Ultrasonic module
        transmitter setting
98  GPIO.setup(FRONT_ECHO, GPIO.IN, pull_up_down=GPIO.PUD_UP)  # Ultrasonic module
        receiver setting
99
```

```
100
101 def get_front_distance():
102     time.sleep(0.05)
103     GPIO.output(FRONT_TRIG, GPIO.HIGH)
104     time.sleep(0.000015)
105     GPIO.output(FRONT_TRIG, GPIO.LOW)
106     while not GPIO.input(FRONT_ECHO):
107         pass
108     t1 = time.time()
109     while GPIO.input(FRONT_ECHO):
110         pass
111     t2 = time.time()
112     time.sleep(0.1)
113     return (t2 - t1) * 340 / 2 * 100
114
115
116 def detect_front():
117     dis_send = int(get_front_distance())
118     if dis_send < 20:
119         print('Distance from the front: %d cm' % dis_send)
120         return False
121     else:
122         print("Front: no obstacle detected")
123         return True
124
125
126 """
127 Left ultrasonic module code
128 """
129 LEFT_ECHO = 7  # Echo
130 LEFT_TRIG = 5  # Trig
131 #
132
133 GPIO.setup(LEFT_TRIG, GPIO.OUT, initial=GPIO.LOW)  # Ultrasonic module
    transmitter setting
134 GPIO.setup(LEFT_ECHO, GPIO.IN, pull_up_down=GPIO.PUD_UP)  # Ultrasonic module
    receiver setting
135
136
137 def get_left_distance():
138     time.sleep(0.05)
139     GPIO.output(LEFT_TRIG, GPIO.HIGH)
140     time.sleep(0.000015)
141     GPIO.output(LEFT_TRIG, GPIO.LOW)
142     while not GPIO.input(LEFT_ECHO):
143         pass
144     t1 = time.time()
145     while GPIO.input(LEFT_ECHO):
146         pass
147     t2 = time.time()
148     time.sleep(0.1)
149     return (t2 - t1) * 340 / 2 * 100
150
151
152 def detect_left():
153     dis_send = int(get_left_distance())
154     if dis_send < 20:
155         print('Distance from the left: %d cm' % dis_send)
156         return False
157     else:
```

```
158         print("Left: no obstacle detected")
159         return True
160
161
162 """
163 Right ultrasonic module code
164 """
165
166 RIGHT_ECHO = 23  # Echo
167 RIGHT_TRIG = 22  # Trig
168
169 GPIO.setup(RIGHT_TRIG, GPIO.OUT, initial=GPIO.LOW)  # Ultrasonic module
    transmitter setting
170 GPIO.setup(RIGHT_ECHO, GPIO.IN, pull_up_down=GPIO.PUD_UP)  # Ultrasonic module
    receiver setting
171
172
173 def get_right_distance():
174     time.sleep(0.05)
175     GPIO.output(RIGHT_TRIG, GPIO.HIGH)
176     time.sleep(0.000015)
177     GPIO.output(RIGHT_TRIG, GPIO.LOW)
178     while not GPIO.input(RIGHT_ECHO):
179         pass
180     t1 = time.time()
181     while GPIO.input(RIGHT_ECHO):
182         pass
183     t2 = time.time()
184     time.sleep(0.1)
185     return (t2 - t1) * 340 / 2 * 100
186
187
188 def detect_right():
189     dis_send = int(get_right_distance())
190     if dis_send < 20:
191         print('Distance from the right: %d cm' % dis_send)
192         return False
193     else:
194         print("Left: no obstacle detected")
195         return True
196
197
198 """
199 Infrared module code
200 """
201
202 """
203 Infrared sensor pin number
204 """
205 IR_R = 18  # right infrared sensor
206 IR_L = 27  # left infrared sensor
207
208
209 # Infrared initializes as input and pulls up
210 GPIO.setup(IR_R, GPIO.IN, pull_up_down=GPIO.PUD_UP)
211 GPIO.setup(IR_L, GPIO.IN, pull_up_down=GPIO.PUD_UP)
212
213
214 def detect_end():
```

```
215     if (GPIO.input(IR_L) == True)&(GPIO.input(IR_R) == True):      # detect
        black on both sides.
216         return True
217     else:
218         return False
```

9.2 MazeCoordinate.py

```
1   # coding:utf-8
2 import Sensor as rf
3 import Stack as sk
4 import Position as ps
5 import Robot as rb
6
7 current_path = None
8 cross = None
9 start = None
10 robot = None
11
12
13 def init():
14     """
15     Initialize the program
16     :return:
17     """
18     global current_path, cross, start, robot
19     current_path = sk.Stack()
20     cross = sk.Stack()
21     start = ps.Position(0, 0)
22     robot = rb.Robot(ps.Position(0, 0))
23
24
25 def check_cross():
26     """
27     Check if the current location is a cross with ultrasonic sensors
28     :return:
29     """
30     counter = 0
31
32     if rf.detect_left():
33         counter += 1
34     if rf.detect_front():
35         counter += 1
36     if rf.detect_right():
37         counter += 1
38
39     if counter > 1:
40         return True
41     else:
42         return False
43
44
45 def pop_cross(stack, position):
46     """
47     Pop the stack to the place where the last cross occurred
48     :param stack: cross stack
49     :param position: cross position
50     :return:
```

```
    """
    while True:
        last_cross = stack.peek()
        if last_cross.is_equal(position):
            break
        stack.pop()


def repeat_check(position):
    """
    Check if the cross has been arrived before
    :param position: current position
    :return:
    """
    global cross
    arrived = False
    if cross.check_in(position):
        arrived = True

    return arrived


def perform_cross_action():
    """
    Perform the actions when current position is a cross
    :return:
    """

    global robot

    print "Perform the cross actions"

    if rf.detect_left():
        robot.turn_left()
        robot.move_forward()
    elif rf.detect_front():
        robot.move_forward()
    elif rf.detect_right():
        robot.turn_right()
        robot.move_forward()


def perform_not_cross_action():
    """
    Perform the actions when current position is not a cross
    :return:
    """

    global robot

    print "Perform the un_cross actions"

    if rf.detect_left():
        robot.turn_left()
        robot.move_forward()
    elif rf.detect_front():
        robot.move_forward()
    elif rf.detect_right():
        robot.turn_right()
        robot.move_forward()
```

```
    else:
        robot.turn_right()
        robot.turn_right()
        robot.move_forward()


def perform_action(position):
    """
    Perform the actions
    :param position: current position
    :return:
    """

    global cross

    if cross.check_in(position):    # current position is a cross
        perform_cross_action()
    else:                           # current position is not a cross
        perform_not_cross_action()


def test_end():
    """
    Detect if the current position is destination
    :return:
    """

    if rf.detect_end():
        return True
    else:
        return False


def output():
    """
    Output the information
    :return:
    """
    global current_path
    print "The path from start to end:\n"
    current_path.show()


def go_maze(position):
    """
    Maze solving algorithm main body
    :param position: current position
    :return:
    """

    print "\ncurrent position: (", position.x, ",", position.y, ")"

    """
    Pushes the current location into the current_path stack
    """
    current_path.push(position)

    """
    Returns the current location from the recorded path in current_path stack
    """
```

```
    current_position = current_path.peek()

    if test_end():      # Check if the current location is the end using infrared sensors
        print("Reach the end")
        return
    else:
        if check_cross():   # Check whether current position is a cross using ultrasonic sensors
            """
            If it is a cross, check if this cross has ever been reached
            """
            if repeat_check(current_position):
                """
                If the cross has been reached once, pop the stack to the place where the cross last appeared in cross stack
                """
                pop_cross(current_path, current_position)
            else:

                """
                If the cross has not been reached, this cross is recorded into the cross stack
                """
                cross.push(current_position)

        perform_action(current_position)
        next_position = ps.Position(robot.position.x, robot.position.y)
        go_maze(next_position)


# Program starts
init()
go_maze(start)
output()
```

9.3 Robot.py

```
import Sensor as rf


class Robot:
    """
    The Robot Class
    """

    def __init__(self, position):
        self.position = position
        self.direction_name = self.set_direction_name()
        self.direction = self.set_direction()

    def is_not_used(self):
        pass

    def set_direction_name(self):
        self.is_not_used()

        while True:
```

```
            direction_name = raw_input("set the initial direction:\n1. X+\n2. X-\n3. Y+\n4. Y-\n")
            if direction_name == "X+" or direction_name == "X-" or direction_name == "Y+" or direction_name == "Y-":
                break
            else:
                print "Invalid direction, enter again!"

        return direction_name


# Coordinate System
    def set_direction(self):
        if self.direction_name == "X+":
            return 0
        elif self.direction_name == "X-":
            return 2
        elif self.direction_name == "Y+":
            return 1
        elif self.direction_name == "Y-":
            return -1
        else:
            print "Invalid direction!"

    def detect_left(self):
        self.is_not_used()
        if rf.detect_left():
            return True
        else:
            return False

    def detect_front(self):
        self.is_not_used()
        if rf.detect_front():
            return True
        else:
            return False

    def detect_right(self):
        self.is_not_used()
        if rf.detect_right():
            return True
        else:
            return False

    def move_forward(self):
        self.is_not_used()
        print("Move forward")
        rf.motor_forward()
        rf.time.sleep(1)
        rf.motor_stop()
        self.calculate_coordinate()     # Robot move and calculate coordinate

    def turn_left(self):
        self.is_not_used()
        print("Turn left")

        rf.motor_backward()
        rf.time.sleep(0.1)
        rf.motor_stop()
```

```
79
80          rf.motor_turn_left()
81          rf.time.sleep(0.5)
82          rf.motor_stop()
83          self.direction = self.direction + 1
84          self.calculate_direction_name()
85          print "Current direction:", self.direction_name, self.direction
86
87      def turn_right(self):
88          self.is_not_used()
89          print("Turn Right")
90
91          rf.motor_backward()
92          rf.time.sleep(0.1)
93          rf.motor_stop()
94
95          rf.motor_turn_right()
96          rf.time.sleep(0.5)
97          rf.motor_stop()
98          self.direction = self.direction - 1
99          self.calculate_direction_name()
100         print "Current direction:", self.direction_name, self.direction
101
102
103 # Coordinate System
104     def calculate_coordinate(self):
105         if self.direction_name == "X+":
106             self.position.x += 1
107         elif self.direction_name == "X-":
108             self.position.x -= 1
109         elif self.direction_name == "Y+":
110             self.position.y += 1
111         elif self.direction_name == "Y-":
112             self.position.y -= 1
113         else:
114             print ("Wrong direction information!")
115
116 # Coordinate System
117     def calculate_direction_name(self):
118         self.test_direction_reset()
119
120         if self.direction == 0:
121             self.direction_name = "X+"
122         elif self.direction == -2 or self.direction == 2:
123             self.direction_name = "X-"
124         elif self.direction == 1 or self.direction == -3:
125             self.direction_name = "Y+"
126         elif self.direction == -1 or self.direction == 3:
127             self.direction_name = "Y-"
128         else:
129             print "invalid direction"
130
131 # Coordinate System
132     def test_direction_reset(self):
133         if self.direction == 4:
134             self.direction = 0  # Robot rotate  360 degrees counterclockwise
135         elif self.direction == -4:
136             self.direction = 0  # Robot rotate 360 degrees clockwise
```

9.4 Stack.py

```
1
2  class Stack:
3  """
4  The Stack Class
5  """
6
7  def __init__(self):
8    self.items = []
9
10 def is_empty(self):
11   return len(self.items) == 0
12
13 def push(self, item):
14   self.items.append(item)
15
16 def pop(self):
17   return self.items.pop()
18
19 def peek(self):
20   if not self.is_empty():
21     return self.items[len(self.items) - 1]
22
23 def show(self):
24   if not self.is_empty():
25     for i in self.items:
26       i.print_position()
27
28 def size(self):
29   return len(self.items)
30
31 def check_in(self, position):
32   if position in self.items:
33     return True
34   else:
35     return False
```

9.5 Position.py

```
1
2  class Position:
3  """
4  Coordinate
5  """
6
7  def __init__(self, x, y):
8    self.x = x
9    self.y = y
10
11 def is_equal(self, position):
12   return self.x == position.x and self.y == position.y
13
14 def print_position(self):
15   print "(", self.x, " , ", self.y, ")"
```