University of Glasgow

University of Electronic Science and Technology of China

Glasgow College
University of Science and Technology of China

Smart Vehicle Design

Faster Stronger Together

Final Report

Team Design Project and Skills Initial Report

by

Team 09 “Faster Stronger Together”

Member Name	UoG Matrix	Contribution
Liu Weixing	2614287L	13%
Bai Tianyou	2614321B	13%
Wang Zhiyi	2614317W	13%
Wang Ziyu	2614407W	10%
Liu Zhuqing	2613941L	11%
Fan Xilai	2614111F	10%
Zhao Zidong	2613940Z	10%
Wang Chen	2614195W	8%
Zhang Tianyi	2614046Z	8%
Yang Tian	2614170Y	4%

Instructors: Dr. Abdullah Al-Khalidi, Dr. Wasim Ahmad, Dr. Oluwakayode Onireti
Institution: Glasgow College, UESTC
Place: University of Electronic Science and Technology of China
Project Duration: February, 2023 – June, 2023

Abstract

This report presents the design and implementation of a self-tracking intelligent vehicle capable of recognizing arrows and throwing balls. The project involves a team of 10 individuals who collaborate to achieve the project objectives. The report outlines the overall system design, including task decomposition, software and hardware structure analysis. Additionally, detailed subsystem designs are introduced, covering aspects such as vehicle structure, main control chip, power supply, motor drive module, Steering gear installation, vision identification system, gyroscope, ultrasonic radar, communication and clock are discussed in this report. The PID controller for linear drive and automatic tracking is also addressed. Furthermore, the integration of these components and the resulting performance of the robot in line tracking, arrow recognition and ball throwing tasks are analyzed and discussed. Finally, project management is emphasized with a focus on teamwork, planning using Gantt charts as well as budgeting.

Acknowledgements

Thank you very much to our three professors who taught us a lot of knowledge and patiently answered all the questions about the project.

I would also like to thank the teaching assistant in this course. Due to the teacher's busy work or time difference, many emails could not be answered immediately. At this time, the teaching assistant built a bridge between students and teachers, and promoted the progress of the whole project

I would also like to thank all the members of our team. TDPS is a team project. Without cooperation, we would not be able to complete such work.

Contents

List of Figures

LIST OF FIGURES

List of Tables

1

Introduction

Team Design Project and Skills (TDPS) is a course that emphasizes cooperation. In this course, the students are asked to make a smart vehicle and complete corresponding tasks, at the same time, improve our ability of team cooperation. This chapter will give a basic introduction to the background and objectives.

1.1 Background Information (by Wang Zhiyi)

In recent years, the field of vehicles has seen significant advancements, with vehicles becoming increasingly sophisticated and capable of performing a wide range of tasks. This has been driven by rapid developments in technology, including advancements in artificial intelligence, machine learning, and sensors.

One of the key drivers of this progress has been the increasing demand for automation in various industries. As businesses seek to improve efficiency and reduce costs, they are turning to vehicles to perform tasks that are repetitive, dangerous, or require a high level of precision. This has led to the development of vehicles that can perform tasks in manufacturing, logistics, and agriculture, among other industries.

In this context, the TDPS course provides an opportunity for engineering students to explore the potential of vehicles and gain practical experience in designing and building integrated electronic systems. By working collaboratively in teams, students can develop their skills in problem-solving, project management, and teamwork, while also contributing to the development of new technologies that could shape the future of society. In the TDPS project, students are required to work collaboratively in teams to design and develop an integrated electronic system. This project involves several stages, including the planning and design phase, the construction phase, and the testing and evaluation phase. During the planning and design phase, students must identify the specific functions that the system will perform and determine the necessary components and materials. They must also consider the budget constraints and ensure that the final product meets the project requirements.

1.2 Objectives

TDPS aims to design, build and demonstrate a smart vehicle that is capable of performing two sets of tasks automatically and seamlessly. The two sets are distributed on Patio 1 and Patio 2 separately, each set consists of 3 sequential sub-tasks.

1.2.1 Patio 1

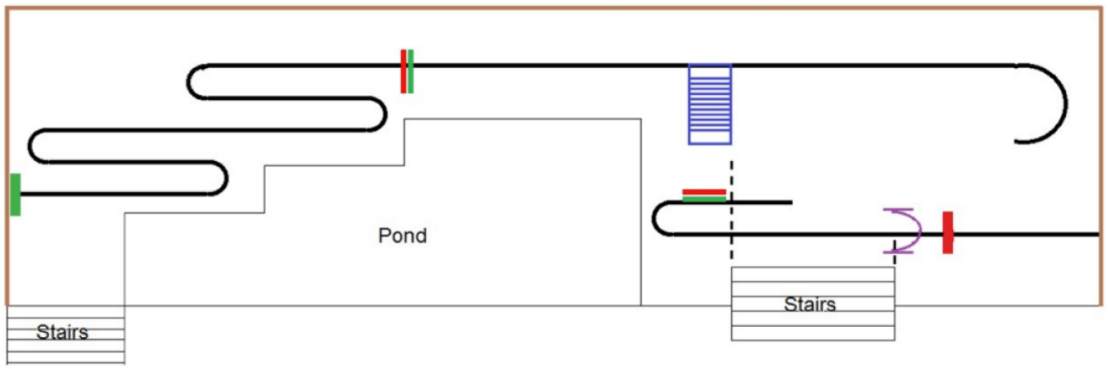


Figure 1.2.1: Patio 1 diagram

The operating area of Patio1 is illustrated in figure. The vehicle needs to:

1. follow the path of the colored tiles from the green starting point to a box at the destination **[Task1]**
2. Recognize the wire mesh bridge near the box and cross on top of it **[Task2]**
3. Find and go through the gate and then stop **[Task3]**

1.2.2 Patio 2

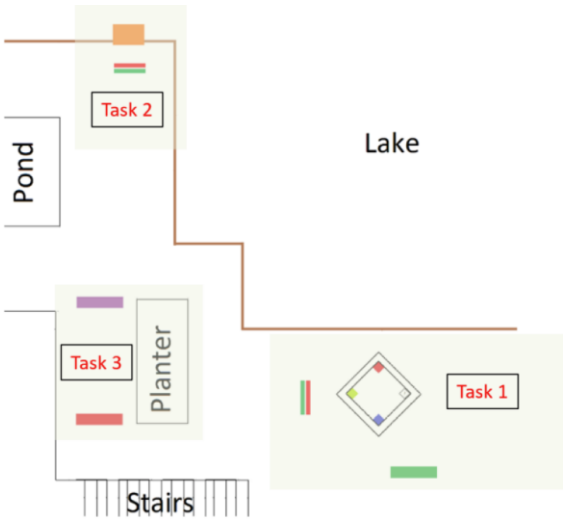


Figure 1.2.2: Patio 2 diagram

The operating area of Patio2 is illustrated in figure. The vehicle needs to:

1. Proceed with line cruising from a green starting point to the yellow square at the right-hand side of the diamond, determine the shape of arrow on the stand and move to the corresponding direction to knockout the matched shape on the stand given at location 1, 2 and 3 **[Task 1]**;
2. Move to the table tennis ball release point (marked in orange in figure) in any possible approach if general rules are not contravened, and release the pre-installed tennis ball into the basket **[Task 2]**;
3. Move to the planter area and stop to transmit a message to a laptop. The message received at the laptop should contain **[Task 3]**:

- Team Name
- Date, day and time
- Time sent should be the current time of day (24-hour clock)
- Proceed with line cruising after successful transmission to reach the destination.

The transmission should be completed by a radio signal at 433 MHz at fixed data rate applying a HC-12 wireless transceiver.

1.2.3 General Rules

The vehicle must be trained and programmed in advance, with the program downloaded into the microcontroller beforehand. At the same time, **real-time commands** are prohibited during the final demonstration.

To help the vehicle complete its task, the team was allowed to use two **beacons** on each patio as visual references for navigation. The beacons can be placed anywhere necessary to further ensure that the vehicle navigates as anticipated.

The **electrical system** and all connections to circuit components and subsystems must be prepared and ensure the **robustness** and **reliability**. While most of the components can be purchased directly online, the team is expected to design the motor drive circuit itself on a printed circuit board (PCB). Moreover, wires should be soldered or screwed to PCBs, V-board, or punchboard as breadboard circuits are not allowed.

With a **budget constraint** of 1,000 yuan, the team will be given **complete autonomy** t o design and construct the vehicle to complete the assigned tasks. All financial costs mu st be accurately documented, itemized and presented with a detailed explanation for th e selection of components, contributing to effective **engineering project management** and finance.

2

Overall System Design

2.1 Task Breakdown (by Liu Weixing)

To accomplish all tasks in each patio completely, it is necessary to clearly identify the functions required to achieve these tasks. These functions are found and listed as following.

2.1.1 Patio 1

1. Line tracking(camera): identify the path intended to follow and travel along it. Note the presence of straight and curved roads in the route, which means that the vehicle needs to make straight and turning movements.

2. Distance measurement(beacon): in order for the vehicle to know when to cross the bridge, it is necessary to use beacons that can be detected by distance detectors.

3. Steering control: the vehicle needs to accurately turn 90 degrees to the right to prevent it from deviating on the narrow bridge and falling out of it.

4. Bridge crossing: crossing this bridge smoothly and smoothly requires the smoothness and horsepower of the vehicle.

5. Arch passing through: after re-entering the trajectory, the vehicle needs to pass through this arch and ultimately stop at the designated location.

2.1.2 Patio 2

1. Arrow recognition: identify the arrow of the first diamond to determine the next action information.

2. Steering control: based on the previously identified arrow, turn to the tip of the corresponding diamond and knock it out.

3. Distance measurement (fence, basket): maintain a safe distance from the right fence through the distance sensor on the right to prevent collisions and inform the vehicle when to turn 90 degrees to the right. The distance sensor in front prevents the vehicle from colliding with the fence and inform it when to turn 90 degrees to the left. The distance sensor in front also detects the basket and notify the vehicle when to place the table tennis into the basket.

4. Tennis releasing: throw the ball into the basket and turn left 90 degrees to move to the Planter area.

5. Wireless communication: stop at the Planter area and then send team message to the laptop.

OVERALL SYSTEM DESIGN

Based on a detailed division of the functions that vehicles need to perform, team members have a clearer understanding of how to implement these two Patio and their own work contributions. This is beneficial for improving the initial design ideas and work direction of the group. Afterwards, the hardware and software designs of the group were based on these various functions.

2.2 Structure Analyze (by Bai Tianyou)

As for the overall system, we designed it in parts, and our team also divided into software and hardware to carry out the work. The structure of the hardware and software will be shown separately next.

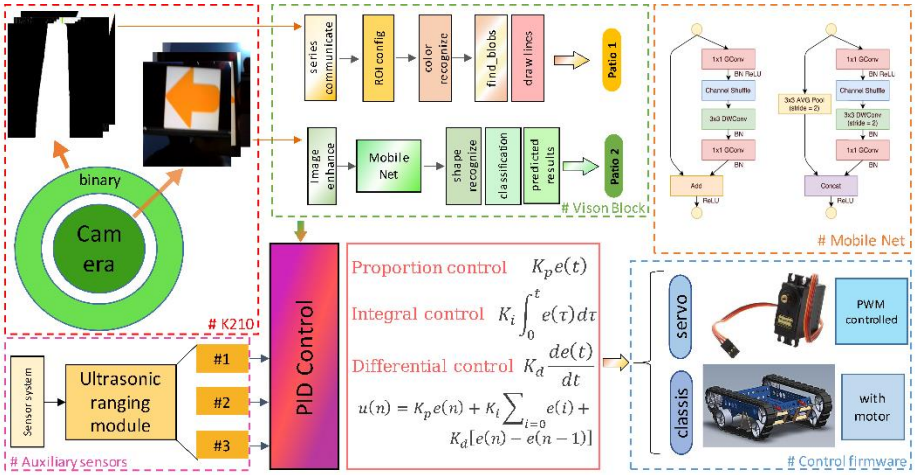


Figure 2.2.1: The system structure

2.2.1 Hardware structure

The design of the hardware part involves the combination of various devices, the design of the power supply mode, the placement of the hardware on our vehicle, etc., all of which will be covered in detail in Part 3. This part will introduce the overall architecture of the hardware and give a schematic diagram (as shown below).

The structure is shown in **Figure 2.2.2**

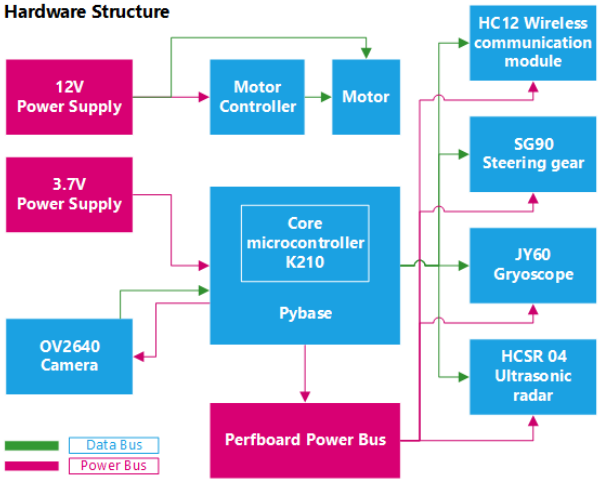


Figure 2.2.2: The brief structure of hardware

In the diagram, the red line represents the transfer of **power** while the green line represents the transfer of **data**. It is easy to see that the whole system is composed of K210+Pybase as the core, K210 is a microcontroller with strong computing power, and Pybase is a supporting expansion board, where all data and operations take place and issue commands to each subsystem.

It is worth noting that due to the **limited power supply pins** on K210 and Pybase, we

use Perfboard to treat each module as common ground and common high. In other words, these modules are uniformly powered by K210, but properly allocated to Perfboard.

2.2.2 Software structure

Different from the hardware structure, the software structure is relatively **flat**, because the whole system is not too complex, so we choose the scheme that a core program, surrounding by a number of package libraries. Since the core board K210 uses **micropython** library, this project is mainly developed in **python**.

In addition to the internal packaging library of **micropython** and the **OpenMV visual library** carried in the firmware, the structure of our own packaging library is as follows:

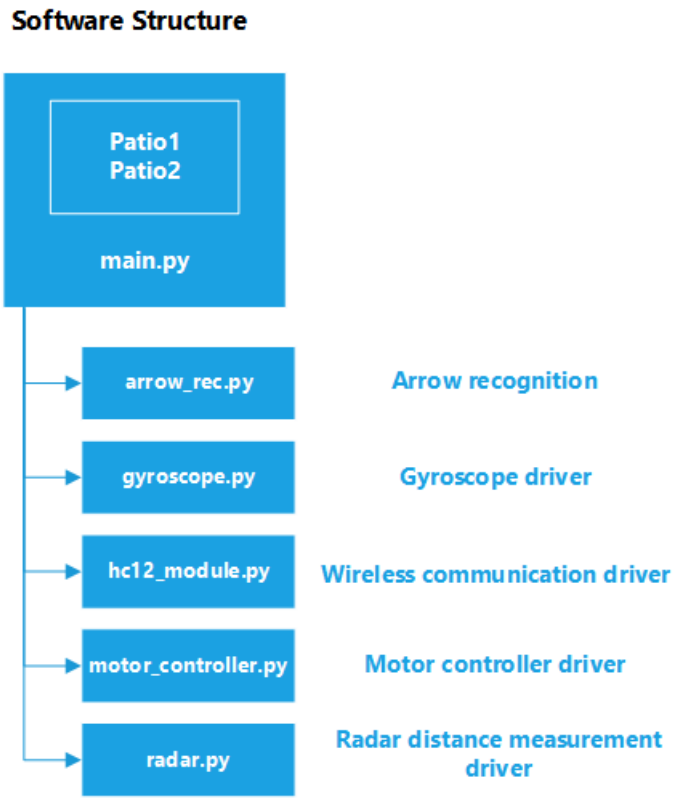


Figure 2.2.3: The brief structure of software

For the Patio1 and Patio2, a different main program is used. However, the libraries invoked have remained the same, as shown in **Figure 2.2.3**. These libraries have different roles, some handling arrowhead recognition tasks, while others may include motor or gyroscope drives, etc.

By calling these libraries, the whole program runs. The specific functions of each module will be elaborated in the subsystem section (Chapter 3).

3

Subsystem Design

3.1 Vehicle Structure (by Zhao Zidong)

In the following sections, we will discuss the structure of the vehicle's body and the assembly process. We will explore the materials used, the design considerations, and the step-by-step process of assembling the body.

3.1.1 Track driven

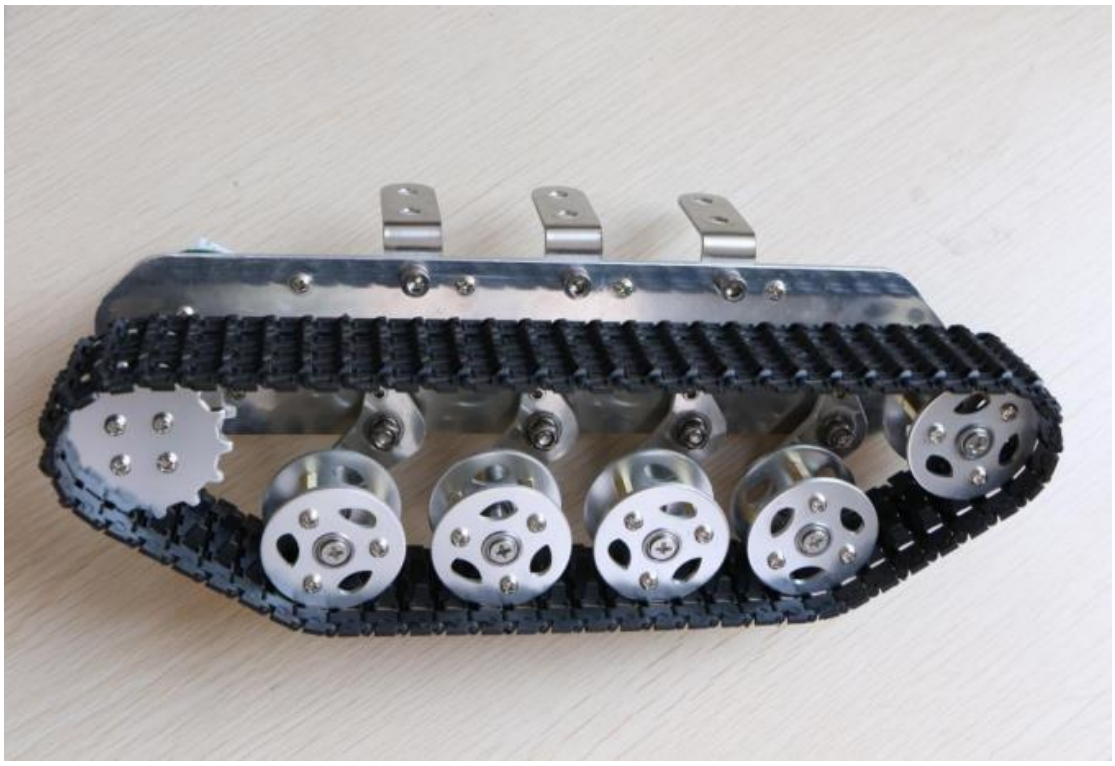


Figure 3.1.1: Picture of vehicle caterpillar belt

In the design of intelligent small vehicles, tracks are often used instead of wheels. The advantage of tracks is that they provide better traction and stability, making it easier to navigate uneven terrain. Additionally, tracks distribute the weight of the vehicle more evenly, reducing the risk of tipping over.

One of the challenges of using **tracks**, however, is that they are not as maneuverable as wheels. To address this, some track-based vehicles use a technique called skid steering, which involves varying the speed and direction of the tracks on each side of the vehicle

to turn. Another approach is to use a differential mechanism, which allows the tracks on

each side to rotate at different speeds, enabling the vehicle to turn smoothly. By leveraging the strengths of tracks and implementing effective turning mechanisms, intelligent small vehicles can achieve both stability and maneuverability, making them ideal for a variety of applications.

We used the **differential method**. This method involves varying the speed of the tracks on each side of the vehicle, causing the vehicle to turn in the direction of the slower-moving track. For example, if the left track moves slower than the right track, the vehicle will turn to the left. By using this method, the vehicle can turn smoothly and efficiently, even in tight spaces. Differential steering is achieved through the use of a differential mechanism, which allows the tracks on each side to rotate at different speeds. This mechanism can be controlled using a microcontroller, which receives input from sensors and adjusts the speed of the tracks accordingly. This microprocessor will be mentioned in the PCB section.

3.1.2 Vehicle body

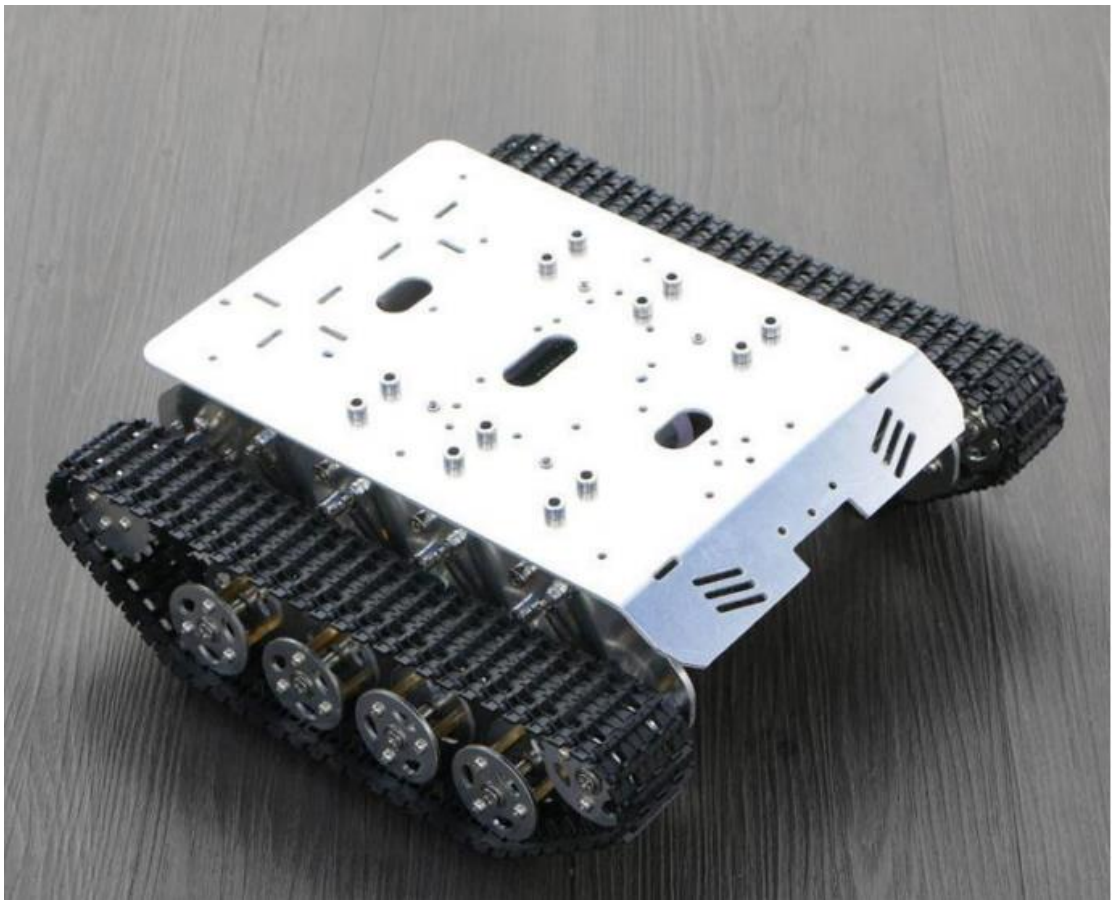


Figure 3.1.2: Overall picture of vehicle body

The vehicle design features a metal frame with perforations on its **surface** that can be used to attach other brackets and boards. This design provides a high degree of flexibility and allows for easy customization of the vehicle's components. By attaching additional components to the perforated surface, designers can add sensors, cameras, and other devices to enhance the vehicle's functionality.

One of the main advantages of using a **perforated surface** is that it provides a simple and efficient way to add and remove components. This makes it easy to modify the

vehicle's design and adapt it to changing requirements. Additionally, the perforations allow for a high **degree of precision in component placement**, which is essential for ensuring optimal performance.

However, there are also some potential drawbacks to this design. The perforations may weaken the structural integrity of the surface, making it more susceptible to damage. Additionally, the perforations may limit the amount of weight that the surface can support, which could impact the vehicle's overall **load-bearing capacity**.

3.1.3 Result and discussion

In fact, after we actually connected, we found some problems:

The speed on both sides of the motor is different: the motor speed is controlled by PWM waves, and although there is a calibration screw, the different speed still does not lead to a good straight-line walking over long distance.

Car plate uneven/too few holes: the car plate is not smooth enough and the holes on the car plate are not very reasonable, so that there are some problems when the device is installed on the car plate.

In addition, since the structure of the car is metal, we were worried about the resistance problem at first, and finally found that the metal surface has insulation paint. So, the whole frame is very good.

3.2 Main Control Chip – K210+Pybase (by Liu Weixing)

As shown in the second part, K210 and Pybase are the core components of the entire hardware architecture, on which almost all of the computation and logic runs.

At the same time, it also has many pins, and can communicate with each module. Unlike many microcontrollers, the K210 can configure each pin independently, that is, almost all pins can be configured to GPIO, AnalogIO, UART, I2C or other states, which greatly increases the possibility of free arrangement of each module.

The following will first introduce some **basic properties** of K210 and its expansion board Pybase, its firmware **built-in algorithm library**, and do some understanding and analysis of its actual use.

3.2.1 Main control parameter

As the **Table 3.2.1** shows[1], some basic parameters like **power supply** were mentioned in the table, which is the best reference for us to build the system.

K210		Pybase	
Item	**Parameter**	**Item**	**Parameter**
Architecture	RISC-V	Power Supply	3.6-6V
Bit number	64	DAC/ADC	Yes
Core number	2	temperature sensor	DS18B20
Firmware	Micropython	humidity sensor	DHT11
Power input	3.6-6V	Battery charging circuit	Yes
High level	3.3V		
AI computing ability	1 TOPS		

Table 3.2.1: Basic parameter of K210 and Pybase

In addition to the above data, the pin data and the corresponding firmware definition are also very important parts to study. The pin diagram of the K210 is shown in **Figure 3.2.1**:

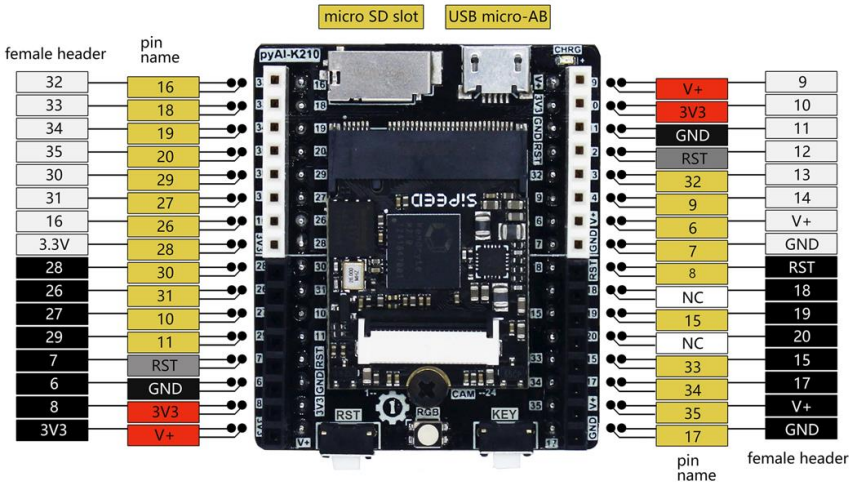


Figure 3.2.1: Pins information of K210

On the other hand, **Pybase** is closely linked to the **K210** as an expansion board in another way. The pins on Pybase correspond **one-to-one** to the interfaces of Pybase and K210, which means that when Pybase is installed on K210, the modules on it will be able to be called just by driving K210.

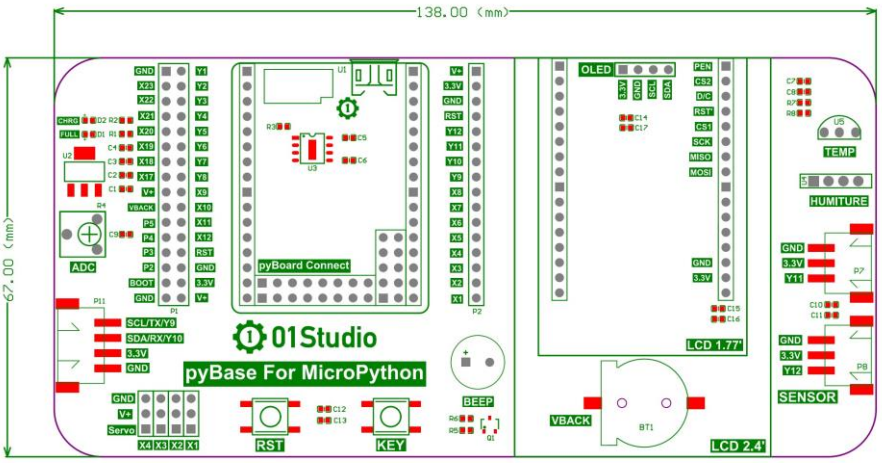


Figure 3.2.2: Structure of Pybase

Like it shows in **Figure 3.2.2**, the left side is where it connects to the K210, and the pin ports p1 and p2 (shows in **Figure 3.2.3**) can be used just as using the pins in K210.

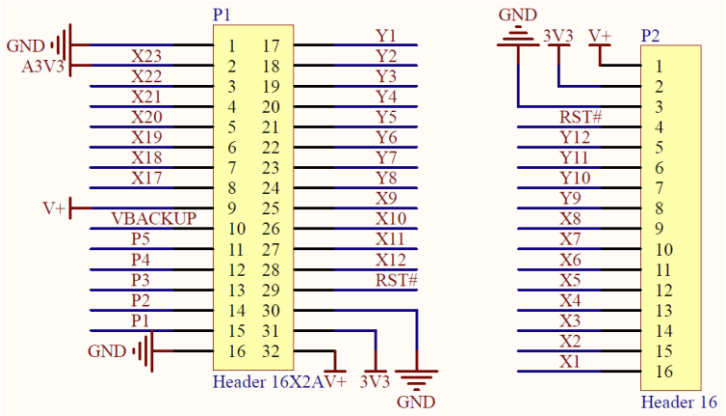


Figure 3.2.3: Port 1 and Port 2

In addition, there are many accessories, such as buzzers, temperature and humidity sensors, etc., but they are not related to this project, so these are not mentioned here.

3.2.2 Algorithm library

Before describing its built-in algorithm library, here will be a brief introduction to its **IDE**, K210 has a supporting IDE, as his software development means, after burning the firmware, open the IDE can see the various sub-interfaces, as shown in the following **Figure 3.2.4** and **Figure 3.2.5**:

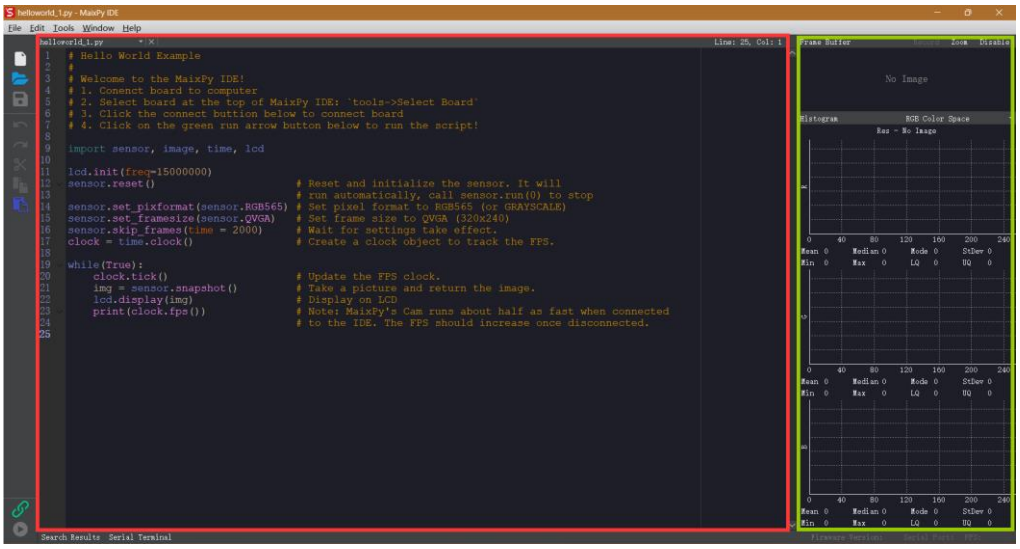


Figure 3.2.4: Development IDE GUI

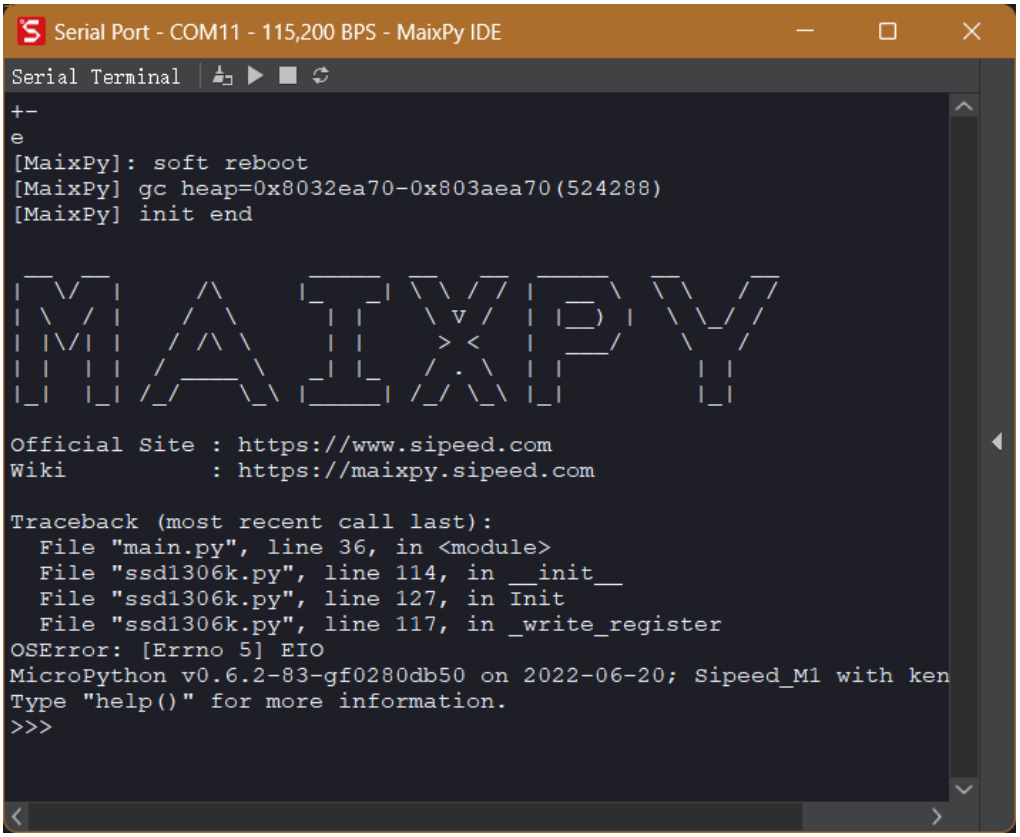


Figure 3.2.5: Serial port GUI

In **Figure 3.2.4**, The red-marked area is coding area while the green-marked area is computer vision (CV) area, it will show the graph and its histogram in buffer.

When a program was burned in, the serial port will display the information return on it. Just as a common IDE[2].

The **built-in library** is relay on the firmware. It includes some vision algorithm from **OpenMV**. It can drive the sensor and LCD (which will be mention later) easily. **The Figure 3.2.6** shows the effect for some function in vision library.

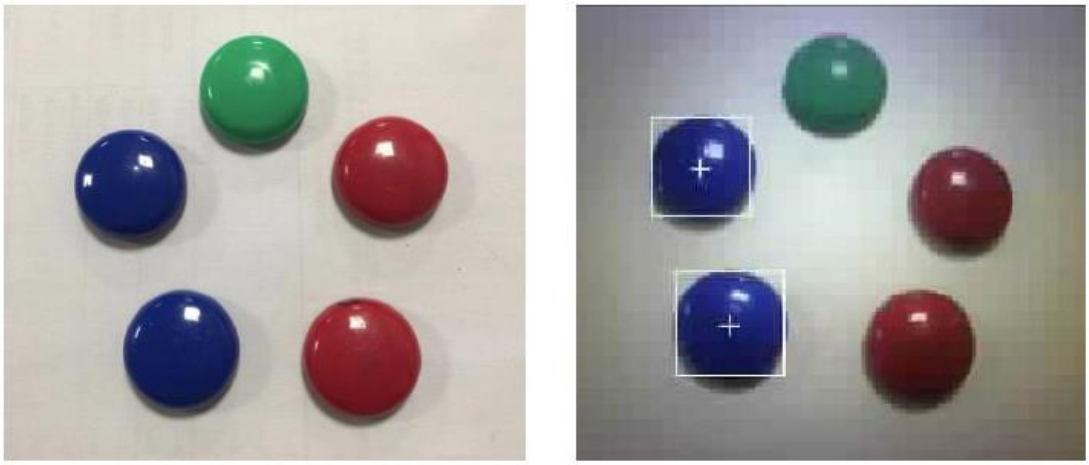


Figure 3.2.6: The effect of function find_blobs() for blue

It is an example for us that we are using the function, ***find_blobs()*** in OpenMV. Once the threshold was set, it can find the certain color in the picture.

3.2.3 Result and Discussion

The K210 microcontroller is a very powerful chip which has significant computational power. The very number of pins can also contribute to the project. However, in the process of use, the memory is much lower than the home computer or forced developers to change their habits, or it is likely to crash or disconnect due to memory overflow.

Moreover, in terms of debugging, there are also very big differences between K210 and other platform development. For single-chip microcomputer debug, it is difficult to directly access the memory to read data, can only add the corresponding debugging code.

In addition, the hardware part, the robustness of the connection, the use of environmental conditions may have an impact on it, because there is no protection, the core board is more fragile, in fact, in the later debugging, the initial K210 core board USB interface welding, resulting in us having to find a new replacement for debugging, which is also one of the debugging experiences.

3.3 Power Supply (by Liu Zhuqing)

In the intelligent tracking robot, the design of the power supply system is very important. This is because the power supply system is the foundation for the normal operation of the car. If the power supply system is not designed properly, it may cause the car to not work properly, affecting its performance and stability.

In this project, the power supply design mainly includes the following aspects:

1. Power type: It is necessary to choose a suitable power type for the car, such as batteries, chargers, etc.
2. Circuit design: It is necessary to design a reasonable circuit to ensure stable power supply to various components of the car, such as motors, sensors, etc.
3. Fault protection: It is necessary to design fault protection measures, such as short circuit protection, overcurrent protection, etc., to ensure the safety and stability of the car.

3.3.1 Power Type and Circuit Design

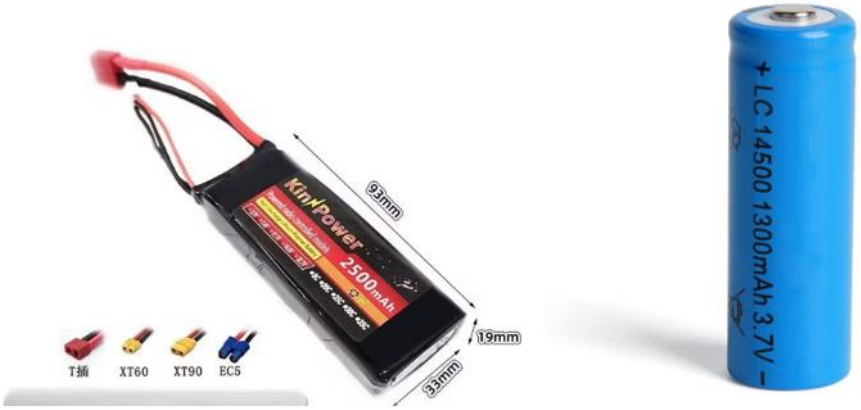


Figure 3.3.1: 12V aircraft model battery and 3.7V Li battery

In order to provide power to the MG513P30_12V DC motor[3], a 12V high-capacity aircraft model battery was selected, with a capacity of 2500mAh, which allows the small car to operate stably and for a long time. The large capacity of the battery also provides convenience for on-site testing, as there is no need to worry about the battery suddenly running out of power during testing. Because the voltage requirement of k210 is between 3v and 4v, a 3.7V 14500 lithium battery was chosen to supply power to k210. The capacity of this lithium battery is smaller than that of the aircraft model battery, which is 1300mAh. However, for K210 and various components with much lower power consumption than the motor, its battery capacity is sufficient. The figures **Figure 3.3.1** above show these two batteries[4].

The **Figure 3.3.2** shows the circuit design for the entire power supply system.

There are three main parts which are perfboard, PCB and k210 in the power system.

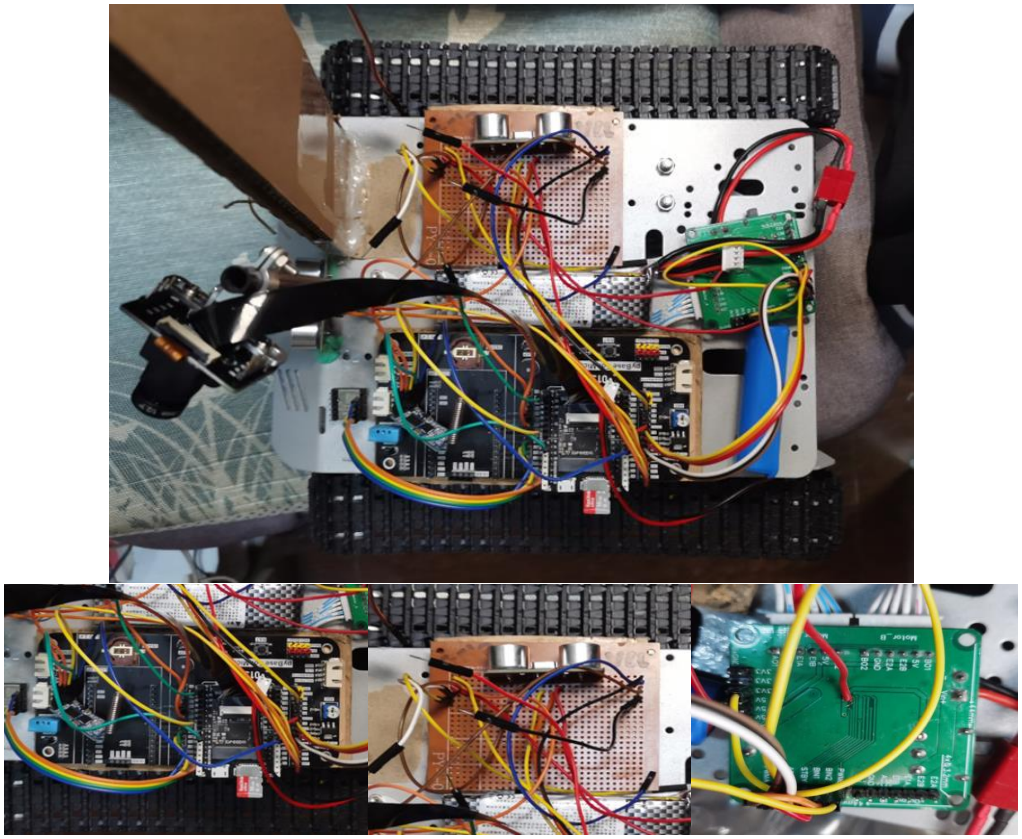


Figure 3.3.2: Entire circuit design for the power supply

For the k210 board, it is powered by the 14500 Li battery, and provide the VCC of 3.3V and GND for functional components, such as an ultrasonic sensor and the gyroscope. Due to the limited number of pins of VCC and GND on the k210, the perfboard is used to solve this problem by welding two rows of wires respectively to two wires which one of is connected to a VCC pin of k210 and another one is connected to a GND pin of k210.

As for the PCB, it is connected to the 12V aircraft model battery and provide stable power supply for the two DC motors.

3.3.2. Fault Protection

In the circuit of an intelligent patrol car, excessive current may cause damage to components in the circuit, such as motors and electronic components. Therefore, setting a fuse is very necessary as it can serve as a fuse protection in the circuit[5], protecting other components in the circuit, and improving the reliability and stability of the robot.

Figure 3.3.3: 2A glass tube fuse

By measuring the current of the car during operation with an ammeter, it was found that

the current under normal working conditions is generally 1.7A, not exceeding 1.8A. Therefore, a glass tube fuse with a rated current of 2A (as shown in **Figure 3.3.3**) was selected to protect the power supply circuit. Once the current exceeds the rated value, the fuse will blow to prevent devices from damaging.

3.3.3. Result and Discussion

In the actual test, the power supply system encountered problems such as voltage mismatch and insufficient battery capacity. Voltage mismatches can be caused by the use of incompatible power supplies or devices that cause the voltage output to be inconsistent with the voltage required by the vehicle. This can result in electronic devices not working properly or even damaged. The way to solve this problem is to carefully select the power supply or equipment to meet the requirements of the vehicle and ensure that its output voltage matches the voltage required by the vehicle.

The addition of fuses is a key protection device in the power supply system to prevent circuit overload and short circuit. It is an electrical safety device with a fuse element that fuses when the current exceeds its rating, cutting off the circuit and thereby preventing overload damage or fire to wires and equipment.

Another problem is insufficient battery capacity. Insufficient battery capacity may be due to the selection of a battery with a smaller capacity or battery aging, damage and other reasons. This will cause the battery to drain quickly, unable to provide enough power for vehicles and electronic devices. In order to solve this problem, it is necessary to select the appropriate capacity of the battery, and regularly check and maintain the battery to ensure its normal operation.

The occurrence of these problems reminds us to be very careful when choosing energy supplies. When designing and selecting a power supply system, it is necessary to consider the power needs of the vehicle to ensure that the selected power supply and equipment can provide enough power and match the voltage requirements of the vehicle. At the same time, the capacity and performance of the battery should be fully evaluated, and the appropriate battery type and capacity should be selected to meet the power needs of the vehicle[6].

3.4 Motor Drive Module Design (by Wang Zhiyi)

3.4.1 Design objective

The motor one of the most important and core parts in the vehicle. It drives the vehicle to move and change the vehicle attitude. Besides the basic moving functions above, it is also able to balance the vehicle and response to action commands timely. In our project, the two DC motors are used to drive the vehicle and the PCB using **TB6612** module is designed to control the motors.

As the vehicle could be able to move on complicated ground and make some specific movement required in the two patios. In patio 1, the vehicle should move straightly and smoothly turn along different curves. In addition, it should also be able to turn accurately at the entry of the bridge and keep balance on the uphill and downhill. In patio 2, it should turn to certain directions accurately and turn in space. Moreover, the stone road in patio 2 could influence the movement of the vehicle so that the wheel should be choose to adapt to rough road surfaces. Additionally, the control complexity is also considered since the less motor the better. As a result, the requirement for the system should follow the following points:

1. Turning smoothly, accurately and in place.
2. Keeping balance on slope and strong enough to go uphill.
3. Keeping stable on rough surface.
4. Less motor and could be better controlled by a single chip (e.g., TB6612)

There are some popular choices for vehicle wheel are considered shown in **Figure 3.4.1**, For example, the **normal wheel** with steering shaft which is common in daily car. **Mecanum wheel** is a kind of wheel that could realizing omnidirectional motion. However, we choose the **caterpillar track** as the wheel because it has more benefits than its counterparts in our general design for the project. The consideration and comparison are shown in **Table 3.1**.

Figure 3.4.1: The steering shaft, Mecanum wheel and caterpillar track vehicle

	Turning in Place	Move on Rough Surfaces	Motor Numbers	High Stability and Power
Steering Shaft	✗	✗	4	✗
Mecanum Wheel	✔	✗	4	✗
Caterpillar Track	✔	✔	2	✔

Table 3.4.1: The Comparison of the three popular wheels

Since the steering shaft wheel cannot turning in place and Mecanum wheel could lost power since the special form of the wheel. Moreover, our design could use the motor more efficiently with the duo-motor system. In conclusion, the caterpillar track meets our requirement and expectation. One **TB6612 chip** is used to control the system because the TB6612 has two output control.

3.4.2 Function realization

As for our motor, we choose DC motors[7], which could be directly driven by the battery and chips. The principle of the motor is shown in the **Figure 3.4.2**. The power supply is connected and the current is generated in the circuit to pass the magnetic induction line. According to the Ampere's rule, the forces are generated to move the motor. As the motor rotate for 180 degrees, the brushes could reverse the direction of the current to keep the force in the same direction. As a result, the direction of the motor could be controlled by the current direction and the speed is controlled by the current magnitude.

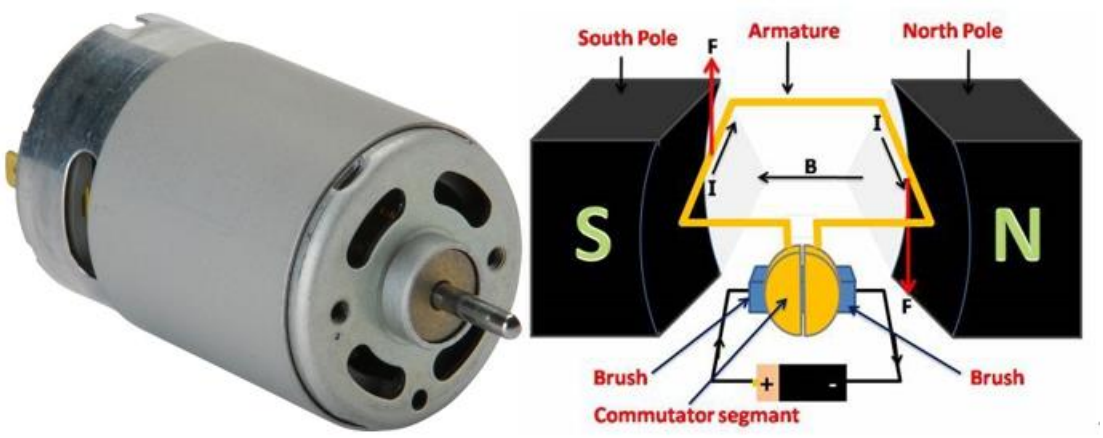


Figure 3.4.2: The DC motor and its principle

Following these rules, the motor could be controlled by only three variables, two for the direction and one for the speed. The direction is controlled by two digital inputs in either high level or in low level. A **PWM** (Pulse Width Modulation) signal is used to control the speed of the motor by changing the current magnitude in the circuit.

PWM signal is used as a control of the speed. The speed of the motor will change with the duty circle of the PWM signal. Since the PWM signal controls the switch of the power supply, the power supply for the motor is also a PWM signal. Because the RMS voltage of a PWM voltage could be controlled by the duty circle of the PWM signal, the speed of the vehicle could have the relationship as followed.

$$V = kV_{RMS} = V_{max} \times Duty\ Circle$$

However, because of the fraction, the system could not follow the linear relationship so

that the parameter to control the motor need to be tested and adjusted.

As the speed of the two motors on each side could be controlled separately, the turning could be realized by the velocity difference on each side. When the left wheel speed exceeds the right wheel speed, the vehicle will turn right and it is similar to turn left. When the total speed of the two sides which means the velocity is opposite number on the two motors, the vehicle will turn in place. The movement control is shown in **Figure 3.4.3**

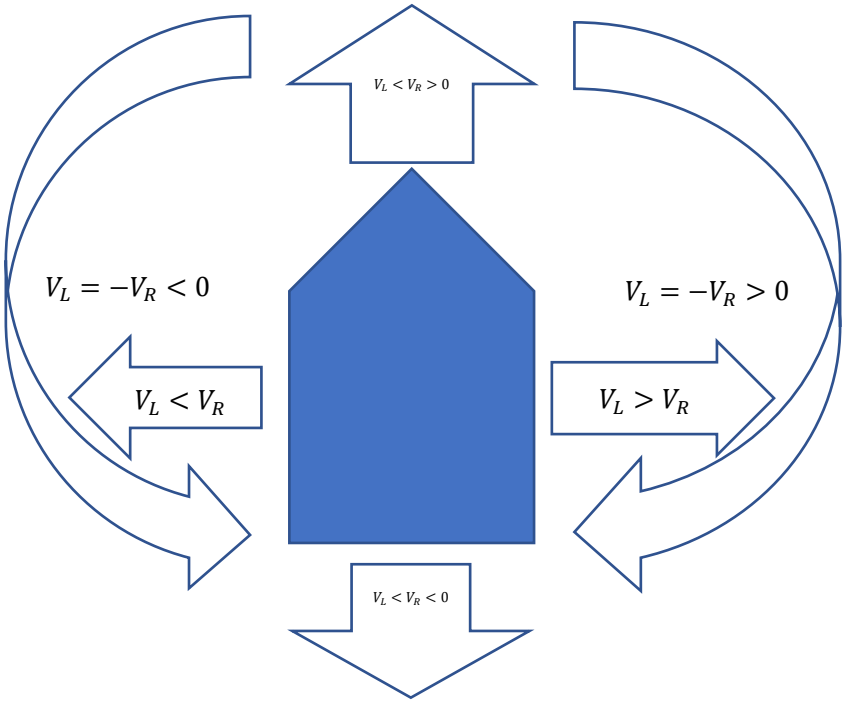


Figure 3.4.3: The movement control based on the speed difference

3.4.3 PCB design

A PCB is a complete electronic system that is integrated and stable, designed to replace breadboards or protoboards. Altium Designer (AD) is a software platform that allows for the convenient design of both schematic diagrams and PCBs without the need for net lists.

TB6612FNG chip, which is a dual-channel DC motor driver. It contains two H-bridge driver circuits that can control the rotation of the car's motors. This chip has high current output and low voltage control, making it suitable for small robots, toy cars, and other similar applications. The maximum output current of the TB6612FNG chip is 3A, and the maximum operating voltage is 15V. Moreover, it also has protection functions such as overheat protection and undervoltage lockout, ensuring the safety of the system.

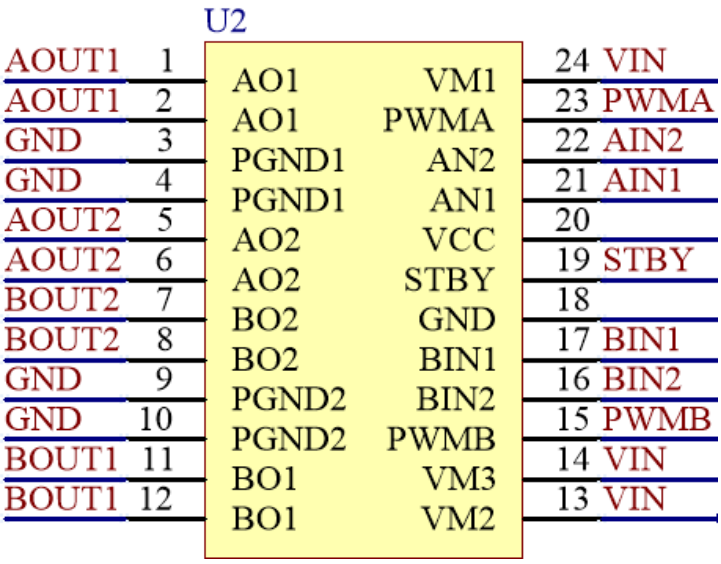


Figure 3.4.4: TB6612FNG chip

In order to ensure the normal use of the TB6612, the voltage regulator module is also necessary.

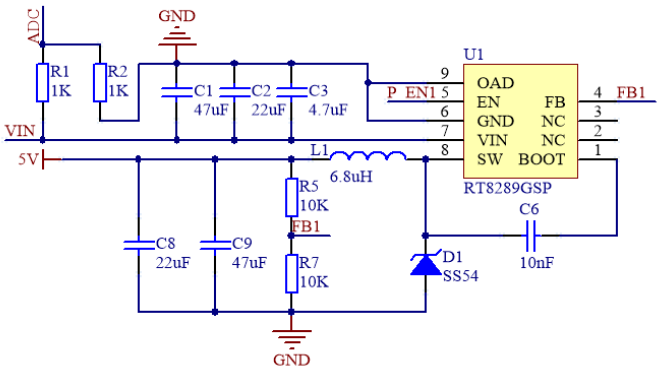


Figure 3.4.5: Voltage regulator module

Draw the other modules, resulting in a schematic diagram. These modules include: power switch, 5v to 3.3v circuit, overtemperature protection module, motor drive circuit, input and output pins.

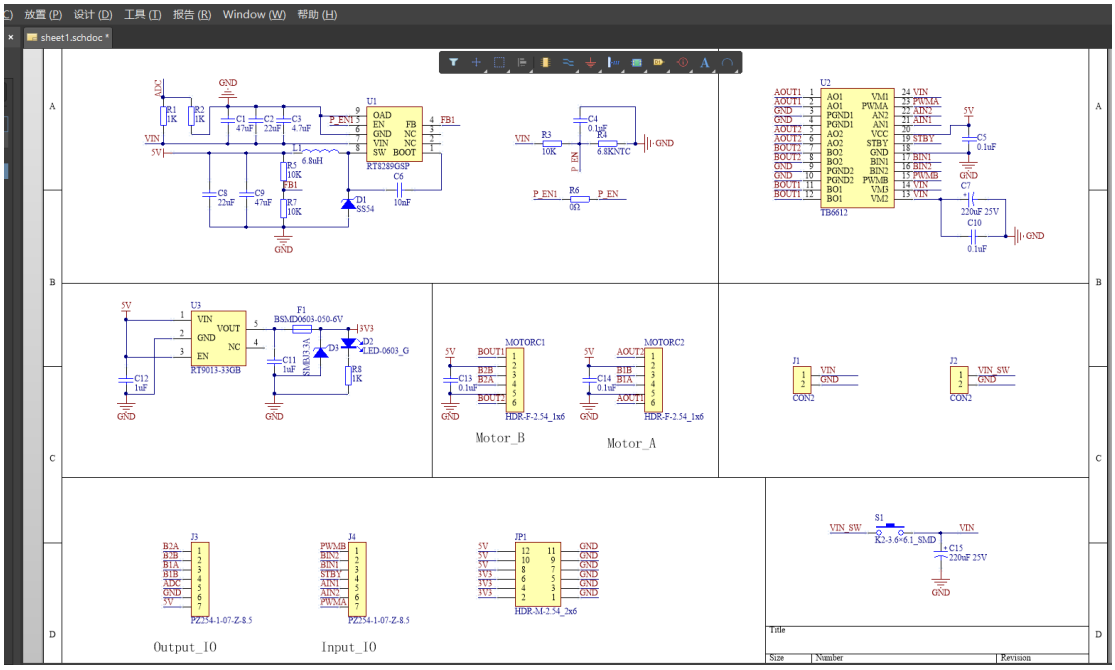


Figure 3.4.6: schematic of PCB

After drawing the schematic, the next step is to make layout.

First, components are placed on the board in their appropriate locations, taking care to ensure that they are positioned correctly and oriented properly. Next, ground and power planes are established, with ground lines being connected to a common ground plane and power lines being connected to a power plane. Signal lines are then routed between components, taking care to minimize crosstalk and interference. Finally, thermal management is considered, with heat sinks and other cooling mechanisms being added as needed to ensure that the board can operate reliably under a variety of conditions. Throughout the process, attention is paid to ensuring that the board layout is optimized for performance, reliability, and manufacturability.

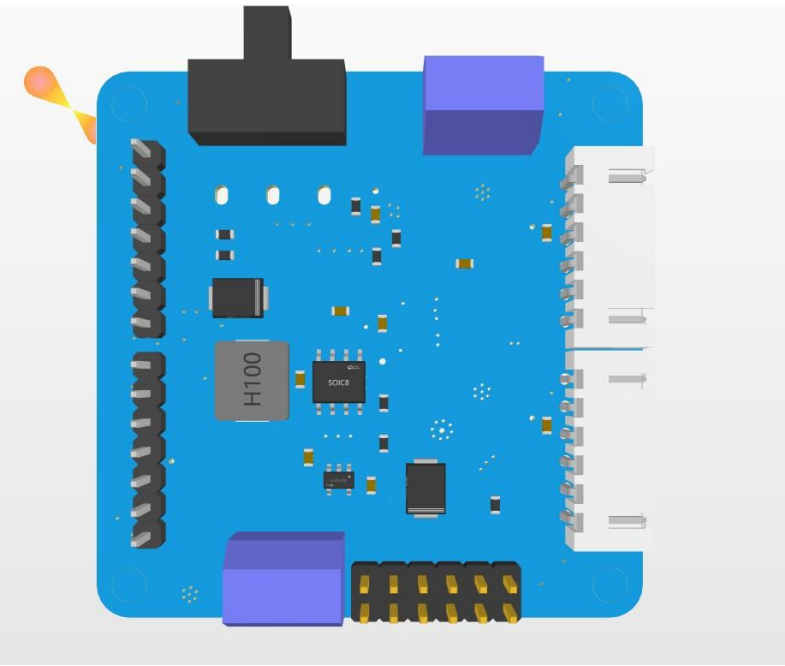


Figure 3.4.7 figure of 3D layout(front)

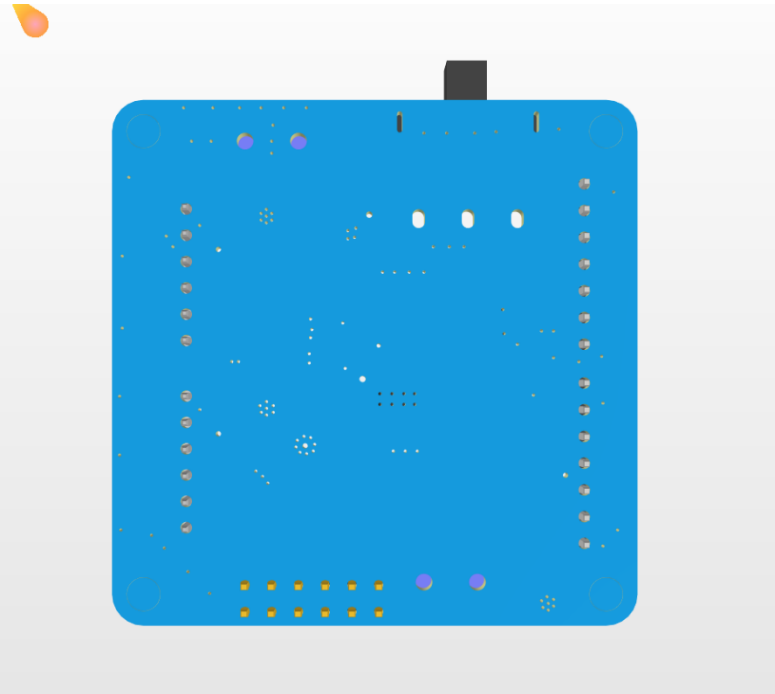


Figure 3.4.8: figure of 3D layout (Reverse side)

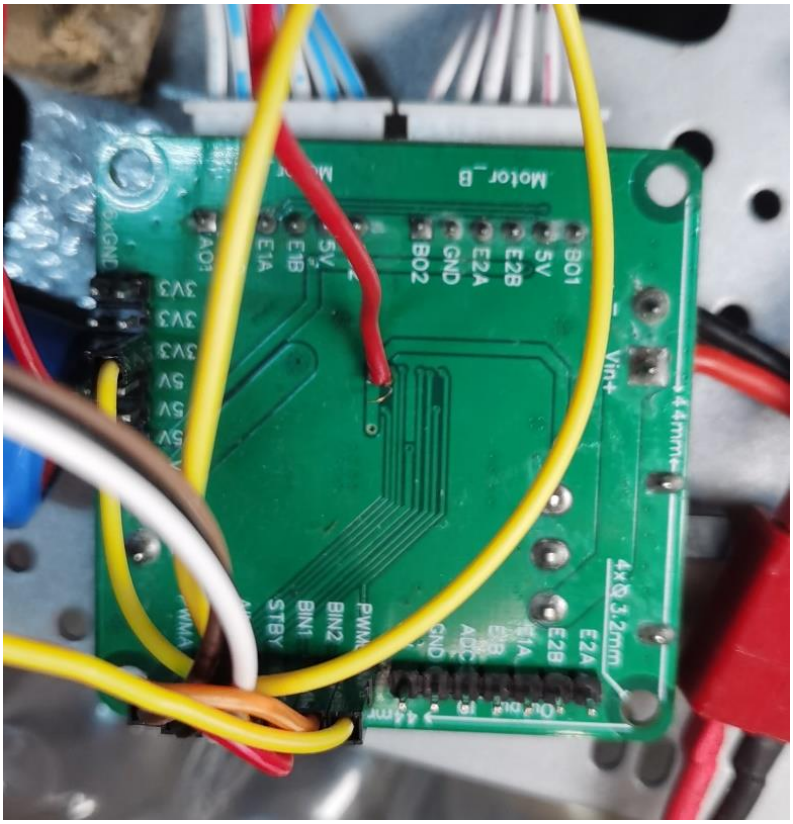


Figure 3.4.9: figure of physical PCB

This PCB board is 53.3mm long and 56.3mm wide, with a power supply range of 4.5V-15V and a maximum internal current of 3A. It is capable of controlling two motors simultaneously, making it ideal for applications requiring dual motor control. The board's compact size and high current capacity make it efficient for our project.

3.4.4 Software Design

Since the software platform the project based on is MicroPython, to enhance the robustness and anti-interference, the software to control the motor is encapsulated in class called *motor*. It controls the movement of the vehicle by control the three pins on each motor, the three pins V_A, V_B and PWM_Signal follows the **Table 3.4.2**.

	Stop	Forward	Backward	Stop
PWM	PWM signal	PWM signal	PWM signal	PWM signal
V_A	0	1	0	1
V_B	0	0	1	1
Speed	0	>0	<0	0

Table 3.4.2: The control of the motor

After the design of separate motor, the motor system of the whole vehicle could be the combination of the whole vehicle. The function encapsulated in the class is as **Table 3.4.3**.

	Velocity Difference	**Function**
fw()	$V_L = V_R > 0$	Move forward / backward
turning()	$V_L \neq V_R > 0$	Turning while moving
circling()	$V_L = -V_R$	Turning in place
pause()	$V_L = V_R = 0$	Stop

Table 3.4.3: The function design of the motor control

3.4.5 Result and Discussion

The system is combined as **Figure 3.7** and tested with different function. The vehicle could move as the command. However, in practical, we met some problem in turning and moving straightly.

Firstly, the motor has initial **speed difference** that makes it hard to move straight. In our practice, the right motor always moves faster than the left motor under the same duty circle of the PWM wave. It is of great importance to make the vehicle to move straight or the angle difference could accumulate as the distance increase to cause the failure in the mission. As a result, we design a PID system to adjust the speed of the motors in real-time so that vehicle could move straightly.

Secondly, the different fraction on different surface could lead to different speed limit. For example, we found that the start duty circle at the marble pavement is 25% while the **duty circle** on the stone area of patio 2 should be 35%. As a result, a proper speed is adjusted on each surface.

Thirdly, the inertia of turning could cause excessive turning. It will threaten the accuracy at each turn and miss the target point so that a PID system will control the vehicle turning slower while it direct closer to the target direction.

These problems are considered and solved so that the motor control system finally became a stable, robust and accurate system for the vehicle.

3.5 Steering Gear Installation and Design (by Wang Zhiyi)

3.5.1 Steering gear principle

In the patio 2, it is required that a table tennis ball should be thrown into a trash bin. In this part, the release device we use is a DIY ball holder which is controlled by a steering gear which could change its angle from 0 to 180 degrees. As a result, the release depends on the angle of the steering gear[8].

The steering gear are chosen to be easily manipulated and it should be stable enough to not open in external interference. As a result, we choose the SG90 which is a popular plastics steering gearing controlled by impulse signal shown in **Figure 3.5.1** The impulse signal to control the angle of the gear is as shown in **Table 3.5.1**

Figure 3.5.1: The pins and control structure on the SG90

Each Pulse is At Least 20 ms Apart		
Impulse Time / ms	Angle / degree	State
0.5	0	
1.0	45	
1.5	90	Close
2.0	135	
2.5	180	Open

Table 3.5.1: The Impulse Time to control the angle of the steering gear

However, considering the noise and interference that could causing incorrect operation, the steering gear should be controlled by a PWM wave with a cycle period of 20ms and controlled by the duty circle. This method is of benefits because the steering gear is activated for all time.

According to the control table **Table 3.5.1**, the practical control method could be transformed to the Table as followed in **Table 3.5.2**.

PWM (Frequency = 50Hz)		
Duty Circle / %	Angle / degree	State
2.5	0	
5.0	45	
7.5	90	Close
10.0	135	
12.5	180	Open

Table 3.5.2: The PWM control table for angles

As a result, the angle follows the equation $Duty\ Circle = 2.5 + \frac{Angle}{45}$. When the vehicle is initialized, the duty circle of PWM signal on the steering gear is set to be 2.5% until it detects the trash bin and then set the duty circle to be 12.5%.

3.5.2 Installation

In order to throw the ball into the bin, there are some challenges. For example, the height of the trash bin is about 30 cm and the vehicle could push away the bin before the ball are drop into the bin. As a result, the extended arm is designed to extend beyond the headstock and the height of the arm should exceed 30 cm. In addition, the ball should be released while moving so that the inertia would bring the ball further into the bin. As a result, the arm is installed as followed shown in **Figure 3.5.2**

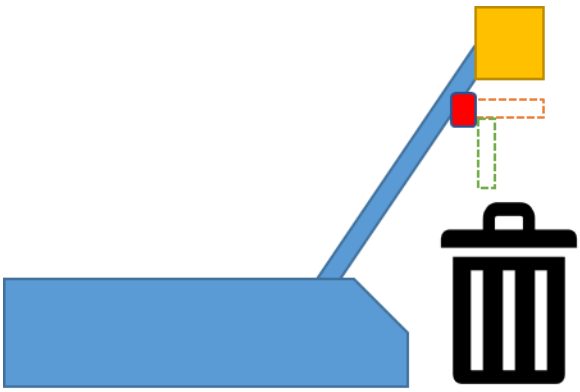


Figure 3.5.2: The design of the release system

3.5.3 Software Design

It is connected to a PWM pin. When the vehicle is initialized, the duty circle is set to be 2.5% to close the holder. When the system moves into the given distance of the trash bin, the duty circle is set to be 12.5% to release the ball. The output is as shown in **Figure 3.5.3**

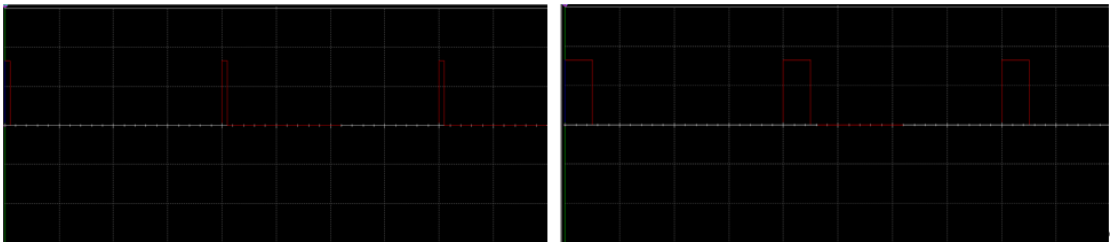


Figure 3.5.3: The PWM wave to close and release the ball

3.5.4 Result and Discussion

The steering gear could hold and release the ball which reach our expectation and requirement. However, the PWM signal could activate the steering gear for a long time that will reduce the lifespan of the gear so the PWM signal could be set to be only activate when needed and using a pull-down resistor.

3.6 Visual recognition – OV2640 (by Bai Tianyou)

According to **Figure 3.6.1**, this project requires the tested vehicle to recognize and analyze the visual messages contained in the testing site, which includes dark trajectory, arrow directions and so on. In order to effectively utilize all the visual information in the testing site, an end-to-end analyzing model was created, where the vehicle can be better guided[9].

3.6.1 Sensor configuration

Although the working environment of our design contains diversified visual features, it is difficult to extract and utilize all these features. On the one hand, these features may be hard to be detected by the camera, for example, the trajectory in patio 1 is very similar with the background. On the other hand, these image signals can be easily interfered by external noises. The light exposure, light reflection can all affect the output of our sensor, as shown in **Figure. 3.6.1**. Under this circumstance, to capture the visual information, the camera of K210 should be properly calibrated and drove.

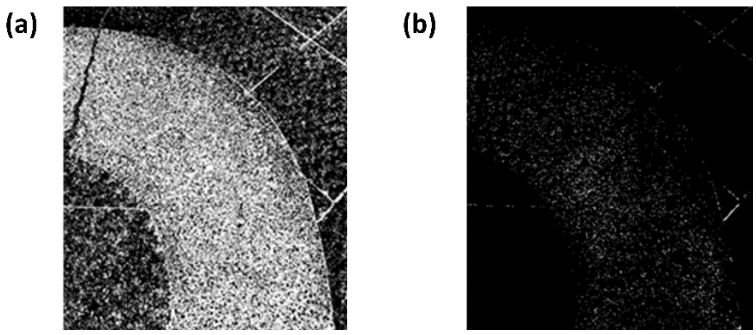


Figure 3.6.1: The outcome in different light exposure

Figure. 3.6.1. The outcome of our algorithm in different light exposure, where (a) shows the normal binary images generated by our algorithm, (b) shows the defective results under an extreme high light exposure.

These calibrations can be conducted on the software system. To begin with, we adopted the sensor APIs in the micropython platform. These APIs allows us to decide the type and quality of images taken by the camera. During the field testing, we found that the track in patio 1 is illegible in grayscale images. In this way, we set the sensor to receive RGB pictures, so that more information can be transferred. Moreover, since the transmission of RGB images occupies large space, we only take one picture out of 3000 snapshots to avoid video buffering in the microcontroller.

Figure 3.6.2: The raw images obtained from the camera

After that, since the picture direction of the camera is wrong as shown in **Figure 3.6.2,** we flipped the picture vertically and then rotated the picture for -90° around z-axel. This

operation allows the program to directly sense the real situation in the test ground. The entire configuration of the image sensor is listed in **Table 3.6.1.**

Image Type	RGB 565
Image Size	[360, 240]
Skipped Frames	3000
Brightness	-1
Auto Gain	False
Vertical Flip	True

Table 3.6.1: The Entire Sensor Configuration of K210 Camera

3.6.2 Image processing

As we mentioned, the images received from the camera may contain various noise, which may lower the accuracy of our recognition. To filter the noise and alleviate the impact of external factors, a further image processing algorithm is necessary. First of all, there are speckles in the tracks, which may interfere the recognition of color. As a solution, we use a Gaussian filter kernel to conduct a convolution on the image, which can smooth the speckles and generates blobs with similar color. Same operation can also be conducted on the arrow image in patio 2, where the different light intensity may disturb the recognition of arrow. With a Gaussian filter, the light intensity at different pixels would be closer.

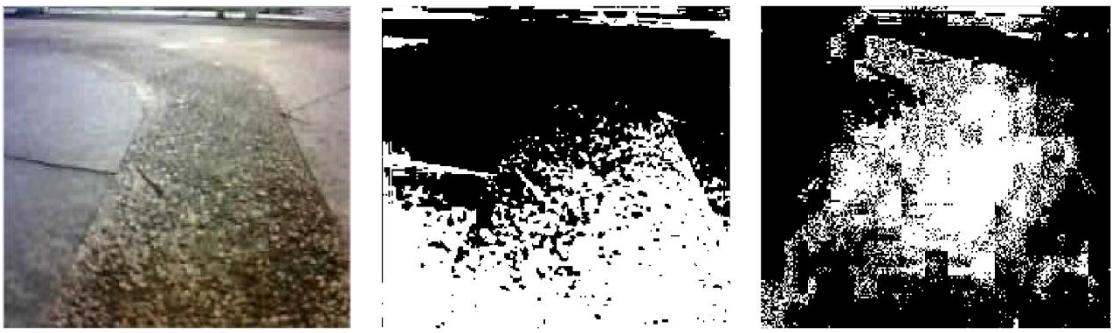


Figure 3.6.3: The images obtained from different steps of image processing

Figure 3.6.3. The images obtained from different steps of image processing, which are the smoothified images, binary image converted from grayscale input, binary image obtained by LAB thresholds, respectively.

After that, we conducted a binary conversion to the input images, which can make the track clearer. The binary conversion can be conducted under two different thresholds, the grayscale threshold and the LAB threshold. The grayscale threshold requires less calculation, while the LAB threshold has a more detailed division. During the field testing, we found that the grayscale threshold is not enough to clearly divide the original images as shown in **Figure. 3.6.3**, therefore, we adopted the LAB threshold to convert the RGB images to binary images. What should be noticed is that this LAB threshold can be affected by external factor as well, which requires us to change the threshold according to the weather and light exposure on the testing day. The binary images generate by this block would be shown as **Figure 3.6.4.**

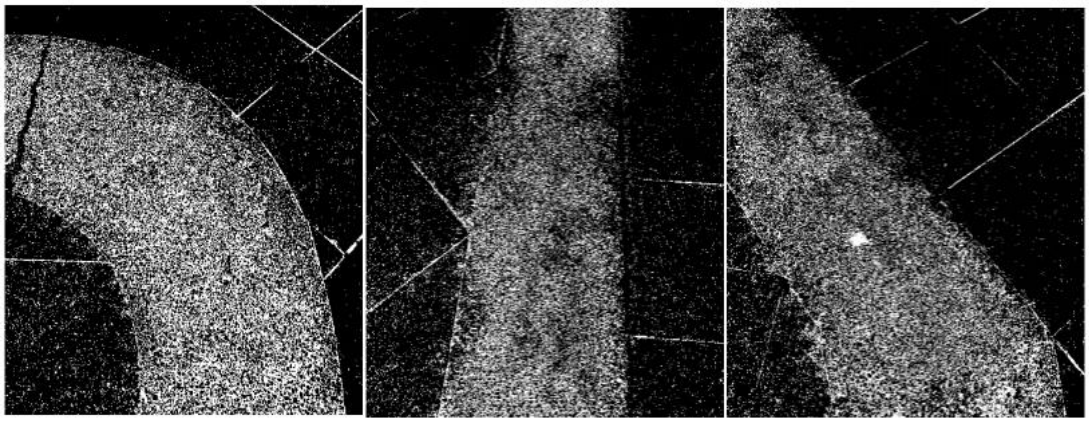


Figure 3.6.4: The binarization result of the image processing block

3.6.3 Auto-Tracking algorithm

According to the requirements in patio 1, the auto-vehicle should be able to distinguish the dark trajectory from the background and decide the speed of two wheels. To fulfill these tasks, the auto-tracking algorithm is constructed as shown in **Figure 3.6.5.**

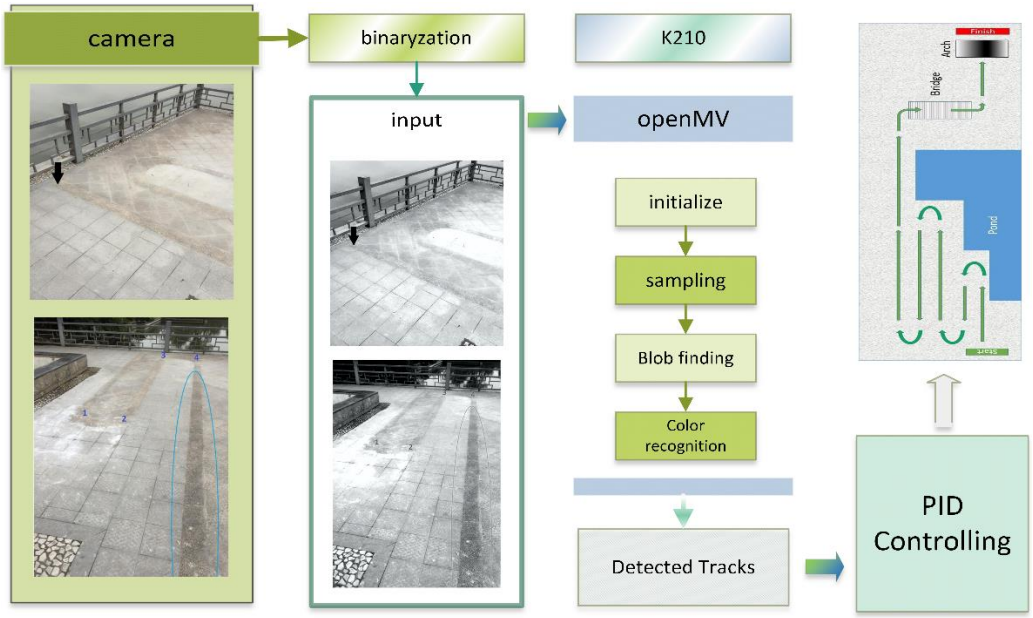


Figure 3.6.5: The structure of the auto-tracking algorithm in patio 1

As shown in Fig. 3.6.3, this model first invokes the binary images generated by image processing module. Then, a color recognition was conducted to find the dense white color blobs. Considering that only the blobs in the central part of image can affect the direction of the vehicle, we set the region of interest (ROI) at the middle height of the image, which is 120. According to this ROI, the we only search for white color at a fix height with "**find_blob**" function, as shown in **Figure 3.6.6**.

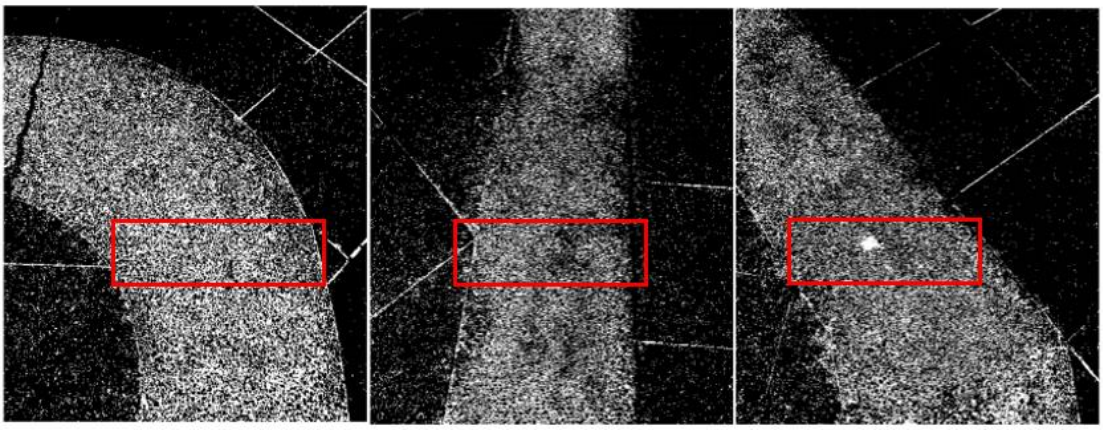


Figure 3.6.6: The detection region of the binary inputs

The **find_blob** function divides a series of blobs with white color, and generates a "**blob**" type of output. This output not only shows the divide region of object color, but also contains the statistical features of the division results. Based on that, we can know the position (p_i,120) of the central point of the blobs, where p_i is the abscissa value of the central point. Since the **find_blob** function may return multiple color blobs, we calculate the average value of the abscissa values of each blob to represent the entire central points of all blobs, which can be calculated as Eq. (1).

$$\overline{p_i} = \frac{1}{N}(p_1 + p_2 + \cdots + p_N) \tag{1}$$

The central point of the entire image is certain, whose position is (160, 120). The distance of two central points can be calculated as Eq. (2)

$$e_d = \overline{p_i} - p_o \tag{2}$$

where p_o is the abscissa value of the image central point. By comparing the blob central points and the image central points, we can derive the moving direction of the auto-vehicle. For example, if the blob is at the left of the image, the vehicle should move left; if it is on the right side of the image, the vehicle should move right.

Once the direction can be decided, a PID module was inserted to calculate the exact value of left wheel speed and right wheel speed. The PID algorithm, which stands for Proportional-Integral-Derivative, is a control loop feedback mechanism widely used in engineering and industrial applications to regulate and stabilize systems. It calculates an output value based on the error between a desired setpoint and the actual process variable. In this case, the error would be the distance of two central points e_d. The PID algorithm incorporates three main components, proportional (P) term, integral (I) term and derivative (D) term.

The P term produces an output that is directly proportional to the current error. It adjusts the control variable in proportion to the deviation between the setpoint and the actual value, as shown in Eq. (3).

$$y_p = K_p \cdot e_d \tag{3}$$

The P term provides a fast response to reduce the error, but it may lead to overshooting and oscillations.

The I term takes into account the accumulated error over time. As shown in Eq. (4), it

sums up the errors and continuously adjusts the control variable to eliminate any long-term steady-state error.

$$y_i = K_i \cdot \sum e_d \tag{4}$$

The I term is effective in addressing offset errors, but it can introduce instability if not properly tuned.

$$y_d = K_d \cdot [e_d(k) - e_d(k-1)] \tag{5}$$

The D term considers the rate of change of the error. According to Eq. (5), it predicts the future trend of the error by analyzing its rate of change. The D term helps dampen the system's response, reducing overshooting and improving stability.

The output of the PID controller would be calculated as Eq. (6).

$$y = y_p + y_i + y_d = K_p e_d(k) + K_i \sum_{k=0}^{T} e_d(k) + K_d [e_d(k) - e_d(k-1)] \tag{6}$$

Here, K_p, K_i, K_d are factors that need to be well tuned, y is the difference of left wheel speed and right wheel speed, T is the calculating epochs of this algorithm. In this way, we can get the speed of two wheel as Eq (7) and (8).

$$v_l = v_s + y \tag{7}$$

$$v_r = v_s - y \tag{8}$$

Here, v_l, v_r are the left wheel speed and right wheel speed respectively, v_s is the fixed speed, which is 0.4 as we set[10].

Algorithm 1: auto-tracking algorithm for patio 1

Input: s, r, $\overline{p}_i$, p_o
Output: v_l, v_r
Set K_p=0.06, K_i=0.001, K_d=0.07, $drop$=0.1
Initialize S=30, T=20, t=0, v_s=0.4, e_d=0, $e_b = e_n$=0, $sum = d$=0
Get s, r from radar on the front and right side of the vehicle
while r>30:
    if s > 30 **do**
        upgrade $\overline{p}_i$, p_o
        Calculate $e_d = \overline{p}_i - p_o$
        while t < T **do**
            $e_b = e_n$, $e_n = e_d$
            sum += e_d, $d = e_n$ - e_b
            $y = K_p \times e_d + K_i \times sum + K_d \times d$
            $v_l = v_s + y$, $v_r = v_s$ - y
            Motor Controlling
            $\overline{p}_i$ += y - $drop$
            $t = t + 1$
        end while
    else do
        turn(270)
        forward(v_s)
end while

Table 3.6.2: The PID algorithm

The entire procedure of patio 1 is described in **Algorithm 1**. What should be noticed is

that in order to guide the vehicle to the bridge, a sonar at the front of the car is adopted. We set two beacons before the entrance and exit of the bridge. Once the sonar detects the first beacon, the auto-tracking program is stopped and the vehicle turns right to move across the bridge. After that the second beacon guides the vehicle to head to the gate.

3.6.4 Arrow recognition

In patio 2, the vehicle is required to recognize the direction of arrow, and decide which direction should be turning. As shown in **Figure 3.6.7**, the arrow direction contains left, up and right[11].

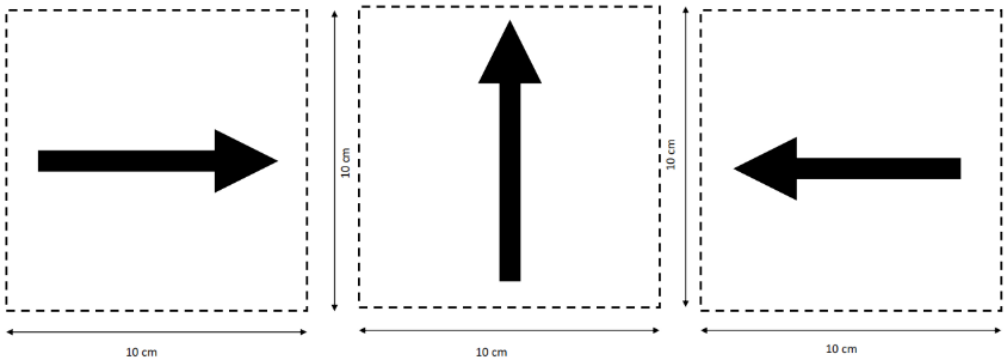


Figure 3.6.7: The directions of arrows

To recognize the direction of arrows, we constructed a neural network to study the correlation between images of arrows and their directions. The neural network is designed according to MobileNet, which is light-weight, accurate and suitable for microcontroller operations[12].

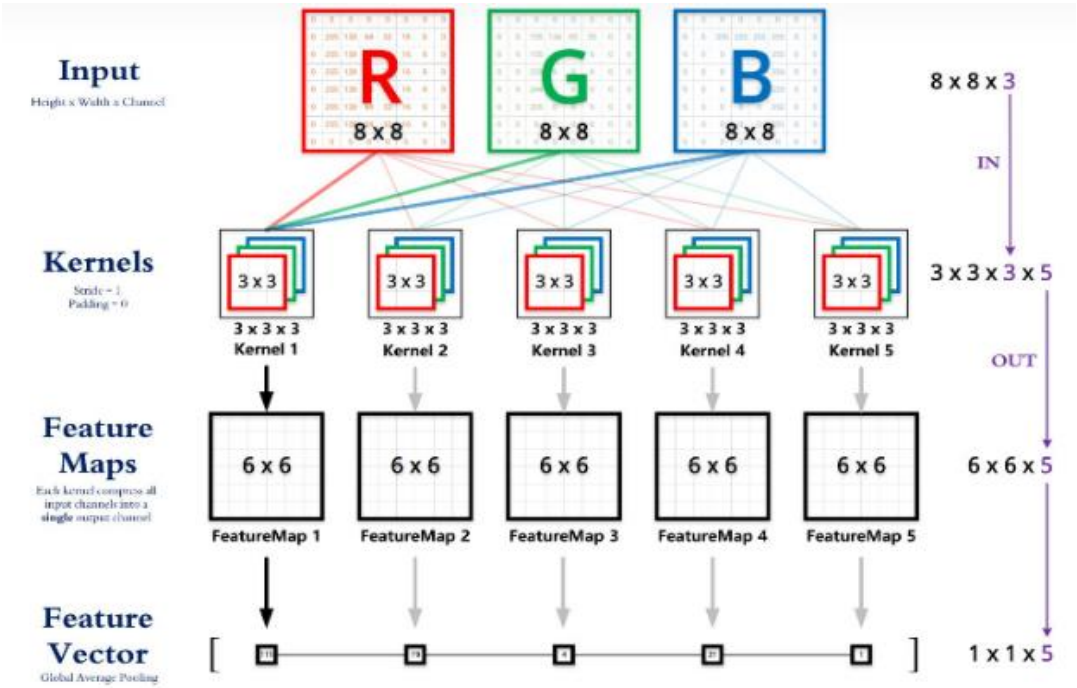


Figure 3.6.8: The structure of MobileNet

As shown in **Figure 3.6.8**, the MobileNet is a lightweight convolutional neural network (CNN) that can be imported to the microcontroller of K210. This network is composed of

3×3 CNNs and 1×1 CNNs. The 3×3 CNNs is a down sampling mechanism, which reduces the computational cost and refine the learning of interested features. The 1×1 CNNs lower down the complexity of the parameters and optimizes the training process. The mobile net block receives the images from camera and classify the images to certain class. Then the PID controller would react to the classification result and change the course angle of the auto-car.

Based on this model, we collected the pictures of arrows in different directions, and constructed a dataset to train and test the model. After training the weight document was uploaded to the microcontroller. When the vehicle stops in front of the arrow, the camera will take on snapshot of arrow, and send it to the MobileNet. The arrow was then classified and tells the vehicle which direction should it be turning.

3.6.5 Result and discussion

In order to evaluate the performance of our program and improve the visual mode, we conducted a series of testing on the real working environment.

a) The tracking performance of patio 1

To test the tracking ability of the vehicle, we put the car on the testing field on a clear day with sufficient sunlight. Without human interruption, the vehicle is allowed to move automatically. As shown in **Figure 3.6.9**, the auto-vehicle is capable of recognizing the dark trajectory and moving along the path.

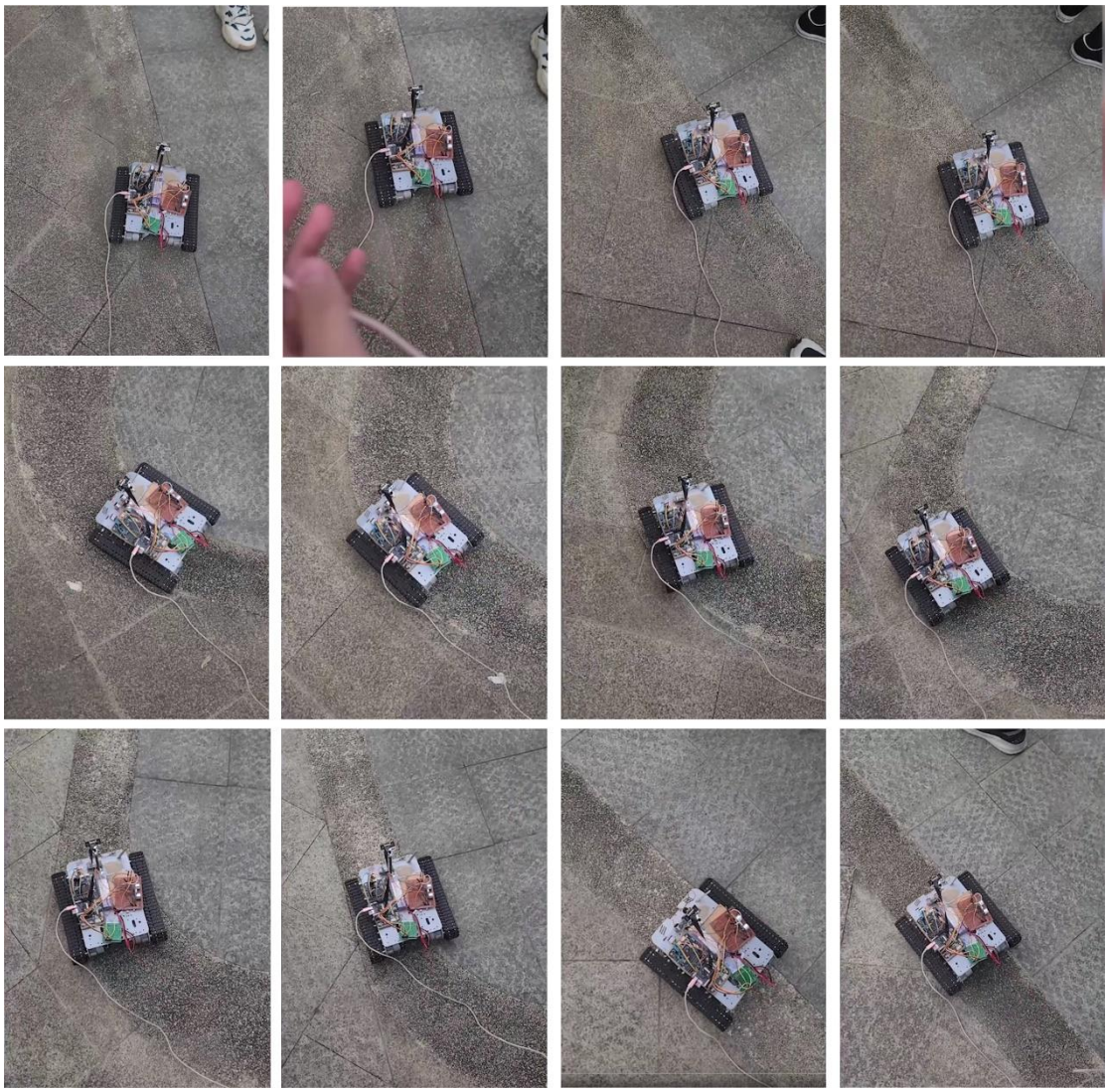


Figure 3.6.9: The moving track of the auto-vehicle

As is illustrated in **Figure 3.6.9**, the vehicle is proven to be able to automatically track the path. It moves straightly on straight lines, and successfully turns all the turns. During the testing, we found that the color of the path is significant for auto-tracking, when the path has a slight color, the vehicle turns very bumpy. Moreover, the angle of the camera is very important, the camera should be as much close to the ground, so that more information can be captured.

b) The classification of arrow directions

To evaluate the classifying accuracy of the MobileNet, we created a test set containing 30% of the arrow images to simulate the actual working of the program. There are 3 classes in the test (left, up and right), and the model is required to classify the images by learning the correlations between pixel values and class label. The accuracy is shown in **Figure. 3.6.10**.

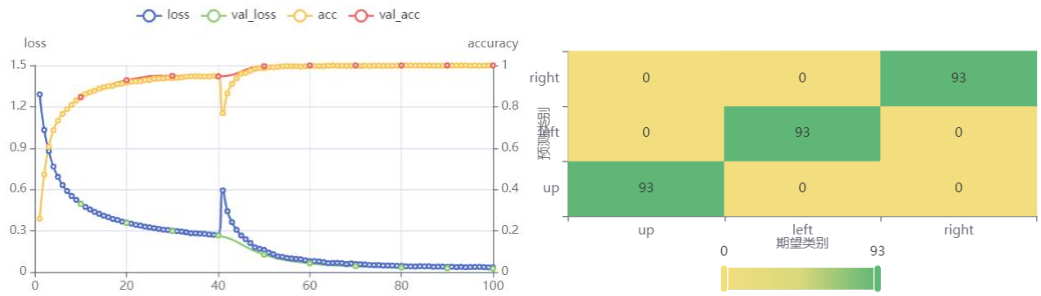


Figure 3.6.10: The accuracy of the classifying model

As is demonstrated in **Figure 3.6.10**, the MobileNet can reach an accuracy of 98.9%, and for each case the accuracy would be more than 93%, which is sufficient for the actual detecting task.

3.7 Gyroscope – JY60 (by Liu Weixing)

A gyroscope is a device or sensor used to measure and maintain the Angle and direction of an object. It works based on the principle of gyroscopic effect, using the rotating gyroscope to maintain its own stability, when the gyroscopic wheel rotates[13], due to the principle of conservation of angular momentum, the axis of the gyroscope will maintain a relatively fixed direction. By measuring the rotation speed and direction of the gyro wheel, the angular change and direction of the object can be determined.

Gyroscope plays an important role in the smart vehicle. It is a device used to measure and maintain the Angle and direction of an object, and can sense the attitude and steering of a vehicle. The smart car uses a gyroscope to monitor steering Angle and steering speed for precise steering control. The gyroscope can provide real-time directional information to help the smart car maintain a smooth and accurate path when turning.

3.7.1 JY60 parameter

Various JY60 related parameters including Angle parameters, interface parameters and electrical parameters are described below[14].

Parameter	Condition	Typical Value
Measuring Range		Z: ±180°
Course Accuracy	Six-axis algorithm, static	0.5°
Resolution	Horizontal placement	0.0055°

Table 3.7.1: Course Angle parameter

In **Table 3.7.1**, the cruise Angle parameters of JY60 are introduced. In fact, in addition, JY60 also has data such as triaxial acceleration, triaxial Angle and triaxial speed. However, since the car only needs to walk on a plane, the data selected is only the yaw Angle in the Angle[15]. As you can see, the JY60 has a high accuracy, and coverage of the global range.

Parameter	Condition	Minimum	Default	Maximum
Communication interface	UART	9600bps	9600bps	9600bps
Output content		Triaxial acceleration, angular velocity and angle		
Output rate		20Hz	20Hz	20Hz
Working temperature		-40℃		85℃

Table 3.7.2: Communications parameters

Table 3.7.2 includes some communication parameters. First of all, the working environment of -40℃ to 85℃ is very comfortable, which ensures the stability of JY60. JY60 can adopt UART communication or I2C communication, and the baud rate is 9600. In this experiment, UART is used to send messages in order to obtain a more stable rate of return[16].

To ensure that the rotation Angle can be correctly detected, a high enough output rate is necessary, the JY60 has a 20Hz output rate, which means that 20 different sets of information can be returned to the microcontroller every second, for the smart car with a slow rotation speed, such an update rate is very sufficient.

Parameter	Condition	Minimum	Default	Maximum
Working Voltage		3.3V	5V	5.5V
Working Current	Work		10 mA	
	Idle		10 uA	

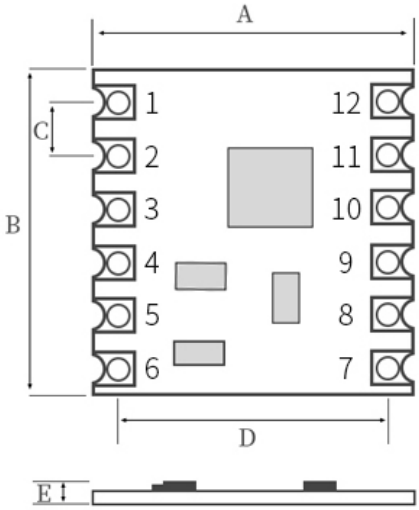
Table 3.7.3: Electric parameters

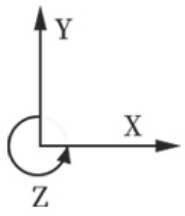
In **Table 3.7.3**, we learned some electrical properties of JY60. The working voltage of 3.3V is enough to drive JY60, so we finally chose the scheme of using 3.3V pin power supply on the core board.

Figure 3.7.1: The scope of JY60

3.7.2 JY60 drive

As shown in the following figure, JY60 has multiple sets of pins, which can support UART and I2C protocols. In the actual operation process, we adopt UART communication mode, that is, connect VCC, GND, RX and TX four pins. Note that RX and TX need to be relatively connected on the gyroscope and board.





Units:mm

	Size
A	15.24
B	15.24
C	2.54
D	12.7
E	2

Figure 3.7.2: JY60 product size

JY60 follows the WT61 protocol, which is explained as follows:

Data Read:

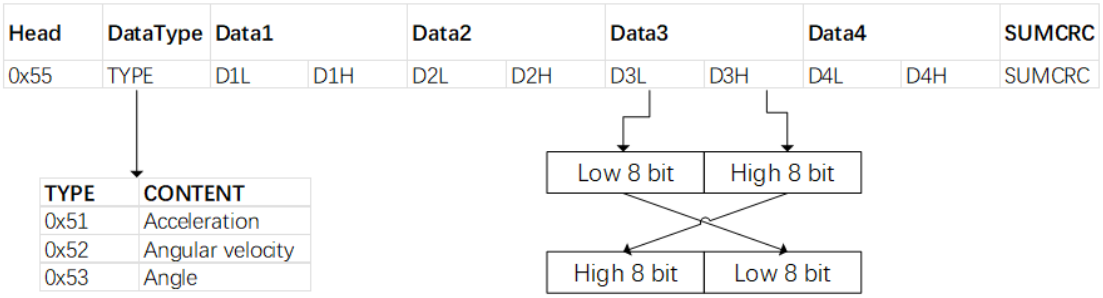


Figure 3.7.3: Brief introduction of WT61 data read

Data Write:

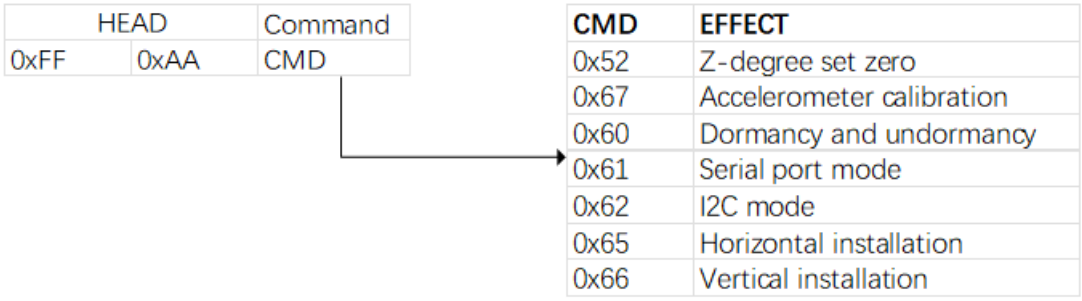


Figure 3.7.4: Brief introduction of WT61 data write

3.7.3 Software design

Since we only need yaw Angle data, after taking out the yaw Angle data in the program design, it is packaged into a class combined with motor motion: class gyroscope()

When the yaw Angle data is obtained, it is encapsulated into a class named "gyroscope" with the following methods:

1. **def get_angle()**: Used to get the current Angle easily. The method can call the sensor to get the latest yaw Angle data and return that value as a result.
2. **def set_zero()**: Set the current direction to zero. This method can set the current Angle to zero so that subsequent Angle measurements are made relative to this baseline.
3. **def turn(direction)**: Used to turn to the specified direction. The method accepts one parameter, direction, indicating the target direction to turn. According to the target direction and the current Angle, the method can calculate the Angle that needs to be rotated, and realize the steering by controlling the motor.

There will take function “turn(direction)” as an example, when it calls the function, it will firstly find **which direction** and **how many degrees** it should turns.

The process is repeated until the direction reversal, it stops.

3.7.4 Analysis and Discussion

In the actual production process, we encountered the problem of inaccurate scale, and later found that the gyroscope was not fixed, and failed to meet the requirements of horizontal placement, so that the Angle data he obtained was biased.

At first, we used 5V power supply according to the instruction manual, and then the internal wiring of the gyroscope burned out, we replaced the gyroscope and used 3.3V, although it was different from the description in the manual, but this is a very effective experience, which told us that it is best to experiment from a lower voltage.

In addition, in the process of controlling the motor steering, there is a speed too fast, resulting in inertia, in order to solve this problem, we make it in the process of rotation, the closer to the target Angle, the smoother the deceleration, which can reduce the inertia brought by the problem of over rotation.

3.8 Ultrasonic Radar for Distance Measurement (by Wang Zhiyi)

3.8.1 Ultrasonic Radar HC-SR04

In some complicated circumstance, the visual sensor is not stable and robust enough to finish all the mission especially in some complex background and variable lighting conditions. In these cases, increasing accuracy could burden the small microcontroller and waste more time and cost. As a result, one of the popular ways is using the ultrasonic radar for distance measurement to obtain the environment information[17].

Figure 3.8.1: The HC-SR04 structure

HC-SR04 is a budget radar for a smart vehicle. The parameter table is shown in **Table 3.8.1**

Operating Voltage	DC 5V
Operating Current	15 mA
Operating Frequency	40 KHz
Max Range	4 m
Min Range	2 cm
Ranging Accuracy	3 mm
Measuring Angle	15°
Trigger Input Signal	10µS TTL Impulse
Dimension	45 x 20 x 15 mm
Price	<1 £

Table 3.8.1: The performance index of the HC-SR04

It is shown that the radar meets our expectation and requirement. Moreover, the radar is cost-efficient.

3.8.2 Control of the Radar

It is known that the speed of the sound wave is 334m/s. Considering the speed would increase with the temperature of the environment, the sound of the sound wave can be calculated by the equation.

$$V_s = 331.5 + 0.607T\ (m/s)$$

There are 4 pins on the radar module: Vcc, GND, Trig and Echo. The Vcc and GND will provide a 5V power supply while the Trig send TTL command for data and then the Echo will return an echo high level signal whose length is proportional to the distance. A typical cycle for the distance measurement is as shown in **Figure 3.8.2**.

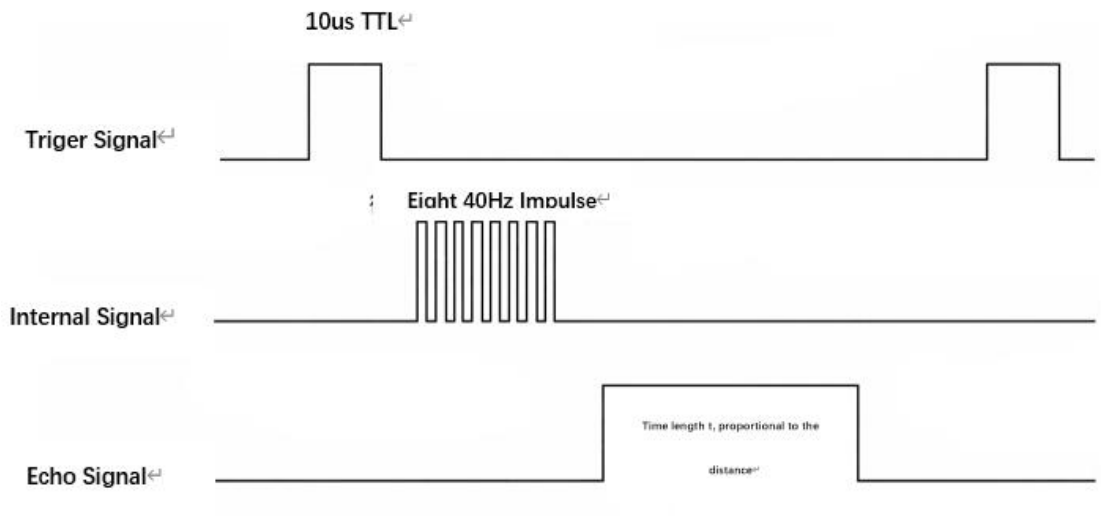


Figure 3.8.2: A cycle of distance measurement

When the time duration t is measured, the distance can be calculated as the following equation:

$$d = \frac{V_s \cdot t}{2}$$

As the temperature will not significantly change in a short time, the sound speed V_s is set to be a constant 334m/s.

3.8.3 Software Design

The radar is encapsulated in to a class that could be called by all the other functions. It could send the impulse to the radar and receive the echo t tance and return it to the program. This function is widely used in the specific sections in both patios. Two separate radars are used and controlled for different circumstance and assist the function of visual sensors.

Since some accidental error could interfere the movement, we design a system using queue and median filter to filter out the errors. The pseudo code is shown as below.

```
while True:
    distance <- radar
    queue <- distance
    if median(queue) > threshold:
        Next Step
    else:
        continue
```

3.8.4 Result and Discussion

The result perfectly met our expectation and requirement. The error is ignorable since the distance to detect is relatively low and the surface is flat and smooth. The radar worked well from 5cm to 400cm which is suitable for most of our target shown in **Figure 3.8.4**.

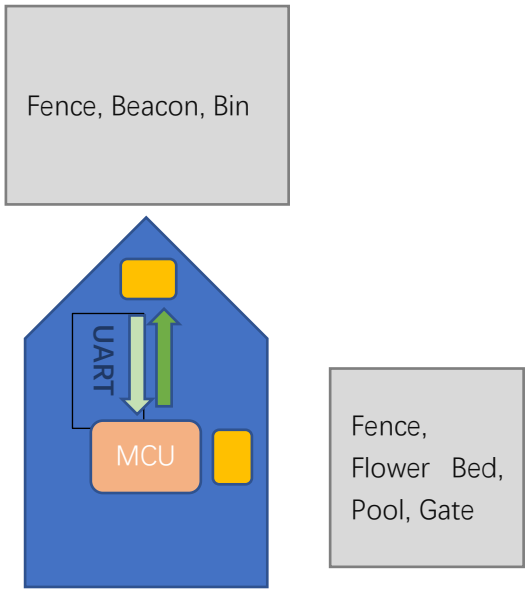


Figure 3.8.3: The target to detect for the two ultrasonic radars

Especially in patio 2, the movement logic along the fence could be as shown in **Table 3.8.1** only based on the radar sensor.

However, there are also several challenges that interfere the detection.

Firstly, the height of the fence on along the lake is hollow so that if the radar is not on the same height to the base of the fence, there will be no echo to be detected. As a result, we design a base for the radar at the same height of the fence or in the 15° area in the radar detection area as shown in **Figure 3.8.4**.

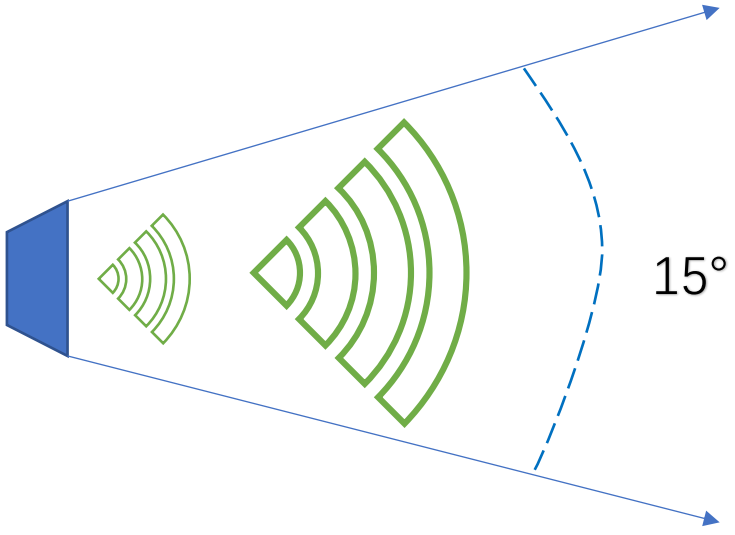


Figure 3.8.4: The detectable area of the radar

Secondly, the slope could also avoid reflection of the sonic wave. For example, the slope on the bridge in patio 1 avoids the detection by the radar as shown in **Figure 3.8.4**. As a result, a beacon is needed to hint the vehicle to stop and turning to the bridge.

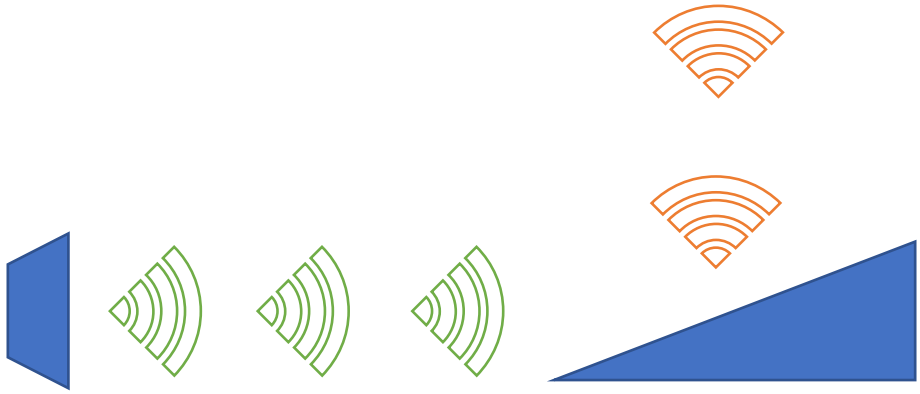


Figure 3.8.5: The distance error caused by slopes

Thirdly, the hollow trash bin without trash bag could let the sonic wave pass without reflection as shown in **Figure 3.8.5**. In this case, beacon or the pressure sensor are needed. In this project, the trash bin is in the fixed position so the time could be set to reach the predetermined location.

Figure 3.8.6: The low reflection rate of hollow trash bin

3.9 Communication and Clock (by Fan Xilai)

The goal of the wireless communication part, which belong to task 3 of patio 2 is to realize communication between robot and laptop. Specifically, when the robot enters the planter area after throwing the ball, the robot should stop and transmit a message including team information and time information to a laptop.

HC-12 wireless communication module is used as required. UART only support to transmit character which obey the rule of ASCII code or 8-bit binary code, the question is how to transfer the time of day (24-hour clock). As there is no component that can provide accurate time information yet, a simple but effective clock module named DS1302 was applied.

3.9.1 UART Communication

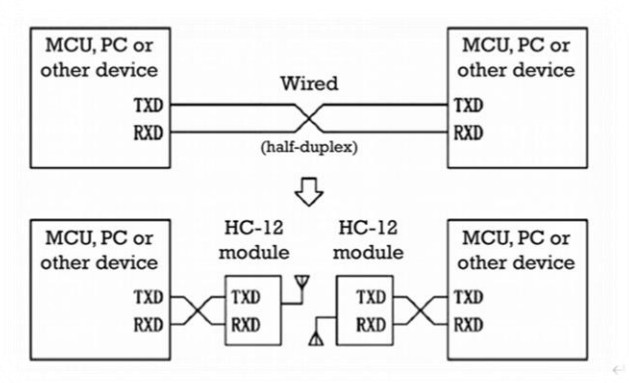


Figure 3.9.1: The communication principle for UART

In the HC12 module, UART, as a communication interface, is responsible for packaging the data from the microcontroller into the data format specified by the UART communication protocol and sending it to the communication device. At the same time, it can also receive data from the other device of the communication and send it back to the microcontroller for processing after analysis.

UART is a serial communication protocol commonly used to transfer one or more bytes of data from one terminal to another. It can transmit asynchronously or synchronously.

UART uses two lines for communication, namely TX (Transmit) and RX (Receive). When one device needs to send data to another device, it sends the data to the TX line, and after sending, the other device receives the data through the RX line. In the process of sending and receiving, data is packaged and parsed by information such as start bit, data bit, check bit and stop bit.

3.9.2 HC12 Parameter and Drive

The HC12 wireless UART communication module is an embedded data transmission module which can be used between two devices. HC12 integrates a programmed MCU, with an attachable spring antenna for signal acceptance, which was demonstrated in Figure 3.9.2.

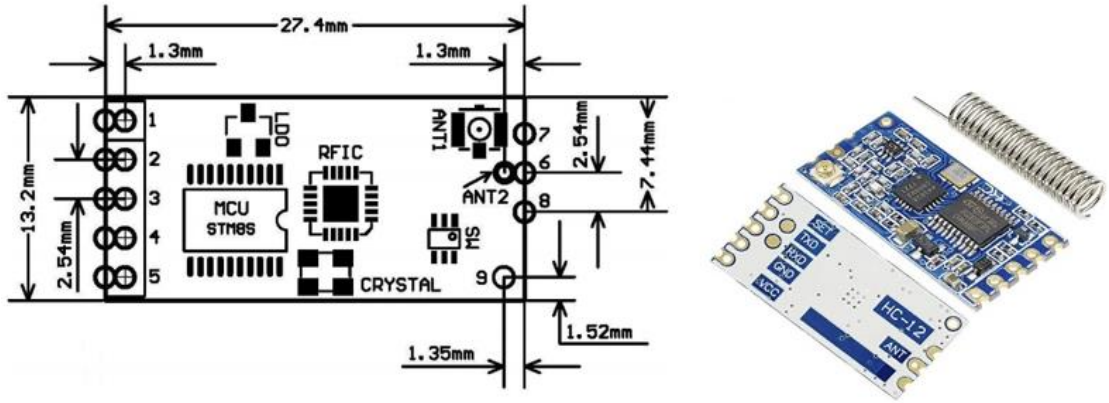


Figure 3.9.2: Dimension Diagram and Physical Appearance of HC12

1. 3.2V-5.5V DC power supply, 5V in operation manual， finally used 3.3V

2. 200mA working current at least.

3. 433.4MHz-473 MHz for Operating frequency range, up to 100 channels of communication

4. The maximum transmit power: 100mW (settable)

Pin 1 is defined as VCC, and pin 2,7,8 as GND. The input (RXD) and the output (TXD) pins are pin 3 and 4, connecting with the internal power supply and the external voltage source respectively. Additionally, there are 4 UART transmission modes for HC12, FU1 to FU4,

mode	description
FU1	Power saving mode, less working current.
FU2	Super power saving mode, only suitable for transmitting a small amount of data.
FU3	Default mode.
FU4	Long range communication mode.

Table 3.9.1: Different modes for HC12

It is essential to synchronize modes of two communicating HC12 modules, and the present mode is displayed on AT command.

The working drive of HC12 module is achieved through the following steps:

1. Configure the serial communication parameters, including baud rate, data bits, parity, and stop bits, to communicate with the external master controller.

2. Configure the working mode of the HC12 module, including frequency hopping mode and fixed frequency mode. In frequency hopping mode, the module will automatically hop frequencies to avoid interference and conflict. In fixed frequency mode, the module will not hop frequencies and will maintain a fixed working frequency.

3. Data transmission, connect to the external controller via serial port, write the data to be transmitted into the HC12 module's buffer, and conduct the corresponding data processing and decoding operations.

4. Transmit data, send the processed data signal to the receiving end. Before transmission, it is necessary to set the transmission power and transmission frequency parameters to ensure that the transmitted data signal can be received by the receiving end.

5. Receive data, the received data signal is decoded and processed, and then written into the HC12 module's receiving buffer.

6. Data validation and error handling. During data transmission and reception, it is necessary to check and handle the validation and errors to ensure the accuracy and reliability of data transmission.

A pairs of wireless communication modules are needed to communicate between the vehicle and laptop as transceivers.

3.9.3 DS1302 Clock Parameter and Drive

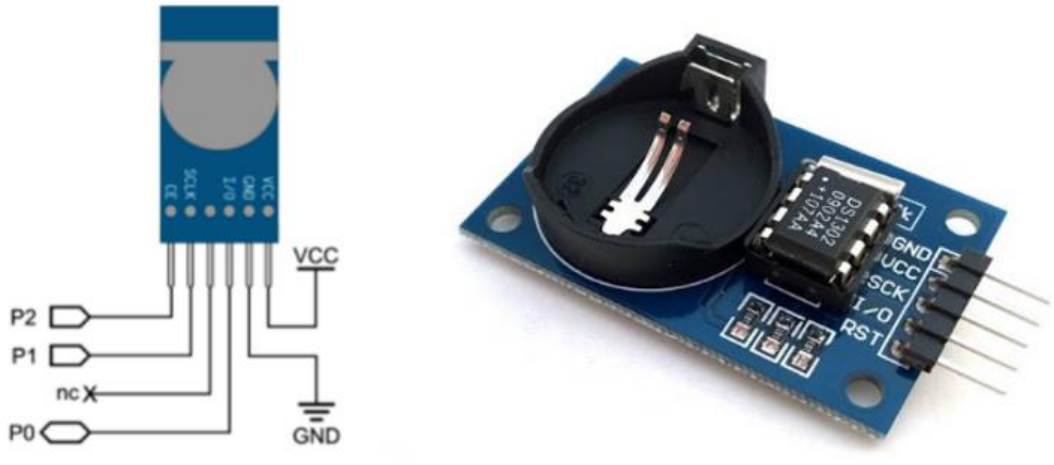


Figure 3.9.3: Pin diagram and physical appearance of DS1302

The DS1302 module can provide detailed time information on seconds. The module has eight pins, but only five pins need to be connected. The VCC pin connects to the current source, and the GND pin is grounded. The CE pin (pin 5) must receive a high input signal during data reading and writing. This pin has two functions: first, it controls the word access shift register control logic; second, it provides a way to end single-byte or multi-byte data transfer. The I/O pin (pin 6) is a two-way communication pin that reads and writes data. The SCLK pin (pin 7) has a serial function.

The working parameters of this module are as follows:

1. Operating voltage: 2V to 5.5V
2. Operating temperature range: -40℃ to +85℃
3. Clock accuracy: ±2℃ / month (at 25℃)
4. Clock frequency: 32768 Hz
5. Communication interface protocol: 3-wire serial interface protocol

The working drive of DS1302 module is achieved through the following steps:

1. Initializing the DS1302 clock module by writing values to the control register and RAM register to set the clock and date.

2. Performing data read or write operations, completing data communication between the controller and DS1302 by using input signals such as CS, CLK, and DIO.

3. The DS1302 clock module calculates the current date and time through the clock circuit and counter circuit and outputs it to the data line.

4. Interrupt output: DS1302 interrupt output can be configured to listen to the clock cycle through the interrupt pin. When the clock cycle is completed, the interrupt pin outputs a pulse for external system response.

3.9.4 Software Design

1. Program the DS1302 timing module, which serves as an additional module to provide time information, to output time data.

2. Package the time monitoring program to improve its reusability, facilitate calling and integration.

3. Program the HC-12 module to obtain time data from the DS1302 after receiving microcontroller instructions and establish a communication connection with a laptop computer terminal.

4. Package the wireless communication program to minimize its impact on global variables, thus reducing its impact on other tasks. This improves the program's reusability and facilitates its calling and integration.

3.9.5 Result and Discussion

The wireless communication was found to perform very well, effectively meeting the requirements. However, several aspects of the communication were not explored in depth during the testing process.

Firstly, the experiment did not attempt to assess the communication performance of the other available communication modes of the HC-12 module. In order to establish the communication between the vehicle and the computer as soon as possible, most parameters are set by default to reduce the time required for adjustment and avoid possible errors

Secondly, the experiment did not consider potential issues of obstructions in the wireless communication path, which could affect the performance of the wireless communication.

Additionally, channel congestion was not explored. During the testing, all HC-12 devices were working in the same environment. As a result, a lot of noise was created on the channel. Therefore, we group only changed to an idle channel, without addressing how noise could be effectively managed.

Furthermore, due to budget constraints (A single microcontroller can take up half the budget), the project did not consider how the HC-12 module could transmit time information if the microcontroller had an inbuilt clock. This could have significant implications in simplifying the circuit of the overall system and could be an important consideration for similar projects in the future.

In conclusion, while the wireless communication achieved through the HC-12 module was successful, there remain several areas that require further exploration and development. Future experiments should consider exploring channel congestion and strategies for noise management. Additionally, the project should consider how the HC-12 module can transmit time information with the clock that comes with the microcontroller.

3.10 PID (by Bai Tianyou)

During the field testing, we found multiple occasions that require frequent attitude correction to ensure the operating accuracy of the vehicle. To achieve that, PID controllers were adopted in many sections of the auto-vehicle system[18].

The PID (Proportional-Integral-Derivative) controller is a widely used feedback control algorithm that is employed in various control systems to regulate a process variable. The PID controller continuously calculates an error signal ε by comparing the desired setpoint value with the actual measured value of the process variable. The controller then adjusts the control output based on this error signal to minimize the deviation between the setpoint and the actual value.

The PID controller can be described in three main steps: proportional control (P), integral control (I) and derivative control (D). The P controller calculates an output value that is proportional to the current error. The output is obtained by multiplying the error by a constant factor known as the proportional gain (K_P), as shown in Eq. (9).

$$Y_P = K_P \cdot \varepsilon \tag{9}$$

The I controller calculates an output value based on the integral of the error signal. The integral action helps in eliminating steady-state errors and reducing the cumulative effect of small errors. It is achieved by integrating the error signal over time and multiplying it by the integral gain (K_I), as shown in Eq. (10).

$$Y_I = K_I \cdot \int \varepsilon \ dt \tag{10}$$

The D controller generates an output value based on the rate of change of the error. It differentiates the error signal with respect to time and multiplying it by the derivative gain (K_D), as shown in Eq. (11).

$$Y_D = K_D \cdot \frac{d\varepsilon}{dt} \tag{11}$$

$$Y = Y_P + Y_I + Y_D = K_P \cdot \epsilon + K_I \cdot \int \epsilon \ dt + K_D \cdot \frac{d\epsilon}{dt} \tag{12}$$

Referring to Eq. (12), the control output of the PID controller is obtained by summing the proportional, integral, and derivative control actions. It is then applied as an input to the actuator or system being controlled, which adjusts the process variable accordingly. The PID controller continuously adjusts its output based on the error signal feedback, striving to maintain the process variable close to the desired setpoint value.

3.10.1 Straight driving controller

After a multitude of experiments, we discovered that the voltage level of left wheel is always smaller than that of right wheel. To make up this flaw, the left wheel is assigned with a positive PWM value to balance the difference. However, the assigned value cannot be applied in any cases. For example, when the ground is wet, the assigned value should be larger; while when the ground is dry, the value should be smaller. Therefore, we should assign a varying PWM value to the left wheel, which can be achieved by a PID controller.

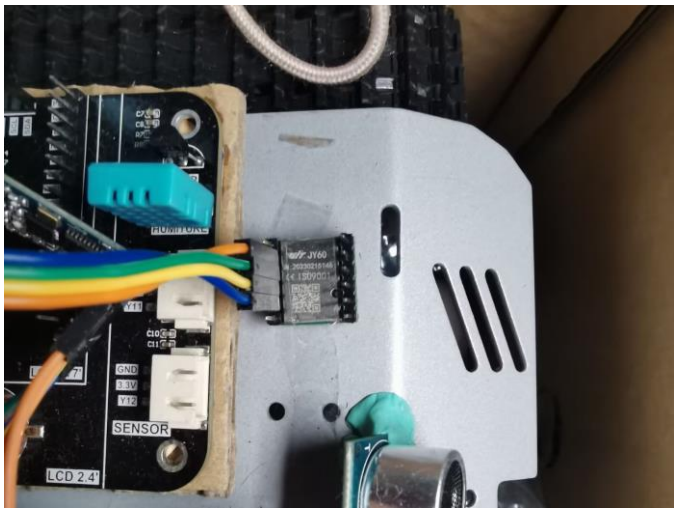


Figure 3.10.1: The gyroscope placed on the auto-vehicle

In order to control the vehicle to move straightly, we first monitored the moving direction of the car with a gyroscope as shown in **Figure 3.10.1**. The gyroscope is capable of recording the turning angle compared with the initial state, and it returns a value from 0° to 360°. In this case, we set the expected value of PID controller as the initial value of the gyroscope, which can impel the vehicle to move straightly. The actual value is the measured angle of the gyroscope.

By comparing the measured angle and expected angle, we can roughly decide the direction that the car should be compensating. If the car is moving left, we should assign a large PWM value to the left wheel. If the car is moving right, we would assign a small value to the left wheel.

Once the logic of this controller is confirmed, we need a recurrent algorithm to continuously adjust the direction of the vehicle. Every time the gyroscope refreshes, a 20-epoch PID algorithm would be conducted as shown in **Algorithm 2**. This algorithm allows the microcontroller to refresh the speeds of two wheels for 20 times in one "while True" loop. The PID controller for straight driving is shown in **Algorithm 2.**

Algorithm 2: PID controller for straight driving

Input: current direction a_c, expected direction a, speed v
Output: speed of left wheel v_l, speed of right wheel v_r
1. Set T=20, t=0, err=0, sum=0, d=0, e_n=0, e_b=0
2. Set K_p=0.06, K_i=0.001, K_d=0.07, d_f= 0, $drop$= 0.1

while $t<T$ **do**
    $err = a_c - a$
    $e_b = e_n$, $e_n = err$
    $sum \mathrel{+}= err$
    $d = e_n - e_b$
    $d_f = K_p\, err + K_i\, sum + K_d\, d$
    $v_l = v + d_f$, $v_r = v - d_f$
    Motor controlling
    $a_c \mathrel{+}= d_f - drop$
end while

3.10.2 Auto-Tracking controller

As we mentioned, the auto-tracking algorithm in patio 1 depends on the comparison of color blobs and the entire image. By analyzing the central points of the detected blobs and the central points of the image, we can determine the direction of movement for the self-driving vehicle. As shown in **Figure 3.10.2**, if the blob is positioned on the left side of the image, the vehicle should steer to the left, and if it is located on the right side of the image, the vehicle should steer to the right.

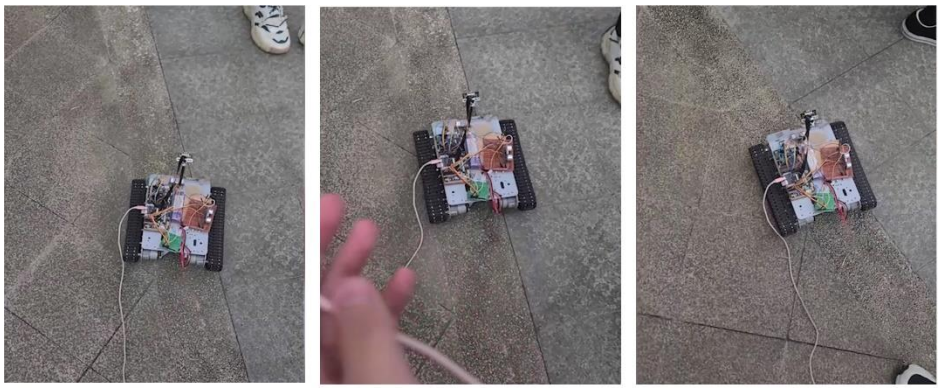


Figure 3.10.2: The actual running procedure of auto-vehicle

To completely move along the dark trajectory in patio 1, the car is required to adjust the speed of two wheels continuously. Every time the color blobs refreshes (or to say every time the camera takes a picture from 3000 snapshots), the microcontroller refreshes the speed of two wheels for 20 times.

Algorithm 3: Auto-Tracking controller based on PID

Input: color blobs center p_b, image center p_i, speed v
Output: left wheel speed v_l, right wheel speed v_r
1. Set $T=20, t=0, err=0, sum=0, d=0, e_n=0, e_b=0$
2. Set $K_p=0.06, K_i=0.001, K_d=0.07, d_f=0, drop=0.1$

while $t<T$ **do**

$err = p_b - p_i$
$e_b = e_n,\ e_n = err$
$sum \mathrel{+}= err$
$d = e_n - e_b$
$d_f = K_p\, err + K_i\, sum + K_d\, d$
$v_l = v + d_f,\ v_r = v - d_f$
Motor controlling
$p_b \mathrel{+}= d_f - drop$
$t = t + 1$

end while

As shown in **Algorithm 3**, we simultaneously execute the P controller, I controller and D controller to find the ideal difference between two wheels. Here, the P term provides a fast response to reduce the error, but it may lead to overshooting and oscillations. The I term is effective in addressing offset errors, but it can introduce instability if not properly tuned. The D term can help dampen the system's response, reducing overshooting and improving stability.

3.10.3 PID configuration

During the testing of our vehicle, we found it crucial to well-tune the parameters of a PID controller, so that we can achieve optimal performance and stability in a control system. The tuning of PID parameters should be not only consistent with the logic of the task in patios, but also suitable for the vehicle to operation stably in a long-term.

To find a feasible PID configuration, we manually tuned the parameters K_p, K_i, K_d. In manual tuning, the PID parameters are adjusted iteratively based on the operator's expertise and observations. The process involves making small changes to the proportional, integral, and derivative gains (K_p, K_i, K_d) and observing the system's response. We adjusted the gains until the desired control performance is achieved, considering factors like stability, overshoot, settling time, and steady-state error.

First of all, since the car should turn right when the distance of two central points is positive or the measured angle is larger than initial angle, the proportional parameter should also be positive. Since the summation of the distance and angles would be extreme large after a long-term operation, the integral parameter should be as small as possible, so that the PID would not be overshoot. The derivative parameter could be relatively large, since we want to avoid large oscillations in the real operation.

Most importantly, since the range of input for two task is similar (image distance: 0, 320], range of angles: [0, 360]), the PID configuration can actually shared by **Algorithm 2** and **Algorithm 3**. The PID configuration would be shown in **Table 3.10.1**.

K_p	0.006
K_i	0.001
K_d	0.007

Table 3.10.1: The PID configuration for both Algorithm 2 and algorithm 3

3.10.4 Result and discussion

In this section, we will discuss the choice of PID parameters and evaluate the performance of these speed controllers. The tuning of the PID controller involves selecting appropriate values for the proportional, integral, and derivative gains (K_p, K_i, K_d) to achieve optimal system performance. The tuning process requires an understanding of the system dynamics, response requirements, and trade-offs between stability, responsiveness, and steady-state accuracy. Various methods, such as trial and error, manual tuning, or advanced tuning algorithms, can be employed to determine the optimal PID gains for a specific control application.

When discussing the choice of PID parameters, it is important to consider the specific control system, its dynamics, and the desired performance objectives. The change of K_p can directly affect the final result of PID. Higher values of K_p result in stronger and more responsive control action, but they can also lead to overshoot and instability. Lower values of K_p may result in sluggish response. The integral term K_i helps to eliminate steady-state errors and improve the system's ability to reach and maintain the setpoint. However, a high value of K_i can introduce overshoot or cause instability, while a low value may result in slow error correction. The derivative term provides damping and helps to reduce overshoot and settle the system response. However, an excessively high derivative gain can amplify noise and make the control system more sensitive to measurement variations.

At the early testing of our PID controller, a large K_i was improperly assigned to **Algorithm 2**, which resulted a heavy overshoot of the vehicle. However, this mistake was corrected in later experiments. The PID configurations shown in **Table 3.10.1** can now successfully generate the correct speed of wheels, as shown in **Figure 3.10.3**.

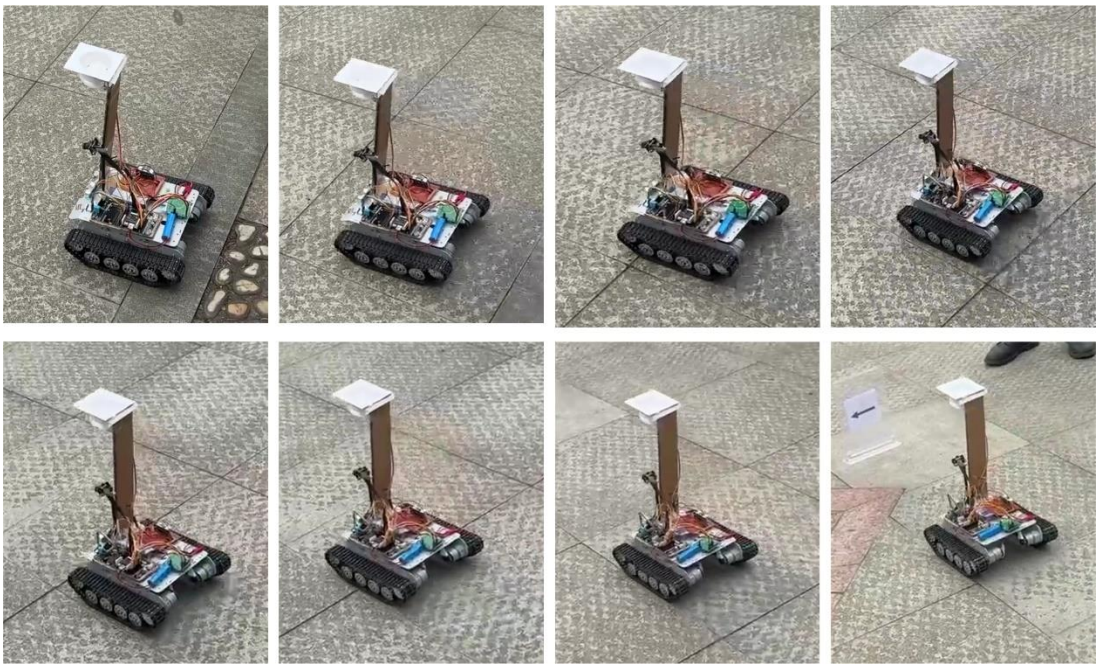


Figure 3.10.3: The actual running performance of the auto-vehicle

4

System Integration and Results

4.1 Overall System Integration (by LiuWeixing)

The car as a whole needs the cooperation of all parts to complete the work, but in fact, Patio 1 and Patio 2 have different requirements for the car. In these two parts, we carried out mild hardware modification and the overall software logic replacement of the car, so as to achieve better results.

4.1.1 Patio 1

Compared with Patio 2, the overall logic of Patio 1 is relatively simple, and its difficulty is mainly in the parameter adjustment and calculation of tracking (these contents have been mentioned above).

The patio 1 we need make the car passing through the track with have both curve track and straight track in it. For the part of following the track. The visual scheme was used in to control the car and form a loop.

For the bridge part, in order to knows when should the car turns, a beacon will be used to let the car knows when it arrives the up-bridge point. Then it will turn 90 degrees right, go up the bridge, go down the bridge. Through the fix distance. Then turn 90 degrees left.

After we go down the bridge, the gate is waiting for passing through. We passing the gate for a fix distance and finally stop to complete Patio 1.

The overall logic of Patio 1 can be summarized in the following figure:

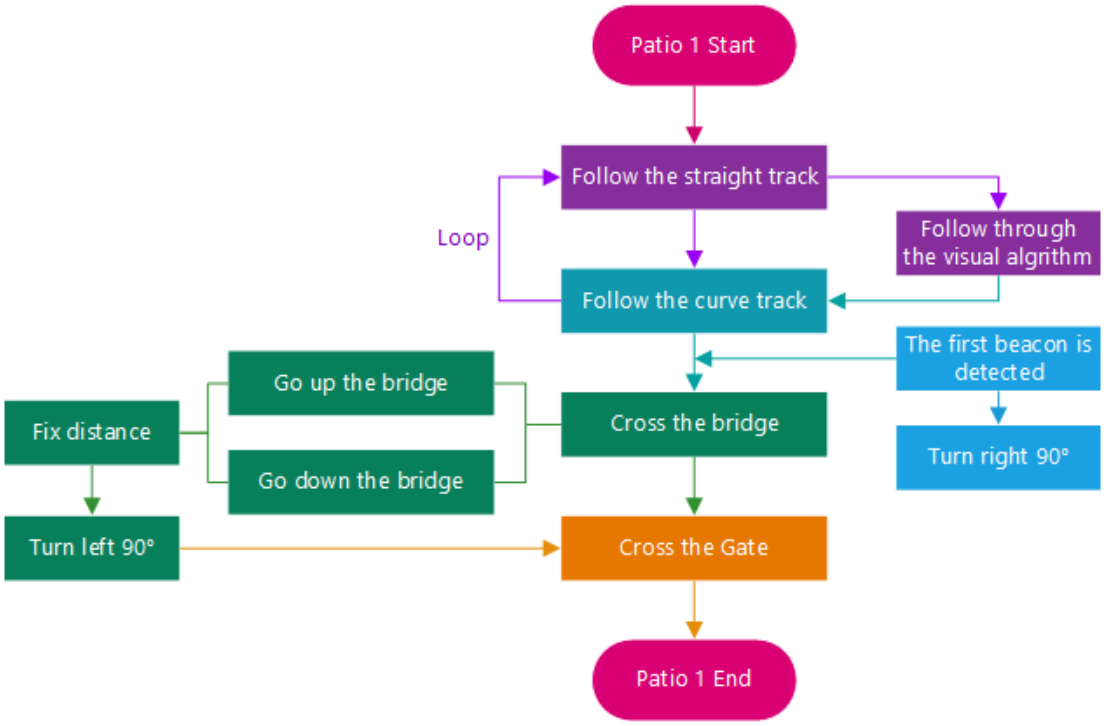


Figure 4.1.1: Whole logic of Patio 1

According to the task assignment of patio 1, at the beginning we are on the trail that needs to be followed, so we directly run the algorithm of advance and follow the trajectory, this algorithm with PID can automatically follow the straightaways and curves, and constantly execute this logic, after following several straightaways and curves, we can detect the first beacon.

After the radar in front detects the first beacon, it will issue an interrupt, requiring the vehicle to turn 90 degrees to the right, and then move forward to complete the task of crossing the bridge. The speed of the bridge after the bridge needs to be fine-tuned, otherwise it is easy to have problems, resulting in the rollover of the vehicle. After a fixed distance, you encounter the second beacon, turn left 90°, continue through the gate, the radar on the right detects a slightly shorter distance, you can stop the car and complete patio 1

4.1.2 Patio 2

In Task 2, we first move to the specified location to identify the direction of the arrow. Then, depending on the result, we need to go to different locations to knock down the corresponding sign.

We then look for the wall to the right and follow it until we reach the bucket to throw the ball. When the ball is finished, we will reverse back to the designated area and send the relevant information. The logic is like below:

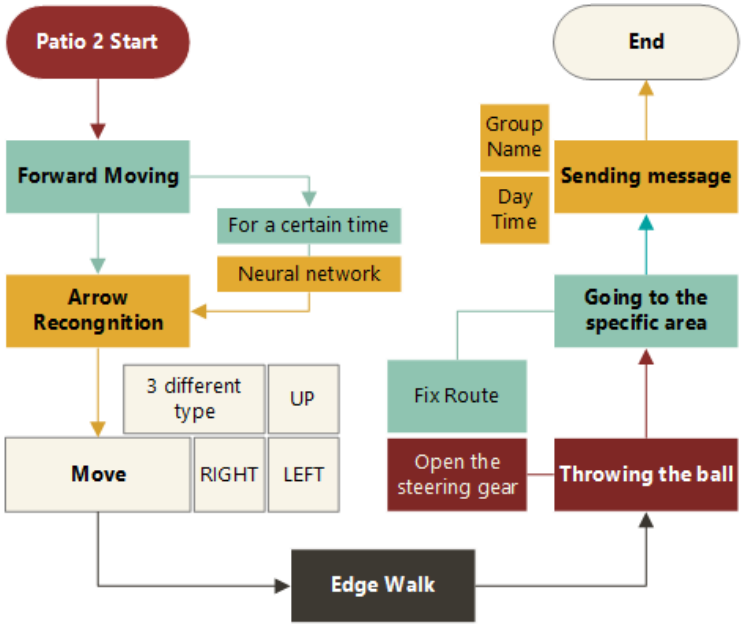


Figure 4.1.2: Whole logic of Patio 2

At the beginning of Patio 2, we needed to move forward and identify the direction of the arrows. Fix a distance and move forward to the sign.

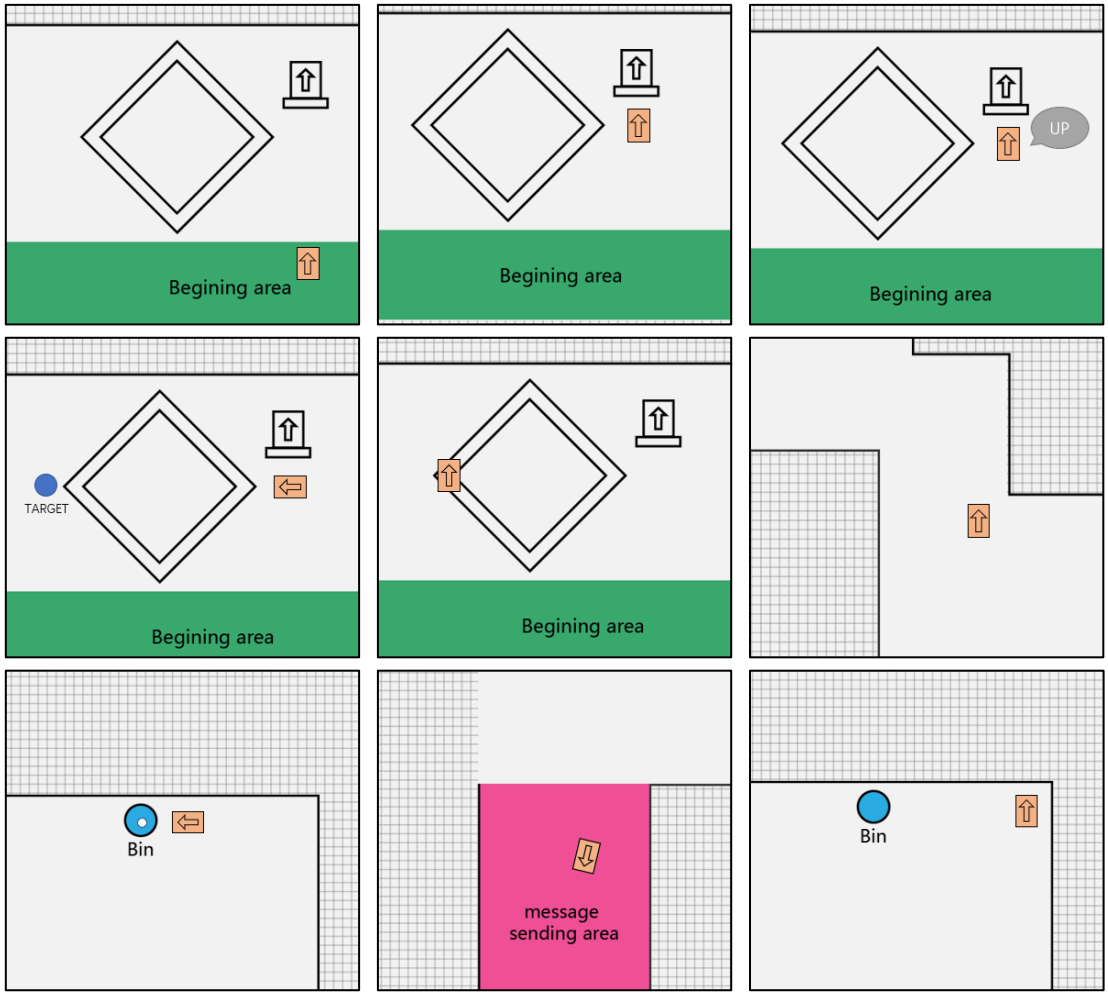


Figure 4.1.3: The step of Patio 2

Then, the arrow is recognized, after the recognition result, walk in accordance with the fixed route, after knocking down the sign, turn 0°, and move forward until there is a block in front of you, turn left, continue to walk forward, when there is no object detected on the right, turn right, and so on, achieve the effect of walking along the side, and finally walk to the bucket, throw the ball.

After throwing the ball, turn 170° to the message launching area and launch the fixed message.

4.2 Analyze and Discussion (by Bai Tianyou)

The working environment of the auto-vehicle, as shown in Figure 4.2.1, contains diversified noises for the operation of camera and motor. There are various enteral factors to influence the vehicle. For example, if the camera is exposed to an extreme strong light, the color recognition would be disabled. Different friction coefficient will also interfere the tuning of PID in Patio 2.

Figure 4.2.1: The possible external influence of the ground, which includes the friction coefficient of the ground, color of the trajectory and the slop of the bridge.

In order to identify the different influence of environment and evaluate the performance of the vehicle, a series of experiments were conducted under a various of conditions. Both patio 1 and patio 2 were tested in field, and experiments against other tasks such as arrow recognition, color recognition were tested on software platform.

4.2.1 Patio 1

a) LAB threshold optimization

During the field testing of K210 camera, we found that the weather has a significant impact on the RGB image binarization. As shown in **Figure 4.2.2**, if camera is exposed to an extreme strong sunlight, the captured image would be bleached, and the image processing algorithm will malfunction. If operated in a cloudy day, the K210 camera will not be receiving sufficient light, which may also result in bad performance of the image processing program.

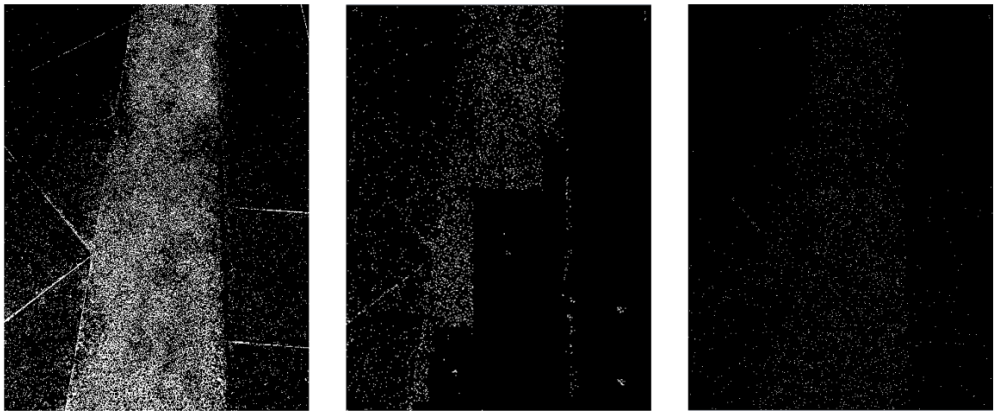


Figure 4.2.2: The image binarization result under different light intensity.

This problem can be solved by frequent tuning of LAB thresholds. The LAB threshold refers to the parameter tuning process associated with the LAB color space, which is a color model used in various image processing and computer vision applications. The LAB color space is designed to approximate human vision and is particularly useful for tasks involving color perception and analysis.

The LAB color space consists of three channels: L, a, and b. The L channel represents lightness, while the a and b channels represent color-opponent dimensions, with a representing the green-red axis and b representing the blue-yellow axis. The LAB threshold refers to the specific value or range used to define the boundary or limit for color differences within these channels.

To tune the LAB thresholds, we first took pictures of the dark trajectory under different light conditions, as shown in **Figure 4.2.3**.

Figure 4.2.3: The RGB images pf dark track under different light intensity.

Based on these images, a dataset for multiple LAB thresholds tuning were prepared for the image binarization. Based on the dataset analysis and task requirements, an initial threshold value or range is selected. This value acts as a starting point for the tuning process. The LAB threshold is then adjusted iteratively based on the results obtained from applying it to the dataset. The adjustments can be made manually by visual inspection or using quantitative evaluation metrics, depending on the specific task. The tuned threshold is evaluated against a validation set or ground truth data to assess its performance. After a multitude on comparison between LAB thresholds, the optimized threshold was confirmed as **[1, 89, -9, -1, 7, 28]**.

b) Tuning of PID parameters

When testing the PID controller for auto-track in patio 1, we found that the choice of PID

parameters was very important for the operation of motor. When an overlarge K_p or K_i was assigned to the controller, the vehicle may move acutely during long time running, or it will violently swing around the track as shown in **Figure. 4.2.4**. In this way, we conducted a Rigorous experiment to find the optimized PID configuration.

Figure 4.2.4: The overshoot of the PID controller when receiving an overlarge K_p.

The tuning of a PID controller involves adjusting its three parameters, proportional gain (K_p), integral gain (K_i), and derivative gain (K_d), to achieve the desired control performance. Before the tuning these parameters, their impact on the motor should be discussed. For K_p, we increased K_p to observe the system's response. A higher K_p amplifies the response to the error signal, but if it's too high, the system may become unstable or exhibit oscillations. Decreasing K_p if the system is oscillating or unstable. Gradually reduce K_p until the oscillations stop or become acceptable. We repeated the above steps until the system response is satisfactory in terms of stability and tracking the setpoint. For K_i, we increased K_i to reduce the steady-state error. A higher K_i amplifies the integral action, enabling the controller to eliminate offset between the setpoint and the process variable over time. It should be very cautious not to set K_i too high as it can lead to overshoot, instability, or slow response. For K_d, we introduced K_d to provide damping and improve the transient response of the system. Then, we increased K_d to dampen the response and reduce overshoot. This can help in systems with fast-changing setpoints or disturbances. If increasing K_d causes excessive oscillations or instability, reduce it. Care should be taken to avoid excessively high or low values of K_d, which can adversely affect system performance. Fine-tuning K_d is done after achieving satisfactory performance with K_p and K_i.

If the system exhibits large overshoot or instability, the K_i should be reduced. We iterate

this process until a suitable balance is achieved between steady-state accuracy and stability.

PID tuning is an iterative and often subjective process, requiring a balance between stability, responsiveness, and accuracy. The final available PID parameters for the auto-track task were confirmed as K_p=0.006, K_i=0.001, K_d=0.007.

After these tuning, the vehicle is proved to be capable of fulfilled patio 1 tasks, as shown in **Figure 4.2.5**.

Figure 4.2.5: The auto-tracking process of patio 1

4.2.2 Patio 2

a) LAB threshold for arrow recognition

During the field testing of patio1, we found that although neural network was adopted, the light condition can still have a strong influence on the classification result. When the light is low, we found that the resolution of left and right arrow was confused.

In order to avoid the influence of sunlight, we took pictures of three types of arrows under different sunlight condition as shown in **Figure 4.2.6**.

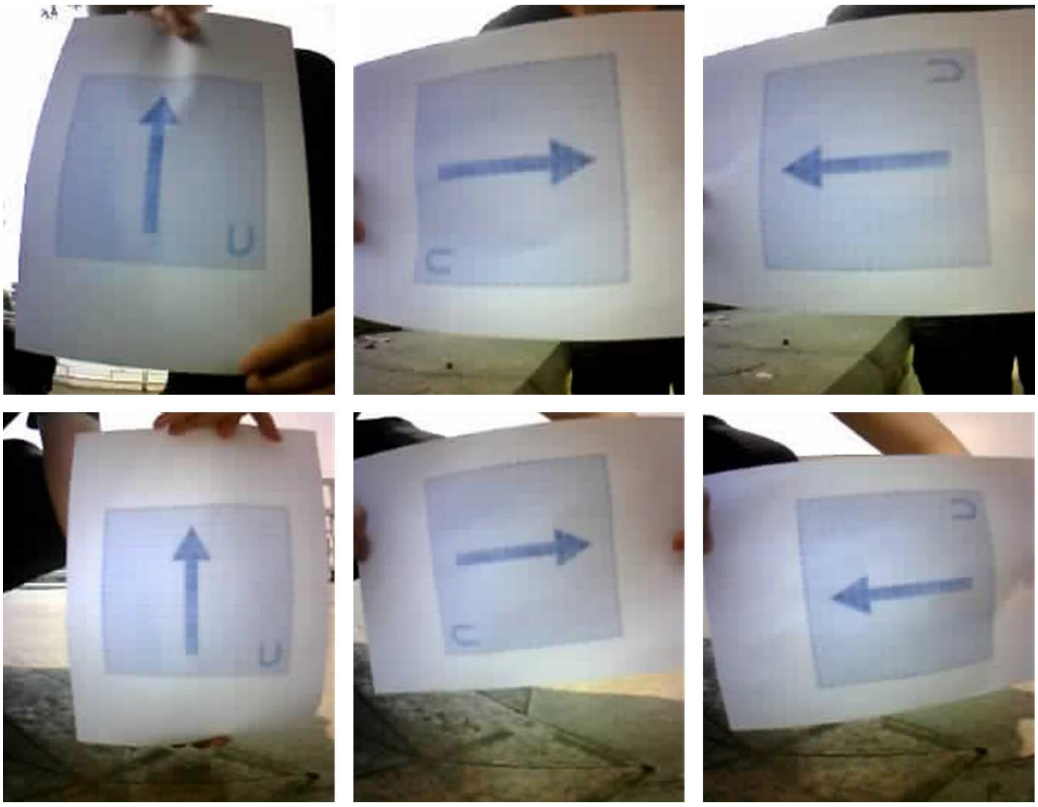


Figure 4.2.6: The picture of arrows under different light condition.

We shuffled the same class of arrow images under different light condition, and created a training dataset. In this way, the model will match the deep correlation between arrow classes and pixel values under different light condition.

The model was trained on an online deep learning platform, and the accuracy of classification is shown in **Figure 4.2.7.**

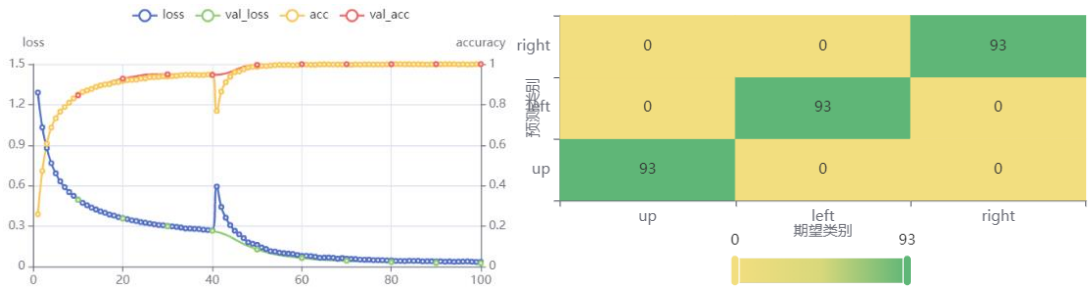


Figure 4.2.7: The training performance of MobileNet.

The K210 microcontroller is also capable of identifying arrow direction in the field testing, as shown in **Figure 4.2.8.**

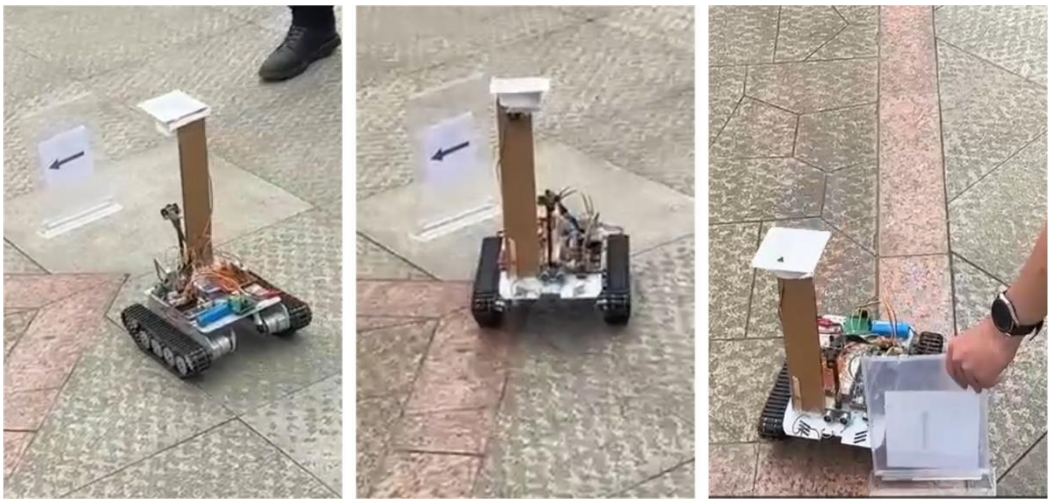


Figure 4.2.8: The arrow recognition result in field testing.

Radar module performance

As we mentioned, in patio 2, the auto-vehicle utilized a series of radar detection program to find the black bucket. There are two kinds of courses in the radar detecting. In one until the right radar do not detects any barrier. Another course is that the car turns left if both the front and right radar detects a barrier, and the vehicle stops when the front radar detects the black bucket.

In first course, the detecting threshold is decided as 30 cm, and the vehicle successfully moved along the baluster, as shown in **Figure 4.2.9**.

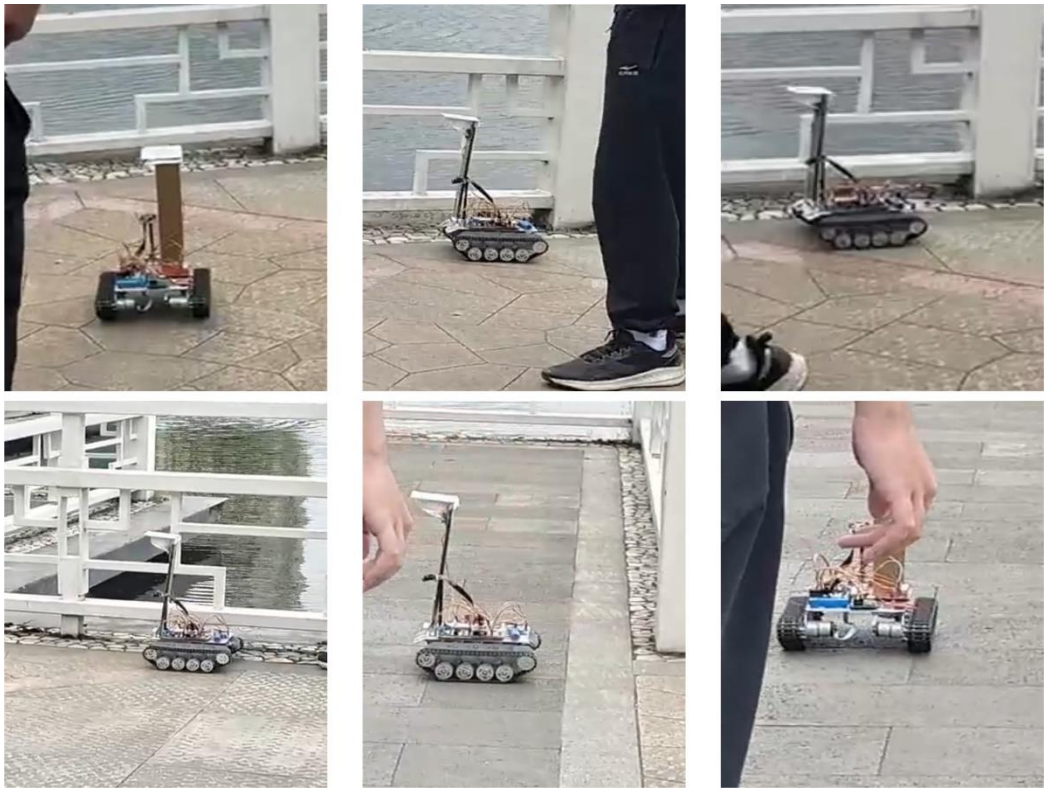


Figure 4.2.9: The actual running procedure of first course.

In second course, the detecting threshold is decided as 20 cm, and the vehicle successfully found the bucket, as shown in **Figure 4.2.10**.

Figure 4.2.10: The actual running procedure of second course.

After that, the vehicle is programed to turn left and head for the stop region, as shown in **Figure 4.2.11**. In the stop region, the car successfully stopped and transferred a message to the computer.

Figure 4.2.11: The final running performance of patio 2.

4.2.3 Future Work

For the work that can improve the project in the future, we roughly divide it into two main categories: teamwork problems and technical problems.

The first thing to say is the problem of team cooperation. In this project design, the coordination of the team has been relatively balanced. There are people who manage the whole project, and there are also people who coordinate each part.

Then there is the technical problem. In fact, the hardware structure and software algorithm of the car can be further improved. In the actual test, the robustness performance of the car can only be said to be average, for example, it cannot adapt to different ground, and it will slightly deviate to one side.

We might add more sensors to the car and give it a better camera to get a more complete picture, and if money is not limited, adding a positioning system to the car (like GPS/ Beidou Global Navigation system) would be a very challenging but cool way to get very good results.

With further research and simultaneous improvements in all areas, the final performance will be even better than it is today.

5

Project Management

5.1 Planning (by Wang Ziyu)

5.1.1 First stage

- Define the requirements for the vehicle to ensure we are focusing on the right features and specifications.

- Research and select the necessary hardware components such as PCB board, vehicle chassis, ultrasonic radar, and servo motors to ensure the robustness and reliability of the vehicle.

- Design the hardware layout and wiring schematics, which will give us a clear idea of the connection among different hardware components and the structure of the vehicle.

- Start software development for the line-following algorithm, which is the core functionality of the vehicle.

- Finish the hardware design and start assembling the vehicle. Assembling the prototype will allow us to test the components and address any issues early on.

- Begin software development for the vehicle's motion control and communication modules. This functionality will enable us to control the vehicle's motion, and communicate with it in a reliable and fault-tolerant way.

- Test the line-following algorithm on simulation environments to ensure that it works as expected.

- Conduct a risk analysis to help us identify potential issues and make necessary trade-offs.

5.1.2 Second stage

- Complete the vehicle assembly and integrate all hardware components. Assembling the vehicle will allow us to test the hardware together and address any issues early on.

- Begin testing the hardware and software integration. This is a critical step to ensure that all components work together seamlessly.

- Optimize the line-following algorithm and test it on real-world scenarios to make sure that it works as expected under different conditions.

- Develop and test communication protocols to ensure that there are no bugs and the communication is stable.

- Complete the software development. This is essential for testing the vehicle's robustness and stability.

- Perform functional testing and debugging. This step will help us identify and address

any problems that may arise in the software or hardware.

- Optimize the performance of the vehicle to ensure it meets our project requirements.
- Conduct a code review and quality assurance check to ensure that the code meets the project requirements.

5.1.3 Third stage

- Prepare for the final showcase. This is the last stage of the project and is critical in terms of communication and presentation.
- Refine the vehicle's design and aesthetics. This is important to make the vehicle look appealing and professional.
- Prepare documentation and presentation materials to share our progress and results with stakeholders effectively.
- Conduct final testing and debugging to ensure that the vehicle is reliable and robust.
- Practice for the showcase presentations to ensure that we can present our work confidently.

5.1.4 Final stage

- Showcase the vehicle and present the project to the teachers. This is the final stage of the project and will allow us to demonstrate our work.
- Receive feedback and answer questions from teachers.
- Evaluate the success of the project and identify areas for future improvements. This will help us to learn from our experience and make better decisions in the future.

5.2 Project Collaboration (by Zhang Tianyi)

In order to successfully implement this project, reasonable division of labor and cooperation among team members are essential. Firstly, in order to ensure the stable progress of the project, a project manager is necessary. He should be responsible for coordinating financial management and project scheduling, keeping track of the progress of each member, and communicating and supervising them in a timely manner. Then, the project adopts a divide-and-conquer strategy, which means that divide the entire team into two groups: software design (5 people) and hardware design (4 people)[19].

The software and hardware teams each have a group leader responsible for organizing documents, tracking group members' progress, and coordinating with the project manager. In the software group, the work is divided into four parts[20]:

(1) power and control;
(2) Machine vision line inspection;
(3) Ultrasonic sensors and gyroscopes;
(4) Steering gear and communication module.

The work of the hardware team is also divided into four parts:

(1) assembly, layout design of the car;
(2) Power supply system and PCB design;
(3) The selection of various module components and communication and docking with the corresponding member of the software group who is in charge of this module.

Each part of the work is undertaken by one person. Table 1 below shows the main contributions of group members.

Name	Main Contribution
Wang Ziyu	Project Manager
Zhang Tianyi	Hardware integration
Wang Chen	Software integration
Liu Weixing	Power and control
Bai Tianyou	Machine vision line inspection
Wang Zhiyi	Ultrasonic sensors and gyroscopes
Fan Xilai	Steering gear and communication module
Liu Zhuqing	Module selection and docking
Zhao Zidong	Power supply system and pcb design
Yang Tian	Steering gear support

Table 5.2.1: Team members' contribution

In addition, the project manager and the leader of the software and hardware team are mainly responsible for the non-technical part of the work, which has relatively less workload. Therefore, they are also responsible for reviewing and approving the component purchase applications of team members, as well as organizing notebooks

and documents. The writing of each section of the final report is the responsible by each member whose work is related to this part.

5.3 Gantt chart (by Yang Tian)

Gantt Chart is a project management tool used to visualize the planning, progress, and assignment of a project. It displays the timeline of the project and related tasks in the form of a bar chart, so that team members can clearly understand the activities and timelines in the project.

Through Gantt charts, project teams can better understand the relationship between the project's schedule and tasks. It can help team members know which tasks need to start and end when, and the priorities between tasks. In addition, the team can monitor and control the project through the Gantt chart, identify delays or schedule deviations, and adjust the plan to keep the project on schedule.

Here is the Gantt chart.

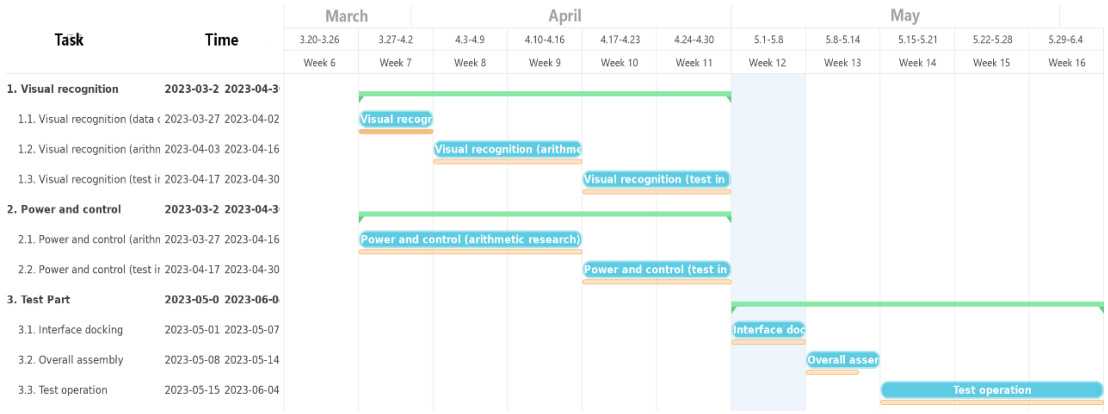


Figure 5.3.1: Gantt graph

5.4 Budget (by Wang Chen)

In project management, the importance of cost control cannot be ignored. Here are a few points about the importance of controlling costs:

1. Budget rationality: Controlling costs helps to ensure that the project budget is reasonable and aligned with the organization's financial objectives. Through effective cost control, budget overruns, financial risks and financial distress can be avoided.

2. Resource optimization: Cost control can help optimize resource utilization. Through reasonable allocation and management of resources, resources waste and low efficiency can be reduced, resource utilization can be improved, and the total cost of the project can be reduced.

3. Project schedule control: Cost control is closely related to project schedule. Effective cost control can help project managers better control the project schedule, ensure that the project is completed on time, and avoid project delays or interruptions due to cost overruns.

To sum up, cost control is very important in project management. It not only helps to ensure budget rationality and profitability of the project, but also optimizes resource utilization, reduces risk, controls project schedules, and provides decision support. Through effective cost control, organizations can improve the success rate of projects and achieve better economic benefits.

The following is the budget table of my group:

Name	Number	Cost (total)
k210	1	¥460.50
Crawler Chassis	1	¥299.00
HC-SR04	3	¥14.40
Steering Gear	2	¥12.30
Bluetooth (BT-11)	1	¥8.85
Battery (55V)	1	¥49.00
Battery and Fuse (3.7V)	1	¥15
T-connector with wire	2	¥11.80
Camera Bracket	1	¥55.50
HC-12	1	¥20.00
Clock Module	1	¥3.81
Gyroscope (JY60)	1	¥46.00
Total		**¥996.16**

Table 5.4.1: Budget

Conclusion

To sum up, the successful design, implementation, and integration of the automatic line following intelligent car with arrow recognition and ball throwing functions was thanks to the effective cooperation of the ten-member team in the project. The collective effort and collaboration of the team is essential to achieving the project goals. Each team member contributes their expertise and works together towards a common goal.

The entire system design, task breakdown, and analysis of hardware and software architecture is the result of the team's collective brainstorming and decision-making process. The design of the subsystems, including the vehicle structure, the main control chip, the power supply, the motor drive module, the steering gear installation, the visual identification system, the gyroscope, the ultrasonic radar, the communications and the clock, was accomplished through close collaboration and mutual understanding among the team members.

In the system integration phase, the team seamlessly integrates the various subsystems and ensures their compatibility. The team's collective knowledge and problem-solving skills help solve any challenges or problems that arise during the integration process.

The performance of the robot in line pursuit, arrow recognition and ball throwing tasks is analyzed and discussed, and the collaborative effort of the team is highlighted. Through effective communication and coordination, the team fine-tuned the PID controller, optimized the visual recognition algorithm, and calibrated the ball throwing mechanism to achieve the desired effect.

The project management section of the report emphasizes the importance of teamwork. As reflected in the Gantt chart, team planning and project coordination ensure proper assignment of tasks and effective monitoring of progress. The team's commitment to effectively managing the budget further demonstrates their dedication and professionalism.

Overall, the successful completion of this project demonstrates the importance of teamwork in achieving complex goals. The combined efforts of this team of ten, combined with their expertise and effective project management strategies, contributed to the design and implementation of a self-tracking smart car capable of recognizing arrows and throwing balls. The project demonstrates the potential of collaborative teams in the field of robotics and automation.

Reference

[1] 'Getting Started with K210 - Sipeed Wiki'. https://wiki.sipeed.com/news/MaixPy/K210_usage.html (accessed Jun. 10, 2023).

[2] N. H. Tollervey, *Programming with MicroPython: embedded programming with microcontrollers and Python*. O'Reilly Media, Inc., 2017.

[3] 'APTECHDEALS 12 volt Dc Motor Price in India - Buy APTECHDEALS 12 volt Dc Motor online at Flipkart.com'. https://www.flipkart.com/aptechdeals-12-volt-dc-motor/p/itmey9g36jh39wuc (accessed Jun. 10, 2023).

[4] A. Pressman, *Switching power supply design*. McGraw-Hill Education, 2009.

[5] 'Electric Fuses - Arthur Wright, P. Gordon Newbery - Google 图书'. https://books.google.com/books?hl=zh-CN&lr=&id=8r-XrMvGm4YC&oi=fnd&pg=PR11&dq=The+importance+of+fuses&ots=3eAw6QROK1&sig=U6gWNyksBGhw8wy6KCyOvAQipZU#v=onepage&q=The%20importance%20of%20fuses&f=false (accessed Jun. 10, 2023).

[6] D. L. Jones, 'PCB design tutorial', *June 29th*, vol. 3, p. 25, 2004.

[7] D. M. Wolpert, 'Computational approaches to motor control', *Trends Cogn. Sci.*, vol. 1, no. 6, pp. 209–216, 1997.

[8] N. Cameron and N. Cameron, 'Servo and Stepper Motors', *Arduino Appl. Compr. Proj. Everyday Electron.*, pp. 157–176, 2019.

[9] D. Zhang, Z. Xu, L. Wang, C. Song, and Z. Xu, 'Digital logic experiment design based on FPGA development board and OV2640 camera', in *2019 14th International Conference on Computer Science & Education (ICCSE)*, IEEE, 2019, pp. 671–675.

[10] K. J. Åström and T. Hägglund, 'The future of PID control', *Control Eng. Pract.*, vol. 9, no. 11, pp. 1163–1175, 2001.

[11] D. Sinha and M. El-Sharkawy, 'Thin mobilenet: An enhanced mobilenet architecture', in *2019 IEEE 10th annual ubiquitous computing, electronics & mobile communication conference (UEMCON)*, IEEE, 2019, pp. 0280–0285.

[12] L. Zhao and L. Wang, 'A new lightweight network based on MobileNetV3.', *KSII Trans. Internet Inf. Syst.*, vol. 16, no. 1, 2022.

[13] J. B. Scarborough, *The Gyroscope*. Interscience Publ., 1958.

[14] 'JY60 product specification · Shenzhen Victor Intelligent Technology Co., LTD'. https://wit-motion.yuque.com/wumwnr/docs/geusnl? (accessed Jun. 10, 2023).

[15] K. N. Lee, 'Compass and gyroscope: integrating science and politics for the environment', Island Press, Washington, DC (United States), 1993.

[16] K. Liu *et al.*, 'The development of micro-gyroscope technology', *J. Micromechanics Microengineering*, vol. 19, no. 11, p. 113001, 2009.

[17] E. J. Morgan, 'HC-SR04 ultrasonic sensor', *Nov*, 2014.

[18] M. A. Johnson and M. H. Moradi, *PID control*. Springer, 2005.

[19] M. M. Beyerlein, 'Beyond teams: Building the collaborative organization', *No Title*, 2003.

[20] S. W. Kozlowski and D. R. Ilgen, 'Enhancing the effectiveness of work groups and teams', *Psychol. Sci. Public Interest*, vol. 7, no. 3, pp. 77–124, 2006.

Appendix

arrow_rec.py

```
import sensor, image, time ,lcd
from Maix import GPIO
from fpioa_manager import fm

class ArrowRec:
    def __init__(self,img_ori,need_threshold=(10, 83)):
        # (19, 65, -1, 8, -12, 0)
        self.need_threshold = need_threshold
        self.img_ori = img_ori
        # self.img_grey = self.grayscale()
        self.blobs = self.get_blob()
        self.img_cut = self.cutting()

    def get_blob(self):
        blobs =
self.img_ori.find_blobs([self.need_threshold],x_stride=3,y_stride=3,pi
xels_threshold=500)
        compare = 0
        count = -1
        for i in range(len(blobs)):
            if blobs[i][4]>compare:
                count = i
                compare = blobs[i][4]
        return blobs[count]

    def draw_blob(self):
        tmp = self.img_ori.copy()
        if self.blobs:
            print(self.blobs)
            tmp.draw_rectangle(self.blobs[0:4])
            tmp.draw_cross(self.blobs[5], self.blobs[6])
            return tmp
        else:
            print("no blobs find!")
            return False

    def cutting(self):
        if self.blobs:
            return self.img_ori.copy(roi = self.blobs[0:4])
```

```
else:
            print("no blobs find!")
            return False

    def rec_arrow(self):
        if self.blobs:
            h = self.blobs[3]
            w = self.blobs[2]
            if h-w>40:
                return "U"
            else:
                half_w = int(w/2)
                left_half = self.img_cut.copy(roi = (0, 0, half_w, h))
                right_half = self.img_cut.copy(roi = (half_w, 0,
half_w, h))
                l_value =
left_half.find_blobs([self.need_threshold])[0][4]
                r_value =
right_half.find_blobs([self.need_threshold])[0][4]
                if l_value>r_value:
                    return "L"
                else:
                    return "R"
        else:
            print("no blobs find!")
            return False
```

gyroscope.py

```
from machine import UART,Timer
from fpioa_manager import fm

class gyroscope:
    def __init__(self, baud=9600, RX_pin=6, TX_pin=7):
        self.uart = UART(UART.UART1, 9600, read_buf_len=4096)
        fm.register(RX_pin, fm.fpioa.UART1_RX, force=True)
        fm.register(TX_pin, fm.fpioa.UART1_TX, force=True)
        self.state = True
        self.op_code('\x61')
        self.op_code('\x65')
        # self.d_angle = 0

    def __repr__(self):
        return self.get_angle()

    def op_code(self, code):
        self.uart.write(b'\xff')
        self.uart.write(b'\xaa')
        self.uart.write(code)

    def set_zero(self):
        self.op_code(b'\x52')

    def switch(self):
        self.state = not self.state
        self.op_code(b'\x60')

    def data_read(self):
        data = self.uart.read()
        return data

    def get_angle(self):
        if not self.state:
            print("The gs is off!")
            return None
        while True:
            raw_data = self.data_read()
            if raw_data:
                if raw_data[0] == 85 and raw_data[1] == 81 and
len(raw_data)==33:
                    break
```

```
                else:
                    continue
            else:
                continue
        return (((raw_data[29]<<8)|raw_data[28])/32768)*180

    def angle_record_start(self):
        self.set_zero()
        self.d_angle = 0
        self.p_angle = self.get_angle()

    def angle_change(self):
        now_angle = self.get_angle()
        self.d_angle = self.d_angle + (now_angle - self.p_angle)
        self.p_angle = now_angle
        if self.d_angle>=180:
            return self.d_angle - 360
        else:
            return self.d_angle
```

HC12.py

```
from machine import UART,Timer
from fpioa_manager import fm
from Maix import GPIO
from time import sleep_ms

class HC_12:
    def __init__(self, baud=9600, RX_pin=31, TX_pin=16, SET_pin=28):
        self.uart = UART(UART.UART1, 9600, read_buf_len=4096)
        fm.register(RX_pin, fm.fpioa.UART1_RX, force=True)
        fm.register(TX_pin, fm.fpioa.UART1_TX, force=True)
        fm.register(SET_pin, fm.fpioa.GPIOHS10)
        self.set_pin = GPIO(GPIO.GPIOHS10, GPIO.OUT)
        self.set_pin.value(1)

    def send(self, text="aaaa"):
        message = text.encode('utf-8')
        print(message)
        self.uart.write(message)

    def read(self):
        while True:
            message = self.uart.read()
            if message:
                break
        return message.decode('utf-8')

    def command(self, cmd):
        self.set_pin.value(0)
        sleep_ms(40)
        self.uart.write('AT+'+cmd)
        print(self.read())
        self.set_pin.value(1)
        sleep_ms(80)

    def set_channel(self, channel):
        self.command(str(channel))
```

motor_controller.py

```
from machine import Timer,PWM
import time
import utime
from Maix import GPIO
from fpioa_manager import fm
import _thread


class motor:
    def __init__(self, pwm_pin, in_1, in_2):
        self.PWM = pwm_pin

        self.in_1 = in_1
        self.in_2 = in_2

    def controller(self, speed):
        if speed > 0:
            self.in_1.value(1)
            self.in_2.value(0)
            self.PWM.duty(abs(speed) * 100)

        elif speed == 0:
            self.in_1.value(0)
            self.in_2.value(0)
            self.PWM.duty(0)

        else:
            self.in_1.value(0)
            self.in_2.value(1)
            self.PWM.duty(abs(speed) * 100)

    def extracter(self):
        return(self.PWM.duty, self.in_1.value(), self.in_2.value)


class motor_controller_blocking:
    def __init__(self, pwm_pin_L, in_1_L, in_2_L, pwm_pin_R, in_1_R,
in_2_R):
        self.motor_L = motor(pwm_pin_L, in_1_L, in_2_L)
        self.motor_R = motor(pwm_pin_R, in_1_R, in_2_R)

        self.motor_L.controller(0)
        self.motor_R.controller(0)
```

```
    def fw(self, t, speed):
        self.motor_L.controller(speed / 2)
        self.motor_R.controller(speed / 2)

        utime.sleep_ms(t)

        self.motor_L.controller(0)
        self.motor_R.controller(0)


    def turning(self, t, direction, speed):
        if direction == 'L':
            self.motor_L.controller(-speed / 2)
            self.motor_R.controller(speed / 2)
        else:
            self.motor_L.controller(speed / 2)
            self.motor_R.controller(-speed / 2)

        utime.sleep_ms(t)

        self.motor_L.controller(0)
        self.motor_R.controller(0)


    def circling(self, t, direction, speed_avg, speed_diff):
        if direction == 'L':
            self.motor_L.controller(speed_avg / 2 - speed_diff / 2)
            self.motor_R.controller(speed_avg / 2 + speed_diff / 2)
        else:
            self.motor_L.controller(speed_avg / 2 + speed_diff / 2)
            self.motor_R.controller(speed_avg / 2 - speed_diff / 2)

        utime.sleep_ms(t)

        self.motor_L.controller(0)
        self.motor_R.controller(0)

    def stop(self):
        self.motor_L.controller(0)
        self.motor_R.controller(0)


class motor_controller_unblocking:
```

```
    def __init__(self, pwm_pin_L, in_1_L, in_2_L, pwm_pin_R, in_1_R,
in_2_R):
        self.motor_L = motor(pwm_pin_L, in_1_L, in_2_L)
        self.motor_R = motor(pwm_pin_R, in_1_R, in_2_R)

        self.l_speed = 0
        self.r_speed = 0
        self.motor_L.controller(self.l_speed)
        self.motor_R.controller(self.r_speed)


    def fw(self, speed):
        self.l_speed = speed / 2
        self.r_speed = speed / 2
        self.motor_L.controller(self.l_speed)
        self.motor_R.controller(self.r_speed)


    def turning(self, direction, speed):
        if direction == 'L':
            self.l_speed = -speed / 2
            self.r_speed = speed / 2
            self.motor_L.controller(self.l_speed)
            self.motor_R.controller(self.r_speed)
        else:
            self.l_speed = speed / 2
            self.r_speed = -speed / 2
            self.motor_L.controller(self.l_speed)
            self.motor_R.controller(self.r_speed)


    def circling(self, direction, speed_avg, speed_diff):
        if direction == 'L':
            self.l_speed = speed_avg / 2 - speed_diff / 2
            self.r_speed = speed_avg / 2 + speed_diff / 2
            self.motor_L.controller(self.l_speed)
            self.motor_R.controller(self.r_speed)
        else:
            self.l_speed = speed_avg / 2 + speed_diff / 2
            self.r_speed = speed_avg / 2 - speed_diff / 2
            self.motor_L.controller(self.l_speed)
            self.motor_R.controller(self.r_speed)


    def stop(self):
```

```
        self.l_speed = 0
        self.r_speed = 0
        self.motor_L.controller(self.l_speed)
        self.motor_R.controller(self.r_speed)

    def pause(self, btime):
        self.stop()
        time.sleep(btime)

    def motor_interupt(self, KEY):
        utime.sleep_ms(10)
        print("Right Motor: {}".format(self.r_speed))
        print("Left Motor: {}".format(self.l_speed))

        if KEY.value()==0:
            self.motor_L.controller(self.l_speed)
            self.motor_R.controller(self.r_speed)


def fun(KEY):
    utime.sleep_ms(10)
    if KEY.value()==0:
        mc.stop()


def l_speed_generator(motor):
    while True:
        motor.l_speed = -motor.l_speed
        print("hello {}".format(motor.l_speed))
        time.sleep(1)


def r_speed_generator(motor):
    while True:
        motor.r_speed = -motor.r_speed
        print("hello {}".format(motor.r_speed))
        time.sleep(1)
```

radar.py

```
from time import sleep_us, ticks_us
import time
from Maix import GPIO
from fpioa_manager import fm


class HCSR04():
    def __init__(self, trig, echo):
        self.trig = trig
        self.echo = echo

    def getDistance(self):
        distance = 0
        self.trig.value(1)
        sleep_us(20)
        self.trig.value(0)
        while self.echo.value() == 0:
            pass
        if self.echo.value() == 1:
            ts = ticks_us()
            while self.echo.value() == 1:
                pass
            te = ticks_us()
            tc = te - ts
            distance = tc * 0.017
        return distance
```

main.py (patio2)

```
from motor_controller import *
from radar import *

from time import sleep_us, sleep
from gyroscope import *

import sensor

from arrow_rec import ArrowRec

class patio2:
    def __init__(self, mc, gs, HC1, HC2, sg90):
        self.mc = mc
        self.gs = gs
        self.HC1 = HC1
        self.HC2 = HC2
        self.sg90 = sg90

        # constant
        self.btime = 0.5

        # step1
        self.step_1_time = 5

        # step2

        # step3
        self.step_3_time = [1, 1, 1, 1, 1, 1]
        # step4
        self.arr_size_1 = 10
        self.div_time_1 = 0.02
        self.dis_to_fence_1 = 50
        # step5
        self.arr_size_2 = 10
        self.div_time_2 = 0.02
        self.dis_to_fence_2 = 30
        # step6
        self.arr_size_1 = 10
        self.div_time_1 = 0.02
        self.dis_to_fence_1 = 50
        # step7
        self.dis_to_fence_3 = 30
```

```
        self.dis_to_bin = 10
        # step8
        self.angle = 5


    def step_test(self):
        self.mc.fw(1)
        arr_size = 10
        dis_list = [0] * arr_size
        while True:
            dis_list[count%arr_size] = self.HC1.getDistance()
            print(dis_list[count%arr_size])
            print("Step_Test-FW-L:{}, R:{}".format(self.mc.l_speed,
self.mc.r_speed))
            sleep(2)
            print(self.mc.l_speed, self.mc.r_speed)
            count = count + 1
            dis_list_copy = dis_list.copy()
            dis_list_copy.sort()

            if dis_list[arr_size//2]>10:
                self.mc.pause(self.btime)

                self.gs.angle_record_start()
                self.mc.circling("R", 0, 1)
                while self.gs.angle_change() > -90:
                    print("Step_Test-R90-L:{}, R:{},
Angle:{}".format(self.mc.l_speed, self.mc.r_speed,
self.gs.angle_change()))
                    sleep_us(20)
                # 陀螺仪 R90 度
                self.mc.pause(self.btime)
                break

        self.mc.fw(1)
        print(self.mc.l_speed, self.mc.r_speed)
        count = 0
        while True:
            dis_list[count%arr_size] = self.HC2.getDistance()
            print(dis_list[count%arr_size])
            sleep(2)
            count = count + 1
            dis_list_copy = dis_list.copy()
            dis_list_copy.sort()
```

```
            if dis_list[arr_size//2]<10:
                mc.pause(self.btime)
                print(self.mc.l_speed, self.mc.r_speed)
                self.gs.angle_record_start()
                self.mc.circling("L", 0, 1)
                while self.gs.angle_change() < 90:
                    print("Step_Test-L90-L:{}, R:{},
Angle:{}".format(self.mc.l_speed, self.mc.r_speed,
self.gs.angle_change()))
                    sleep_us(20)

                # 陀螺仪 L90 度
                self.mc.pause(self.btime)
                break
        self.mc.fw(1)
        print("Step_Test-FW-L:{}, R:{}".format(self.mc.l_speed,
self.mc.r_speed))


    def step_1(self):
        self.mc.fw(1)
        print("Step_1-FW-L:{}, R:{}".format(self.mc.l_speed,
self.mc.r_speed))
        sleep(self.step_1_time)
        self.mc.stop()
        print("Step_1-STOP-L:{}, R:{}".format(self.mc.l_speed,
self.mc.r_speed))


    def step_2(self):
        sensor.reset()
        sensor.set_pixformat(sensor.RGB565)
        sensor.set_framesize(sensor.QVGA)
        sensor.run(1)
        img = sensor.snapshot()
        a = ArrowRec(img,need_threshold=(8, 43, -8, 4, -8, 2))
        b = a.rec_arrow()
        print(b)
        sensor.run(0)
        sensor.shutdown(False)

        return b


    def step_3(self, direction):
```

```
        print("Step: 3")
        if direction == 'R':
            self.mc.circling("L", 0, 1)
            self.gs.angle_record_start()
            while self.gs.angle_change() < 45:
                print("Step_3R_L45-L:{}, R:{},
Angle:{}".format(self.mc.l_speed, self.mc.r_speed,
self.gs.angle_change()))
                sleep_us(20)
            # 陀螺仪 L45 度

            self.mc.pause(self.btime)

            self.mc.fw(1)
            print("Step_3R_FW-L:{}, R:{}".format(self.mc.l_speed,
self.mc.r_speed))
            sleep(self.step_3_time[0])
            self.mc.pause(self.btime)

            self.mc.circling("L", 0, 1)
            self.gs.angle_record_start()
            while self.gs.angle_change() < 45:
                print("Step_3R_L45-L:{}, R:{},
Angle:{}".format(self.mc.l_speed, self.mc.r_speed,
self.gs.angle_change()))
                sleep_us(20)
            # 陀螺仪 L45 度
            self.mc.pause(self.btime)

        elif direction == 'U':
            self.mc.circling("L", 0, 1)
            self.gs.angle_record_start()
            while self.gs.angle_change() < 90:
                print("Step_3R_L90-L:{}, R:{},
Angle:{}".format(self.mc.l_speed, self.mc.r_speed,
self.gs.angle_change()))
                sleep_us(20)
            # 陀螺仪 L90 度
            self.mc.pause(self.btime)

            self.mc.fw(1)
            print("Step_3U_FW-L:{}, R:{},
Angle:{}".format(self.mc.l_speed, self.mc.r_speed,
self.gs.angle_change()))
```

```
            sleep(self.step_3_time[1])
            self.mc.pause(self.btime)

            self.mc.circling("R", 0, 1)
            self.gs.angle_record_start()
            while self.gs.angle_change() > -90:
                print("Step_3U_R90-L:{}, R:{},
Angle:{}".format(self.mc.l_speed, self.mc.r_speed,
self.gs.angle_change()))
                sleep_us(20)
            # 陀螺仪 R90 度
            self.mc.pause(self.btime)

            self.mc.fw(1)
            print("Step_3U_FW-L:{}, R:{},
Angle:{}".format(self.mc.l_speed, self.mc.r_speed,
self.gs.angle_change()))
            sleep(self.step_3_time[2])
            self.mc.pause(self.btime)

            self.mc.circling("L", 0, 1)
            self.gs.angle_record_start()
            while self.gs.angle_change() < 90:
                print("Step_3U_L90-L:{}, R:{},
Angle:{}".format(self.mc.l_speed, self.mc.r_speed,
self.gs.angle_change()))
                sleep_us(20)
            # 陀螺仪 L90 度
            self.mc.pause(self.btime)

        elif direction == 'L':
            self.mc.circling("R", 0, 1)
            self.gs.angle_record_start()
            angle_now = self.gs.angle_change()
            while angle_now > -45:
                angle_now = self.gs.angle_change()
                print("Step_3L_R45-L:{}, R:{},
Angle:{}".format(self.mc.l_speed, self.mc.r_speed, angle_now))
                sleep_us(20)
            # 陀螺仪 R45 度
            self.mc.pause(self.btime)

            self.mc.fw(-1)
```

```
            print("Step_3L_FW-L:{}, R:{}".format(self.mc.l_speed,
self.mc.r_speed))
            sleep(self.step_3_time[3])
            self.mc.pause(self.btime)

            self.mc.circling("R", 0, 1)
            self.gs.angle_record_start()
            while self.gs.angle_change() > -45:
                print("Step_3L_R45-L:{}, R:{},
Angle:{}".format(self.mc.l_speed, self.mc.r_speed,
self.gs.angle_change()))
                sleep_us(20)
            # 陀螺仪 R45 度
            self.mc.pause(self.btime)

            self.mc.fw(-1)
            print("Step_3L_FW-L:{}, R:{}".format(self.mc.l_speed,
self.mc.r_speed))
            sleep(self.step_3_time[4])
            self.mc.pause(self.btime)

            self.mc.circling("L", 0, 1)
            self.gs.angle_record_start()
            while self.gs.angle_change() < 90:
                print("Step_3L_L90-L:{}, R:{},
Angle:{}".format(self.mc.l_speed, self.mc.r_speed,
self.gs.angle_change()))
                sleep_us(20)
            # 陀螺仪 L90 度
            self.mc.pause(self.btime)

            self.mc.fw(1)
            print("Step_3L_FW-L:{}, R:{}".format(self.mc.l_speed,
self.mc.r_speed))
            sleep(self.step_3_time[5])
            self.mc.pause(self.btime)

            self.mc.circling("L", 0, 1)
            self.gs.angle_record_start()
            while self.gs.angle_change() < 90:
                print("Step_3L_L90-L:{}, R:{},
Angle:{}".format(self.mc.l_speed, self.mc.r_speed,
self.gs.angle_change()))
                sleep_us(20)
```

```
            # 陀螺仪 L90 度
            self.mc.pause(self.btime)

    def step_4_6(self):
        self.mc.fw(1)
        print("Step_46_FW-L:{}, R:{}".format(self.mc.l_speed,
self.mc.r_speed))
        dis_list = [0]*self.arr_size_1
        count = 0
        while True:
            dis_list[count%self.arr_size_1] = self.HC1.getDistance()
            print("Step_46_FW-L:{}, R:{},
D_R:{}".format(self.mc.l_speed, self.mc.r_speed,
self.HC1.getDistance()))
            sleep(self.div_time_1)
            count = count + 1
            dis_list_copy = dis_list.copy()
            dis_list_copy.sort()
            if dis_list[self.arr_size_1//2]>self.dis_to_fence_1:
                self.mc.pause(self.btime)
                self.mc.circling("R", 0, 1)
                self.gs.angle_record_start()
                while self.gs.angle_change() > -90:
                    print("Step_46_R90-L:{}, R:{},
Angle:{}".format(self.mc.l_speed, self.mc.r_speed,
self.gs.angle_change()))
                    sleep_us(20)
                break
                # 陀螺仪 R90 度
        self.mc.pause(self.btime)


    def step_5(self):
        self.mc.fw(1)
        print("Step_5_FW-L:{}, R:{}".format(self.mc.l_speed,
self.mc.r_speed))
        dis_list = [100]*self.arr_size_2
        count = 0
        while True:
            dis_list[count%self.arr_size_2] = self.HC2.getDistance()
            print("Step_5_FW-L:{}, R:{},
D_R:{}".format(self.mc.l_speed, self.mc.r_speed,
self.HC1.getDistance()))
            sleep(self.div_time_2)
```

```
            count = count + 1
            dis_list_copy = dis_list.copy()
            dis_list_copy.sort()
            if dis_list[self.arr_size_2//2]<self.dis_to_fence_2:
                self.mc.pause(self.btime)
                self.mc.circling("L", 0, 1)
                self.gs.angle_record_start()
                while self.gs.angle_change() < 90:
                    print("Step_5_L90-L:{}, R:{},
Angle:{}".format(self.mc.l_speed, self.mc.r_speed,
self.gs.angle_change()))
                    sleep_us(20)
                break
                # 陀螺仪 L90 度
        self.mc.pause(self.btime)


    def step_7(self):
        self.mc.fw(1)
        dis_list = [100]*self.arr_size_2
        count = 0
        while True:
            dis_list[count%self.arr_size_2] = self.HC2.getDistance()
            print("Step_7_FW-L:{}, R:{},
D_F:{}".format(self.mc.l_speed, self.mc.r_speed,
self.HC1.getDistance()))
            sleep(self.div_time_2)
            count = count + 1
            dis_list_copy = dis_list.copy()
            dis_list_copy.sort()
            if dis_list[self.arr_size_2//2]<self.dis_to_fence_3:
                self.mc.pause(self.btime)
                self.mc.circling("L", 0, 1)
                self.gs.angle_record_start()
                while self.gs.angle_change() < 90:
                    print("Step_7_L90-L:{}, R:{},
Angle:{}".format(self.mc.l_speed, self.mc.r_speed,
self.gs.angle_change()))
                    sleep_us(20)
                break
                # 陀螺仪 L90 度

        self.mc.fw(1)
        dis_list = [100]*self.arr_size_2
```

```
        count = 0
        while True:
            dis_list[count%self.arr_size_2] = self.HC2.getDistance()
            print("Step_7_FW-L:{}, R:{},
D_F:{}".format(self.mc.l_speed, self.mc.r_speed,
self.HC1.getDistance()))
            sleep(self.div_time_2)
            count = count + 1
            dis_list_copy = dis_list.copy()
            dis_list_copy.sort()
            if dis_list[self.arr_size_2//2]<self.dis_to_bin:
                sleep(1)
                print("Step_7_Open")
                self.sg90.value(0)
                self.sg90.value(1)
                sleep_us(2500)
                self.sg90.value(0)
                self.mc.pause(self.btime)
                sleep_us(20)
                break



    def step_8(self, angle):
        self.mc.circling("R", 0, 1)
        self.gs.angle_record_start()
        while self.gs.angle_change() > -self.angle:
            print("Step_8_Langel-L:{}, R:{},
Angle:{}".format(self.mc.l_speed, self.mc.r_speed,
self.gs.angle_change()))
            sleep_us(20)
        # 陀螺仪 R angle 度
        self.mc.pause(self.btime)


if __name__ == "__main__":
    # Initialization
    print(1)
    tim_A = Timer(Timer.TIMER0, Timer.CHANNEL0, mode=Timer.MODE_PWM)
    tim_B = Timer(Timer.TIMER0, Timer.CHANNEL1, mode=Timer.MODE_PWM)

    pwm_A = PWM(tim_A, freq=1e5, duty=0, pin=29)    # Y5
    fm.register(26, fm.fpioa.GPIO0) # Y7
    fm.register(27, fm.fpioa.GPIO1) # Y6
```

```
    in_1A = GPIO(GPIO.GPIO0, GPIO.OUT)
    in_2A = GPIO(GPIO.GPIO1, GPIO.OUT)

    pwm_B = PWM(tim_B, freq=1e5, duty=0, pin=31)    # X10
    fm.register(28, fm.fpioa.GPIO2) # Y8
    fm.register(30, fm.fpioa.GPIO3) # X9
    in_1B = GPIO(GPIO.GPIO2, GPIO.OUT)
    in_2B = GPIO(GPIO.GPIO3, GPIO.OUT)

    mc = motor_controller_unblocking(pwm_A, in_1A, in_2A, pwm_B,
in_1B, in_2B)

    fm.register(16, fm.fpioa.GPIO4) # Y1
    fm.register(18, fm.fpioa.GPIO5) # Y2
    fm.register(19, fm.fpioa.GPIO6) # Y3
    fm.register(20, fm.fpioa.GPIO7) # Y4

    fm.register(10, fm.fpioa.GPIOHS0)
    sg90 = GPIO(GPIO.GPIOHS0, GPIO.OUT)

    sg90.value(0)
    sg90.value(1)
    sleep_us(2500)
    sg90.value(0)


    trig1 = GPIO(GPIO.GPIO4, GPIO.OUT)
    echo1 = GPIO(GPIO.GPIO5, GPIO.IN)
    trig2 = GPIO(GPIO.GPIO6, GPIO.OUT)
    echo2 = GPIO(GPIO.GPIO7, GPIO.IN)

    HC1 = HCSR04(trig1, echo1)
    HC2 = HCSR04(trig2, echo2)

    gs = gyroscope()



    # run
    p2 = patio2(mc, gs, HC1, HC2, sg90)

    p2.step_7()

    # a = p2.step_4()
    # print(a)
```

main.py (patio1)

```
import sensor, image,lcd
from machine import Timer,PWM
import time
from time import sleep, sleep_us, sleep_ms
import utime
from Maix import GPIO
from fpioa_manager import fm
import _thread
import lcd
from machine import UART
from radar import *
from gyroscope import *
from motor_controller import motor_controller_unblocking
import struct
sensor.reset()
sensor.set_pixformat(sensor.GRAYSCALE)
sensor.set_framesize(sensor.QVGA)
sensor.skip_frames(time = 2000)
# sensor.set_vflip(True)

img=sensor.snapshot()
width,height=img.width(),img.height()
print(width,height)
ROI_devide=(0,60,20,int(height/2))
clock = time.clock()
section=16
real_width=int(width/section)
half=int(real_width/2)
sleep_ms(20)
door_bias=0
bridge_bias=0

def degree(A, B):
    x = B - A
    d_angle = min(abs(x), 360-abs(x))
    if 0<=x<180 or -360<=x<-180:
        d = 'L'
    else:
        d = 'R'
    return d_angle, d
```

```
tim_A = Timer(Timer.TIMER0, Timer.CHANNEL0, mode=Timer.MODE_PWM)
tim_B = Timer(Timer.TIMER0, Timer.CHANNEL1, mode=Timer.MODE_PWM)
pwm_A = PWM(tim_A, freq=1e5, duty=0, pin=29)    # Y5
fm.register(26, fm.fpioa.GPIO0) # Y7
fm.register(27, fm.fpioa.GPIO1) # Y6
in_1A = GPIO(GPIO.GPIO0, GPIO.OUT)
in_2A = GPIO(GPIO.GPIO1, GPIO.OUT)

pwm_B = PWM(tim_B, freq=1e5, duty=0, pin=31)    # X10
fm.register(28, fm.fpioa.GPIO2) # Y8
fm.register(30, fm.fpioa.GPIO3) # X9
in_1B = GPIO(GPIO.GPIO2, GPIO.OUT)
in_2B = GPIO(GPIO.GPIO3, GPIO.OUT)

mc = motor_controller_unblocking(pwm_A, in_1A, in_2A, pwm_B, in_1B,
in_2B)

fm.register(16, fm.fpioa.GPIO4) # Y1
fm.register(18, fm.fpioa.GPIO5) # Y2
fm.register(19, fm.fpioa.GPIO6) # Y3
fm.register(20, fm.fpioa.GPIO7) # Y4

fm.register(10, fm.fpioa.GPIOHS0)
sg90 = GPIO(GPIO.GPIOHS0, GPIO.OUT)

sg90.value(0)
sg90.value(1)
sleep_us(2500)
sg90.value(0)


trig1 = GPIO(GPIO.GPIO4, GPIO.OUT)
echo1 = GPIO(GPIO.GPIO5, GPIO.IN)
trig2 = GPIO(GPIO.GPIO6, GPIO.OUT)
echo2 = GPIO(GPIO.GPIO7, GPIO.IN)

HC1 = HCSR04(trig1, echo1)
HC2 = HCSR04(trig2, echo2)

gs = gyroscope()

def turn(B, delta=0.35):
    btime = 0.5
    A = gs.get_angle()
```

```
    if A==B:
        return True
    d_anglei, di = degree(A, B)
    while True:
        d_anglen, dn = degree(gs.get_angle(), B)
        mc.circling(di, 0, 0.5*d_anglen/d_anglei+delta)
        if dn!=di:
            break
    mc.pause(btime)

def radar_move(HC, com, dis, dif=0.0248):
    print(dif)
    arr_size_1 = 10
    div_time_1 = 0.02
    dis_to_fence_1 = 50
    btime = 0.5
    if HC=='F':
        gd = lambda: HC2.getDistance()
    else:
        gd = lambda: HC1.getDistance()
    print('move begin')
    mc.fw(0.8, dif)
    count = 0
    if com=='>':
        dis_list = [0]*arr_size_1
        while True:
            dis_list[count%arr_size_1] = gd()
            print(dis_list[count%arr_size_1])
            sleep(div_time_1)
            count = count+1
            dis_list_copy = dis_list.copy()
            dis_list_copy.sort()
            if dis_list[arr_size_1//2]>dis:
                break
    else:
        dis_list = [1000]*arr_size_1
        while True:
            dis_list[count%arr_size_1] = gd()
            print(dis_list[count%arr_size_1])
            sleep(div_time_1)
            count = count+1
            dis_list_copy = dis_list.copy()
            dis_list_copy.sort()
            if dis_list[arr_size_1//2]<dis:
```

```
                break
    if HC=='R':
        sleep(1.5)
    mc.pause(btime)
    print('move end')

# uart = UART(3, 115200)
"""
def send_data_packet(line_bias,door_bias,bridge_bias,status):
    data =struct.pack("<bbhhhh",              #格式为四个字符俩个短整型(2
字节)
                  0x29,                       #帧头 1
                  0x12,                       #帧头 2
                  int(line_bias), # up sample by 4    #数据 1,低位在前
                  int(door_bias), # up sample by 4    #数据 2.
                  int(bridge_bias), # up sample by 4    #数据 2.
                  int(status) # up sample by 4    #数据 2.
                  )
    uart.write(data)
"""
#求目标颜色中值
def find_tar(meanvalue,section):
    allsum=sum(meanvalue)

    sum1=0
    for i in range(section):
        if sum1<=allsum/2:
            sum1+=meanvalue[i]
        elif i==section-1:
            return i
        else:
            return i-1

def step_4():
    radar_move('F','<',40, dif=0.0508)
    turn(270)

def step_5():
    # self.radar_move('R','>',100, 0.0388)
    radar_move('F','<',40, 0.0508)
    turn(0)
    radar_move("R", '<', 20, 0.0518)

#洗数据
```

```
#可以加强归一化和拉大相邻差，但是会被影子影响严重
def clean(data,threshold_shit=None):
    if threshold_shit is None:
        #data[0]=0
        #data[-1]=0
        #摄像头左右两侧图像变形且像素变小
        avg=sum(data)/(len(data))
        if max(data)<190:
        #下午太阳后续白线像素和，下于就是寻黑线，反之白线
            for i in range(len(data)):
                if data[i]<1:
                    data[i]=0
            return data
        else :
            for i in range(len(data)):
                if data[i]<avg-5:
                    data[i]=0
            return data

    else :
    #预先设定阈值  不好用
        for i in range(len(data)):
            if data[i]>threshold_shit:
                data[i]=0
        return data

if __name__ == "__main__":
    sleep(3)
    gs.set_zero()
    while(True):
        data=[]
        clock.tick()
        img = sensor.snapshot()
        img = img.rotation_corr(z_rotation=-90)
        kernel_size = 1
        kernel = [-1, -1, -1,\
        -1, +8, -1,\
        -1, -1, -1]
        # img.find_edges(image.EDGE_CANNY,threshold=(80,160))
        img = img.morph(kernel_size, kernel)
        for i in range(section):
            ROI_cal=(i*real_width,ROI_devide[1],ROI_devide[2],ROI_devi
de[3])
            #print(ROI_cal)
```

```
            #img.draw_rectangle(ROI_cal)
            statistics=img.get_statistics(roi=ROI_cal)
            data.append(statistics[0])
            img.draw_rectangle(ROI_cal)

        #print('avg',sum(data)/(len(data)-2))
        # print(data,clean(data))
        #可以选择预设参数，这需要拿着车手动滚一圈
        #index=find_tar(clean(data,threshold_shit=130),section)
        index=find_tar(data,section)
        if index==None:
            index=section/2

        img.draw_cross(int(real_width*(index+1)),120,color=255,thickne
ss=10)
        direct=index+1-section/2
        print(direct)
        expected = 0
        actual = direct
        error_n = 0
        error_b = 0
        extra_drop = 0.1
        expected = 0
        kp = 0.06
        ki = 0.0001
        kd = 0.06
        sum_error = 0
        d_error = 0
        for i in range(10):
            err =  actual - expected
            error_b = error_n
            error_n = err
            d_error = error_n - error_b
            sum_error += err
            dif = kp * err + ki * sum_error + kd * d_error
            mc.run(0.35+dif, 0.35-dif)
            print("Speed: L:{}, R:{}".format(0.35+dif, 0.35-dif))
            sleep_ms(7)
            actual += dif - extra_drop
            err = expected - actual
        dis = HC2.getDistance()
        dis2 = HC1.getDistance()
        if dis <= 50 and dis2 > 500:
            step_4()
```

```
            step_5()
            break
```