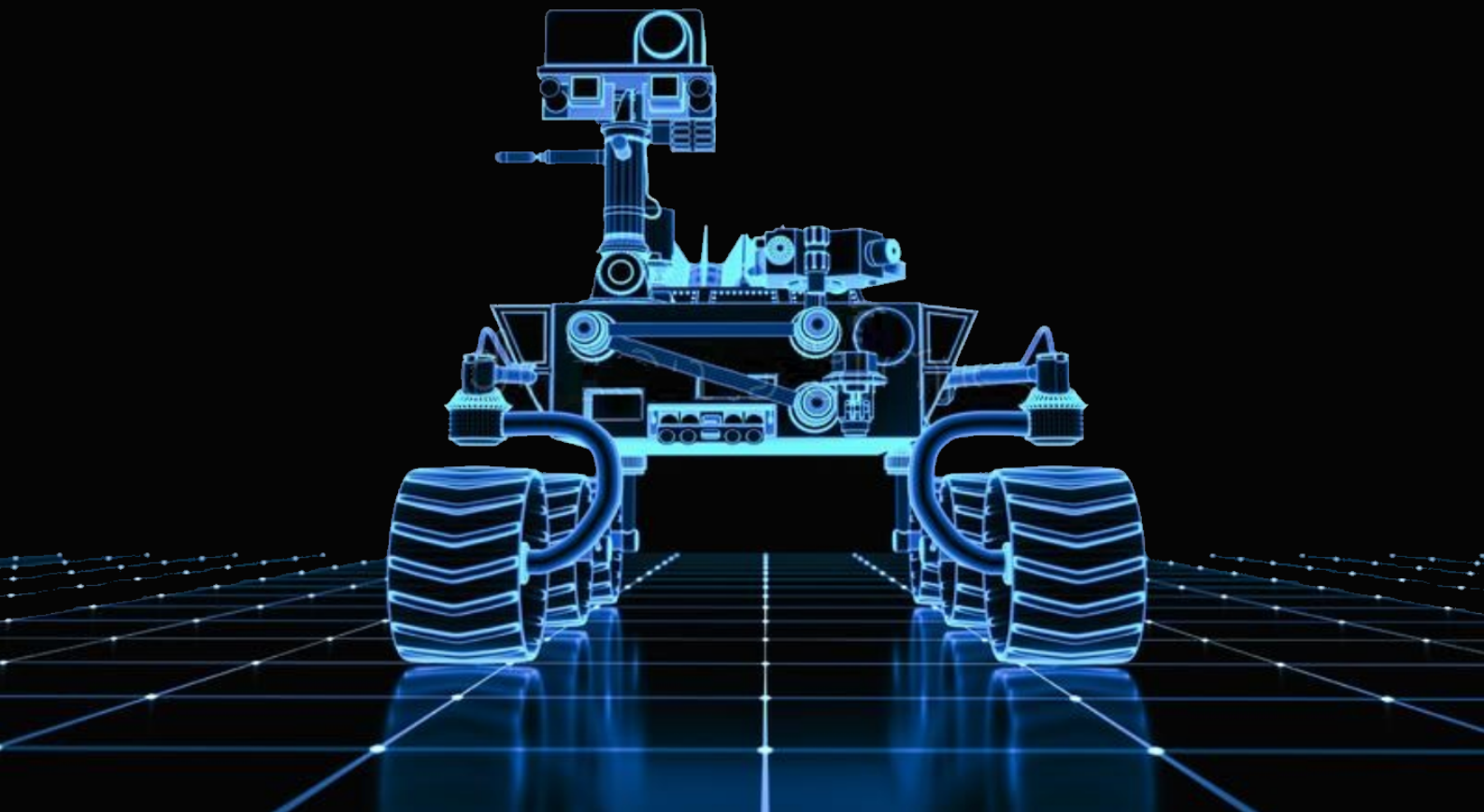
Team Design and Project Skills
Final Report

Team 10
Forza Horizon



Contents

List of Figures

List of Tables

Abstract

This report presents the research process of the car designed to complete the tasks required by the course Team Design Project and Skills, and the tasks include line tracing, obstacle detecting, bridge crossing, mechanical arm control, image recognition and so on. To fulfill these tasks, team management and technical methods are used simultaneously. As for team management, a divisional structure of two divisions is applied, Gantt chart and timeline graph are used to manage team tasks and process. As for the technical methods, the car is controlled by the MCU STM32F103RCT6, and the OpenMV is also used to manage the recognition of shape and object and the ultrasonic distance detecting problems. Many peripherals are chosen to complete the tasks, including the ultrasonic sensor, the Hall encoder, the mechanical arm and so on, which are controlled directly by the STM32. After many field tests, the car can achieve the tasks required independently, but sometimes it might need help to return to the right path. It is believed that this pitfall arose mainly because of the imperfect line tracing algorithm and the changing field situation. In conclusion, through wise team management and carefully choosing car technical plans, the course tasks are achieved successfully.

1 Introduction

In this team design project report, it could be divided into four major sections according to the teamwork, Car Control Tasks, Image Recognition tasks, and the Communication part, which is classified by the group structure and the task quality. In the first section, the basic information about our team is given, including the team structure, the individual responsibility, the team task progress with a timeline and a Gantt Chart and our team budget. As for the second large section, the Car Control group tasks are described in detail, in which the group is responsible for the circuit and PCB design, and peripherals utilization related to the main control STM32F103RCT6 that contain the mechanical structure design, ultrasonic sensor realization, the mechanical arm realization and so on. In the third large section, the Image Recognition group tackles theoretical and practical problems related to the OpenMV module, including the image, shape and object recognition problems, which are presented in detail with core code and sufficient analysis given in each subsection. As for the last part, the communication among the devices in the car and the wireless communication are implemented respectively. The group member analyzes the characteristic of data transmission among each key component in the car, and implements HC-12 and DS1302 module to achieve wireless communication. The details of implementation with code and overview are discussed in each subsection.

Seeing from the result, we can proudly say that the car could accomplish the given tasks both in patio 1 and 2 successfully, but not always perfectly due to the changing environment and the real-time situation. As for the defects, the most obvious defect lies in the line tracing part, as during the line tracing process, the car would vibrate along the path, sometimes the deviation from the desired route would be so large that the car would not be able to return to the right path. This might be due to the technical limitation of the OpenMV we chose, which leads to a not quite satisfying neural network training result. Besides the defects, there are also many trade-offs during the project, especially in the test field process, such as the wheel selection and the shaking car body, the increased car height and the on-car peripheral organization, and so on.

Below is the detail of our teamwork and the technical part.

2 Project Overview

2.1 Work Allocation

Our team are generally divided into two sub groups: Car Control Group and Image Recognition Group, which are responsible for robotic hardware design and visual recognition implementation, respectively. To be specific, tasks for Car Group involves the assembling of the car, circuits design, PCB design, positioning of different components, control of components including the ultrasonic sensor, the drive motor IC, the mechanical arm, the OLED screen, and

the gyroscope. As for the Image Group, members mainly deal with the visual information including the recognition of shapes, Apriltags, the basket, the target path. Particularly, the tasks of patrol and wireless communication are allocated to the Image Group with the consideration of work load.

Table 1 demonstrates the detailed individual task of each member.

Class	Responsibility	Member Name(s)	Student Number
General Manager	Team leader and Image group manager	Li Muquan	2510766L
	Car group manager	Liu Zhe	2510769L
Image Recognition group	Line tracing	Li Yuetai	2510959L
	Shape and object detection	Li Muquan	2510766L
	AprilTag Detection	Li Muquan	2510766L
	Ultrasonic distancing	He Yuechen	2511097H
	Communication between OpenMV and STM32	He Yuechen	2511097H
	Wireless Communication	Ma Yibo	2510733M
Car Control group	Dynamic control and Patrol logic	Liu Zhe	2510769L
	Circuit design	Zhang Shikun	2510790Z
	Structure design and building	Shui Jiajun	2510997S
	Ultrasonic sensing and OLED screen	Lin Meihong	2510850L
	PCB design	Liu Zheng	2510850L
	Mechanical arm control	Feng Guoyu	2511021F

Table 1: Team composition and individual responsibility

In Table 1, the dynamic control includes the motor, its drive and the gyroscope implementation.

2.2 Overall Project Plan and Our Progress

Our overall TDPS project plan is divided into 3 stages, as shown in Figure 2.1.

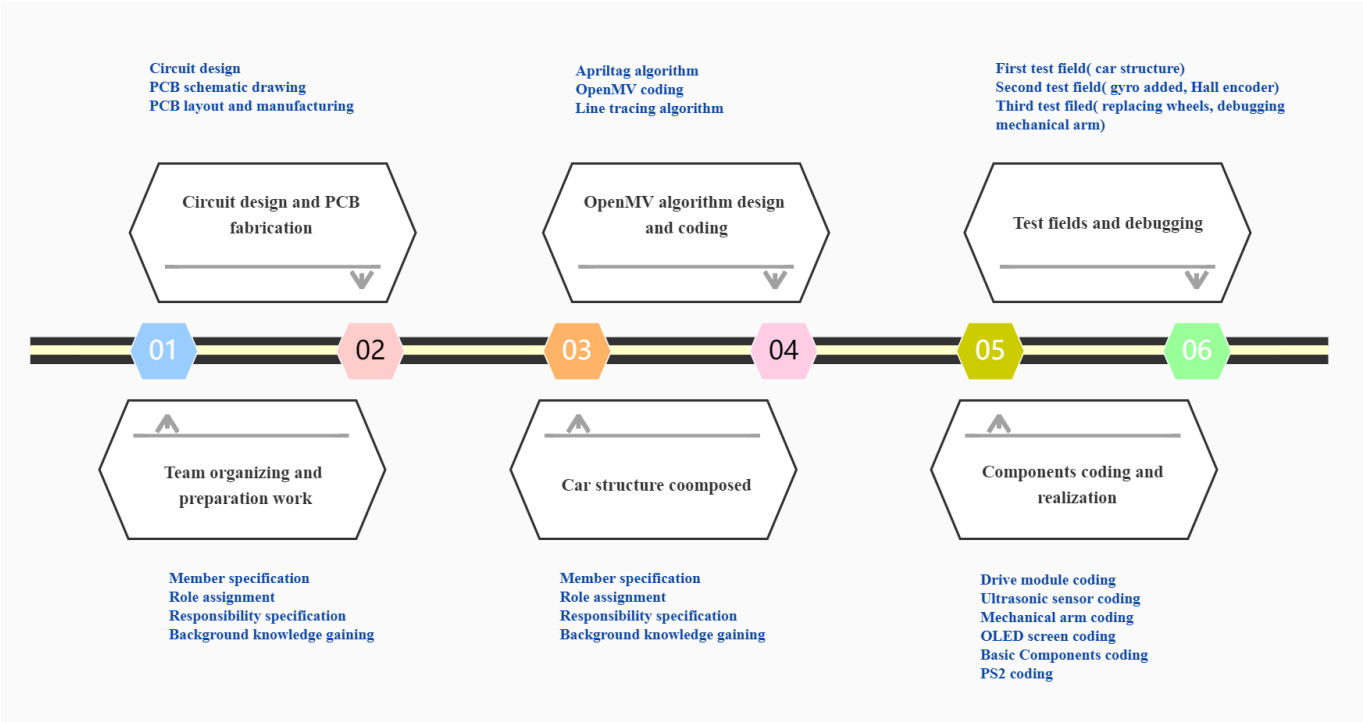


Figure 2.1: Team task: stage specification

Overall plan

In the first stage, team members are decided and well organized. After confirming the overall task requirements of the TDPS course, each member chooses and fully understands the tasks they are responsible for within the team.

Then, through careful comparison among different components within the same category and much discussion, the car components and materials are purchased. At the end of the first stage, the basic structure of the car rover could be drawn.

During the second stage, the components needed, relevant circuits required, and the PCB layout are realized. Before deciding on the circuits and their on-board distribution, a brief discussion was made to decide the components that are definitely required, then the circuits could be set and drawn. Since the PCB board was regarded as the foundation of the overall task, a lot of attention was paid and approximately 7 days were used on designing the PCB layout structure.

After finishing the PCB design, the two group members, the Image Recognition group and the Car Control group, are separated to implement the car more efficiently, and relevant tasks are allocated to each member. As mentioned in section 2.1, members within the Image recognition group mainly deal with the camera and OpenMV module, achieving tasks including image, shape and object recognition, such as the line tracing and the apriltag algorithm realization. As for the Car control group, members give solutions to the car structure design, positioning of the required modules, PCB and circuit design, and so on. In general, Image Recognition group is responsible for OpenMV module realization, while Car Control group focuses on the hardware structure and code implementation of the peripherals.

As for the last stage, code and results of the two groups were combined together, and the field test was carried out to verify function of the car, thus corresponding debugging as well as optimization were made to further refine overall performance of the car. However, we met many problems at this stage. During our first field test, it was found that the structure of the car was not strong enough and it failed to withstand the vibration caused by the uneven road surface during the movement of the car. In addition, the wheel was not large enough, and it seemed that the car had a problem crossing over the bridge. To solve the shaking problem, we used a thicker plate for the car, hoping to absorb some vibration by adding some inertial to the car body. To cross the bridge successfully, we chose a type of motor with larger torque, which is combined with the Hall encoder, as a result, we had to replace the original TT gear motor and used the Hall encoder to gain enough power for bridge crossing. After solving the above problems, we conducted our second trail. At this time, a problem became very obvious, which is the over turning problem of the car. To solve it, we added a gyroscope, gaining a better control over the angle the car turns. At the third test, we found out that the car was still not high enough to shoot the ball into the basket, thus we abandoned the 6cm Mecanum wheels and used a normal wheel with 8.5cm diameter. In addition, to further increase the height of the car, we used higher copper pillars between the layers of the car and a lengthened mechanical arm. Consequently, our car could reach the required height of the basket and complete all the required tasks.

2.3 Timetable and Gantt Chart

At the beginning of the project, the Gantt chart is used to help arrange task events and unify task achieving progress within the two groups, which shows the overall timeline and the duration of each task more clearly. Below is the Gantt chart we generated in this project, starting from the group formation and ending in the third test field. The blue square bars represent for team tasks, the green square bars represent for the Car Control Group tasks and the purple square bars for the Image Recognition Group tasks.

2.4 Cost of Modules

The following table shows the cost of each component we use to fulfill the tasks required, and this is a trade-off result between better accomplishing the tasks and staying within the budget.

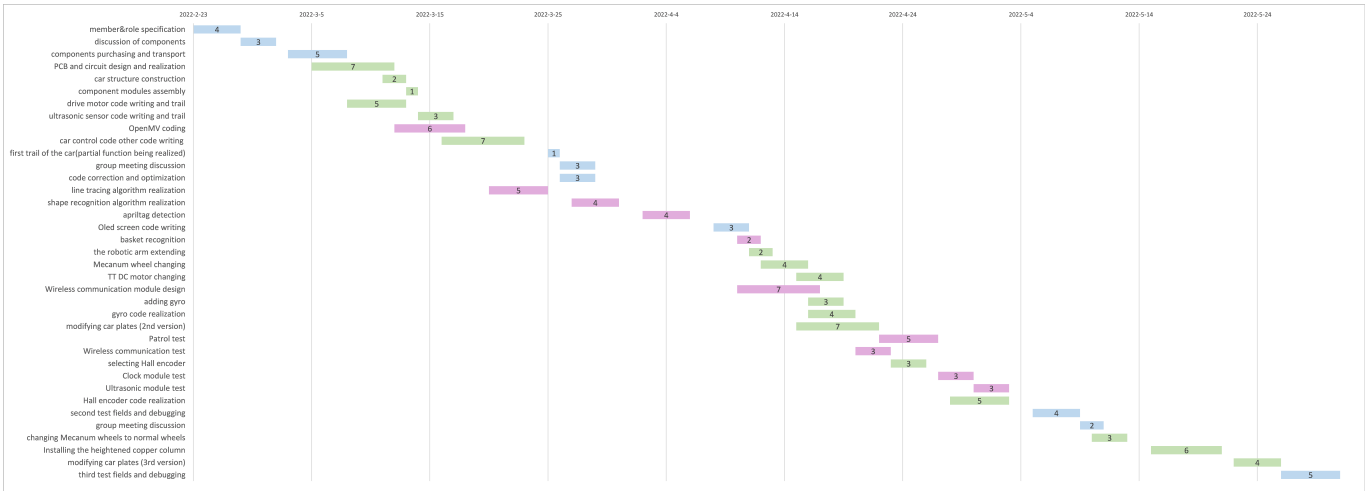


Figure 2.2: Gantt chart of allocated tasks

Device name	Price(CNY)
Mechanical arm(plus servo)	171.49
Wheels*4	51
Terminal wire	2.85
Gyro	13.46
OpenMV module	405
Chassis	296.36
Mechanical arm bracket and screws	26.6
Total Cost	966.8

Table 2: Devices prices summary

3 Car Control: Hardware Design and Peripheral Implementation

In this section, starting from the designed circuit, PCB design process, the mechanical structure of the car and its various peripheral implementation process will be discussed in detail following a logical order. To begin with, the logical description of the two patios are presented first to better understand the working process and the order of the tasks.

3.1 Patio Description

The smart car is supposed to do two different sets of tasks on the two patios. As a result of the design project, the robot should smoothly complete the assigned tasks in sequence on the two patios. This section will briefly describe the required tasks.

3.1.1 Patio 1 Description

A schematic diagram of the Patio 1 task is shown in Figure 3.1. For this patio, the car needs to :

- Move from the green starting point, follow the path formed by the unprocessed surface of the tile (black line in Figure 3.1) to the red destination;
- Identify and cross a wire mesh bridge;
- Identify and pass through the archway directly in front of the destination, shown in purple in Figure 3.1.

Logic Description of Patio 1

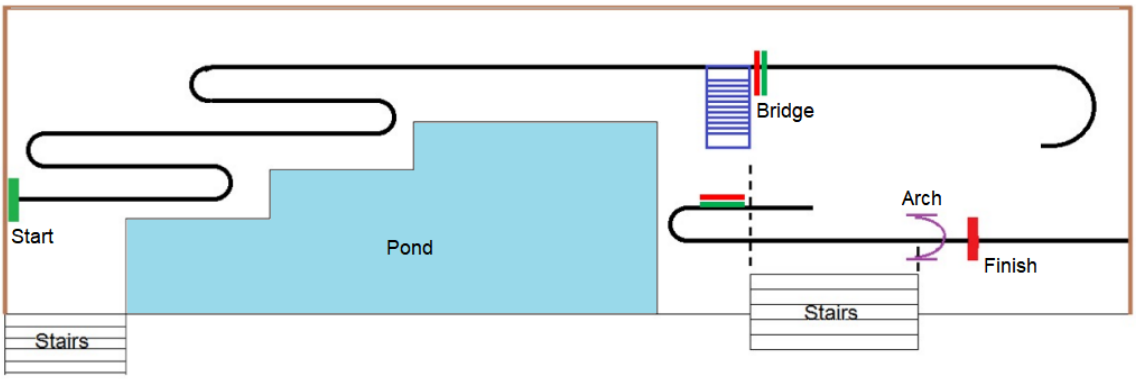


Figure 3.1: The schematic diagram of tasks to accomplish on Patio 1

The tasks required to complete in Patio 1 can be described as several cases in code. Below is a detailed code of the control logic of Patio 1. In which, case 1 represents for the line tracing function, case 2 for box-detecting part, thus the car will turn right and then in case 3, the 4 motors movement is specified to cross the bridge. In case 4, the car will turn left according to the current angle. In case 5, the car will stop in front of the arch. As for case 6 and case 7, they represent for breaking motion and the setting of the servo correspondingly.

```
switch(stage_now)
{       case 0:
        stage_now++;
    case 1:
        mode_change(0x01);
        stage_now++;
    case 2:
        App_control_car(); //line tracing
    case 3:
        motor_time(-50, 50, 50,-50,32);
    case 4:
        angle_motor(1,89);  //turn right
    case 5:
        delay_ms(5000);
        stage_now++;
    case 6:
        stage_now++;
    case 7:
        motor_time(100, -100, -100,100,1430); //cross bridge
    case 8:
        angle_motor(0,93);  //turn left
    case 9:
        delay_ms(5000);
        stage_now++;
    case 10:
        motor_time(100, -100, -100,100,1400);    //stop when reaching the arch
}
```

3.1.2 Patio 2 Description

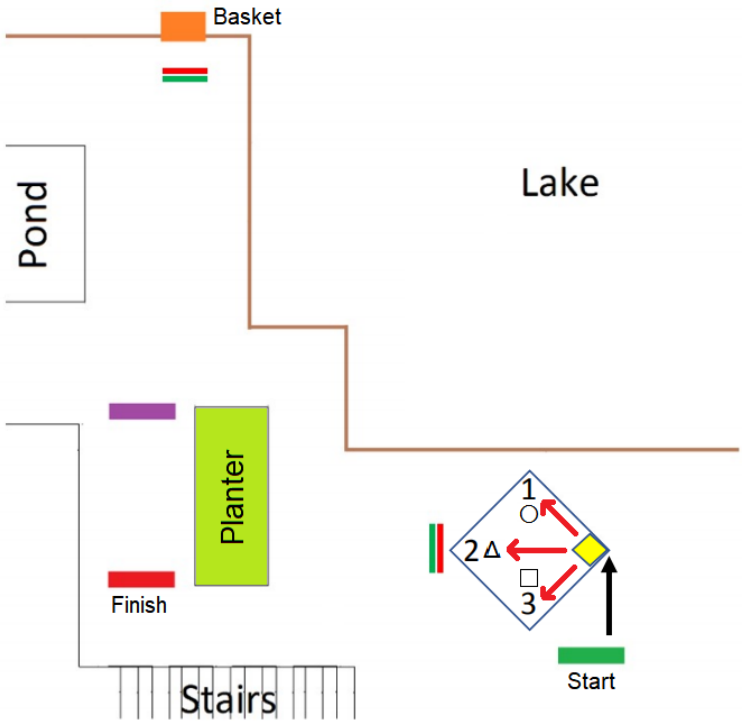


Figure 3.2: The schematic diagram of tasks to accomplish on Patio 2

A schematic diagram of the Patio 2 task is shown in Figure 3.2. On this patio, The car needs to:

- Move from the green starting point, then move to the right to the side of the colored square diamond;
- Determine the shape of the block and move to the block of the corresponding shape, each shape representing a different behavior;
- Move as far as possible to the basket by the lake (marked in orange in Figure 3.2), sail without breaking the general rules, and place the tennis ball in the basket;
- Move to the top edge of the planting area (marked in purple in Figure 3.2), as shown in Figure 3.2, and stop to send a message to the laptop. The message received on the laptop should contain:
  - Team name
  - Team member names
  - Time of day (24-hour clock).
- Continue tracing the line after successful transmission to reach the final red destination.

Logic Description of Patio 2

In the beginning of patio 2, the mechanical arm part is called to hold the ball, and subsequently, the car would move towards to the graph and turn left to face the diamond graph position. Then, the control is switched to OpenMV to complete image recognition by calling the function mode_change(). After recognizing the graph and knocking it down, the car would move far and be prepared to turn right to face the railing. After the car reaches the railing, the control would again be switched to OpenMV, and ultrasonic sensing would be used to let the car move against the railing, and finally reach the shooting position. Specifically, the ultrasonic sensing is responsible for the two left turns while the two right turns are done by setting the motor working time interval and turning direction. After reaching the position, the servo is called and the ball shooting is then completed. Consequently, the car would leave the shooting place and enter into the planter area to conduct wireless communication.

```
switch(stage_now)
            {
                case 1:
                    servor_time(1500,1350,400); //set servo fetch the ball
                case 2:
                    motor_time(50, -50, -50,50,1000); //straight to graph shown
                case 3:
                    mode_change(0x02);   //change the mode
                case 4://left image
                    angle_motor(0,90); //left side stand
                case 5:
                    motor_time(50, -50, -50,50,1000); //straight to the middle of the three
                case 6:
                    angle_motor(0,90); //turn left to face the stand
                case 7:
                    motor_time(50, -50, -50,50,1000); //go straight push down left stand
                case 8:
                    angle_motor(1,90); //turn right and get back
                case 9:
                    motor_time(50, -50, -50,50,2000); //move far and prepare to turn right
                case 10:
                    angle_motor(1,90); //turn right to face the pole
                case 11:
                    motor_time(50, -50, -50,50,2000); //move close to the pole
                case 12:
                    angle_motor(0,90); //turn left to approach the pole
                case 13:
                    stage_now++;//redundant bit
                case 16:
                    stage_now=39; //swift control to OpenMV
            //middle and right image cases are omitted here
                case 35:
                    stage_now++; //redundant
                case 38:
                    stage_now=39; //swift to OpenMV
                case 39:
                    mode_change(0x02);   //swift to OpenMV
                    stage_now++;
                case 40:
                    App_control_car(); //line tracing
                case 41:
                    motor_time(50, -50, -50,50,100);//the first OpenMV turn
                case 42:
                    angle_motor(1,90); //turn right
                case 43:
                    motor_time(50, -50, -50,50,400); //move forward
                case 44:
                    stage_now++;
```

```
                case 45:
                    App_control_car();
                    if(Distance1<=30)
                    {
                        stage_now++;
                        Set_Motor(0, 0, 0, 0);
                        Motor_Speed[0]=0; Motor_Speed[1]=0; Motor_Speed[2]=0;
                            Motor_Speed[3]=0;
                    }
                case 46:
                    angle_motor(0,90);    //Main control turn
                case 47:
                    App_control_car(); //line tracing
                case 56:
                    stage_now++;
                case 57:
                     servor_time(1000,1350,1000); //set the servo
                case 58:
                    servor_time(1000,1000,1000); //set the servo
                case 59:
                    servor_time(1500,1500,1000); //set the servo to complete ball shooting
                case 60:
                    angle_motor(0,90);    //turn left
                case 61:
                    servor_time(1500,1500,10000); //set the servo; move forward; prepare to send
```

3.2 Circuit System Design

Main Contributor: Zhang Shikun

In this section, the control circuit of the smart car will be designed so that it can complete the given tasks. The multiple design schemes of each module will be introduced, as well as the reasons why I choose the corresponding scheme, and finally form the final scheme of the circuit design of the smart car.

3.2.1 Problem Statement

The hardware system of the smart car is the physical basis for its realization of the target task, which is directly related to the reliability and stability of the smart car operation. According to the tasks, the smart car circuit of this project mainly includes a controller (MCU), a power supply voltage regulator circuit, a program download circuit, a motor drive module, a reserved functioning serial port (for connecting to ultrasonic waves, OpenMV) and other parts.

In this design, STM32F103RCT6 is used as the controller, TD1583 and AMS117-3.3 are used as the power supply voltage regulator module [1], CH340 is used as the USB conversion circuit, and TB6612 is used as the motor drive circuit [2], enabling the smart car to operate normally according to the task.

Below is the detailed chosen scheme of each module related to the circuit.

Procedure Outline

The circuit block diagram is shown in Figure 3.4.

As for the modules, considering the main modules on the car, firstly, the drive module includes four DC gear motors, two motor drives TB6612FNG, four encoders, and gyroscopes in which the motor drive is powered by the control board STM32F103RCT6 to control the speed and rotating direction of the motor, then the movement of motor is reflected on the wheel and the car eventually moves in a certain direction.

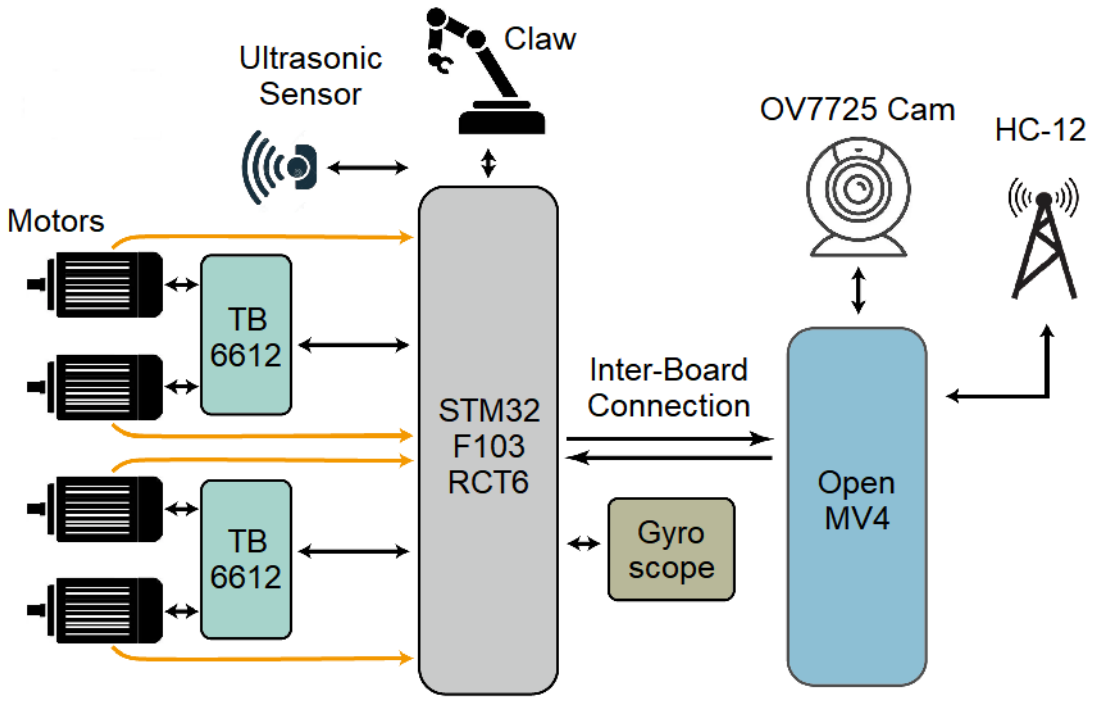


Figure 3.3: System Design Scheme

Secondly, a HC-SR04 sensor was used to function ultrasonic distance sensing. A mechanical arm with a servo is used to fetch objects while the servo would control the angle of the arm using PID algorithm. To realize image recognition and line tracing, the OpenMV module with a camera is used, while the infrared sensor is abandoned.

3.2.2 Intro to MCU: STM32F103RCT6

The main control chip STM32F103RCT6 used in this project is a 32-bit FLASH memory microcontroller based on the ARM Cortex M3 processor core produced by STMicroelectronics [1].

ARM (Advanced RISC Machine) is a British company specializing in chip IP design and licensing. Its products include ARM cores and various peripheral interfaces. The ARM core has three characteristics: low power consumption, high cost performance, and high code density.

At present, more than 90% of mobile phones, a large number of game consoles, handheld PCs and set-top boxes have adopted ARM processors. Many first-class chip manufacturers are authorized users of ARM, and ARM has become a recognized embedded microprocessor in the industry standard.

According to the naming convention, the STM32F103RCT6 is an enhanced general-purpose 64-pin 256K-byte FLASH microcontroller based on the ARM Cortex M3 processor core, with an operating temperature range from -40°C to 85°C [1]. STM32 MCU combines high performance, real-time digital signal processing, low power consumption and low voltage while maintaining high integration and ease of development. With a large number of software and hardware development tools, it represents one of the most powerful products in the industry and is ideal for a variety of small and medium-sized projects.

The STM32 series chip has the following advantages,

1. Extremely high performance: STM32 series chips are based on mainstream Cortex cores;

2. Flexible. All the IO ports on the board except the crystal oscillator are all led out, which is considered very convenient for expansion and use.

3. Rich on-board resources. There are more than ten kinds of peripherals and interfaces on board, increasing its possibility and functions.

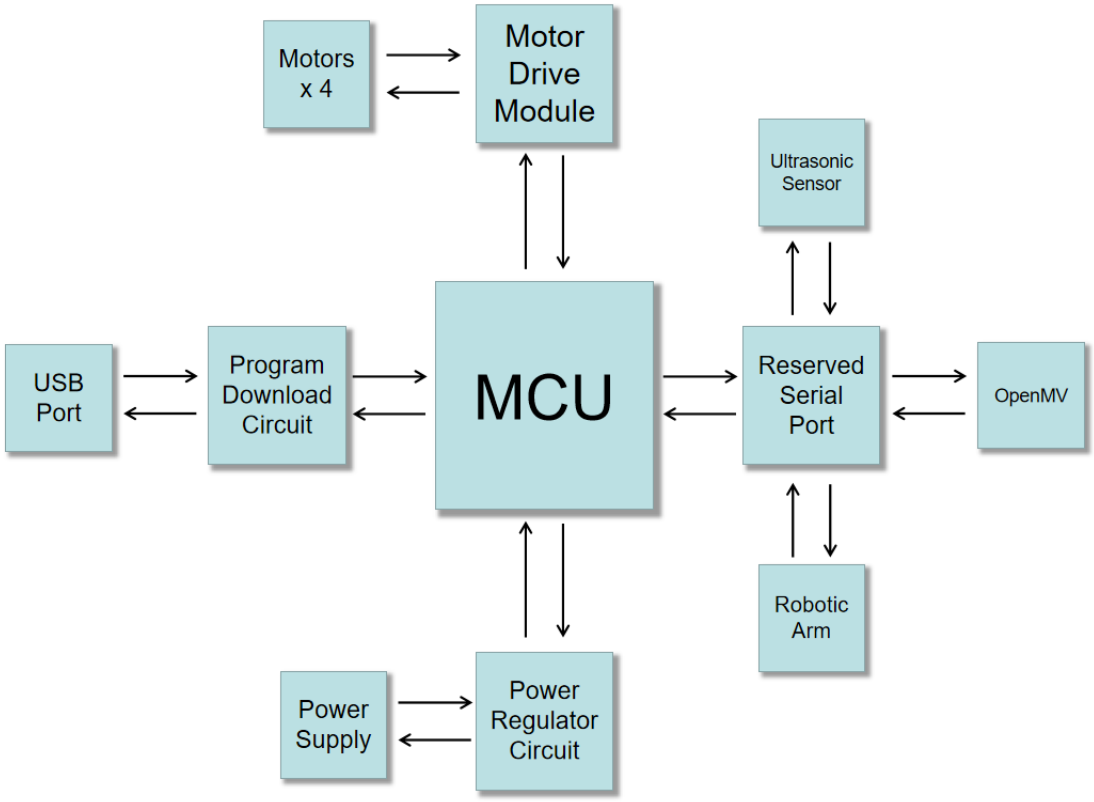


Figure 3.4: Block diagram of system structure

3.2.3 Motor Drive Circuit

The TB6612FNG is chosen as the drive module of the trolley motor to realize the control of the motor. The choice of chip determines the basis of the entire car power system, and also affects the layout design of the entire circuit board. For the whole project, starting to move is the first step in the true sense of the car, and it is one of the first problems that the project needs to solve.

After investigation, it was found that there are three common motor driver chips on the market, namely TB6612, L298N, and A4950 . In this project, TB6612 is used.

TB6612FNG is a DC motor drive module produced by Toshiba Semiconductor. It is an H-bridge integrated circuit based on MOSFET, and its efficiency is much higher than that of a transistor H-bridge. Its power supply voltage is 2.5∼13.5V, and the average current output by the H-bridge is 1.2A, with a maximum 3.2A. In addition, it possesses built-in thermal protection and low-voltage detection shutdown circuits [2].

In contrast, the space required for the L298N is too large due to the radiator installation, so it is not considered here. In comparison, L298N and A4950 have problems such as too much space required and too high driving voltage, so they are not suitable for the circuit we designed.

In summary, TB6612FNG is selected as the motor driver module, and the following circuit is shown in the graph below.

As shown in Figure 3.8, each TB6612FNG module is responsible for driving two motors A and B. AIN1, AIN2, BIN1, and BIN2 are responsible for the rotation directions of motors A and B respectively. PWMA and PWMB are the control signal input terminals of the motor, and the motor speed is controlled by changing the duty cycle of the PWM signal.

Besides, some LEDs are also designed in the circuit, which are connected in parallel at both ends of the motor to absorb the induced electromotive force generated by the change of the rotating state of the motor. These LEDs can also show the direction of rotation of the motor. Moreover, this design reserves the encoder port (JP1-4), in order to detect the rotational state of the motor.

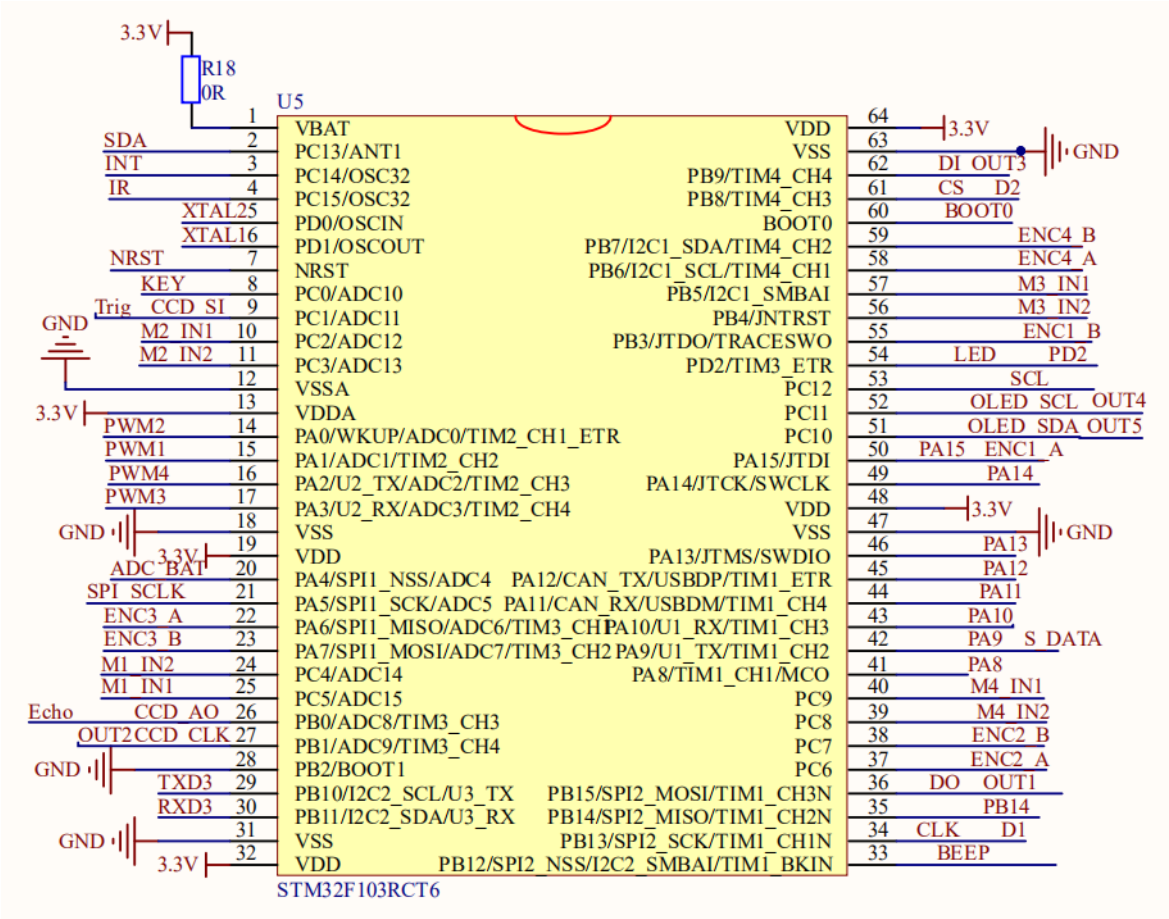


Figure 3.5: STM32F103RCT6 schematic diagram

3.2.4 Power Regulator Circuit

The voltage regulator circuit is a power supply circuit that can keep the output voltage approximately unchanged when the input grid voltage fluctuates or the load changes. According to the type of output current, it is divided into DC voltage regulator circuit and AC voltage regulator circuit. The voltage regulator circuit designed in this project is a DC voltage regulator circuit.

7.4V to 5V Circuit

The TD1583 is a 380 KHz single-fixed frequency monolithic step-down switch mode regulator with internal power MOSFETs. It delivers 3A continuous output current over a wide input supply range with excellent load and line regulation ability. In contrast, the maximum current of the LM7805 voltage regulator module is only 1.5A, and the design margin is too small. As a result, LM7805 is not selected.

5V to 3.3V Circuit

AMS1117 is a three-terminal linear regulator circuit with positive voltage output and low dropout voltage.

AMS1117 is divided into two versions, fixed voltage output version and adjustable voltage output version. As for the output voltage of the fixed output version, it can be: 1.8V, 3.3V and 5.0V, and the output voltage range that the adjustable voltage output version can provide is: 1.8 V∼5.5V, the output voltage accuracy is as high as 2%, and the maximum output current can reach 1A. In addition, AMS1117 integrates with overheat protection and current limiting circuits to ensure the stability of the chip and power system.

In this project, the fixed output version 3.3V (AMS1117-3.3) is used.

3.2.5 Decoupling Capacitor

The decoupling capacitor is a capacitor installed at the power supply end of the component in the circuit. This capacitor can provide a relatively stable power supply, and at the same time, it can reduce the noise coupled to the

pin name	illustration
PWMA	A motor control signal input terminal
AIN2	A motor input terminal 2
AIN1	A motor input terminal 1
YSTB	Normal working/Standby state control terminal
BIN1	B motor input terminal 1
BIN2	B motor input terminal 2
PWMB	B motor control signal input terminal
GND	Ground
VM	Motor drive voltage input（4.5V 15V）
VCC	logic level input（2.7V 5.5V）
PGND	Ground（Protective ground）
AO1	A motor output terminal 1
AO2	A motor output terminal 2
BO1	B motor output terminal 1
BO2	B motor output terminal 2

Table 3: TB6612FNG pin description

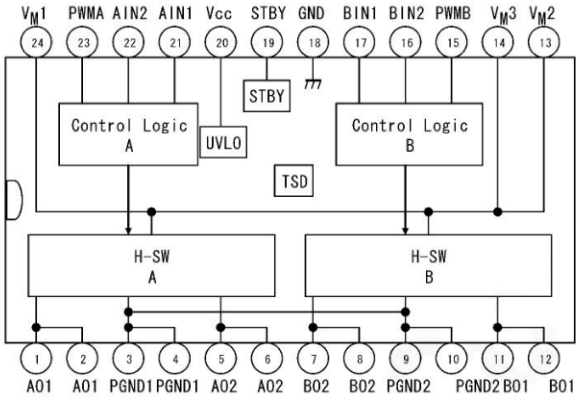


Figure 3.6: TB6612FNG block diagram

power supply end of the component, and indirectly reduce the noise influence to other components by the noise of this component.

Decoupling capacitors can effectively prevent parasitic oscillations caused by the positive feedback formed by the circuit through the power supply. That is to say, the decoupling capacitor can prevent the current impulse formed in the power supply circuit from affecting the normal operation of the circuit when the current size of the front and rear circuits changes.

The use of capacitive decoupling is also the main method to solve the problem of power supply noise. This method is very effective for improving the response speed of transient current and reducing the impedance of the power distribution system. Decoupling capacitors are generally placed at the power supply near the component to reduce the effect of wiring impedance on the filtering effect. Decoupling capacitors are mostly ceramic capacitors, whose value is determined by the fastest rising and falling speed of the voltage signal.

In this project, C1, C3, C4 and C7 mainly filter low frequency noise, while C2, C5 and C6 mainly filter high frequency noise.

In addition, in order to detect the battery voltage to determine whether the battery needs to be charged, a voltage divider circuit is designed at the battery shown as Figure 3.10.

As indicated in the graph, the ADC BAT pin transfers an analog voltage of 1/11 battery voltage to PIN20 of the STM32. After the battery is used for a period of time, the battery voltage decreases, which will cause the input voltage of this pin to drop. The voltage drops below the threshold to indicate that the battery needs to be charged.

Figure 3.7: TB6612FNG motor drive module outlook

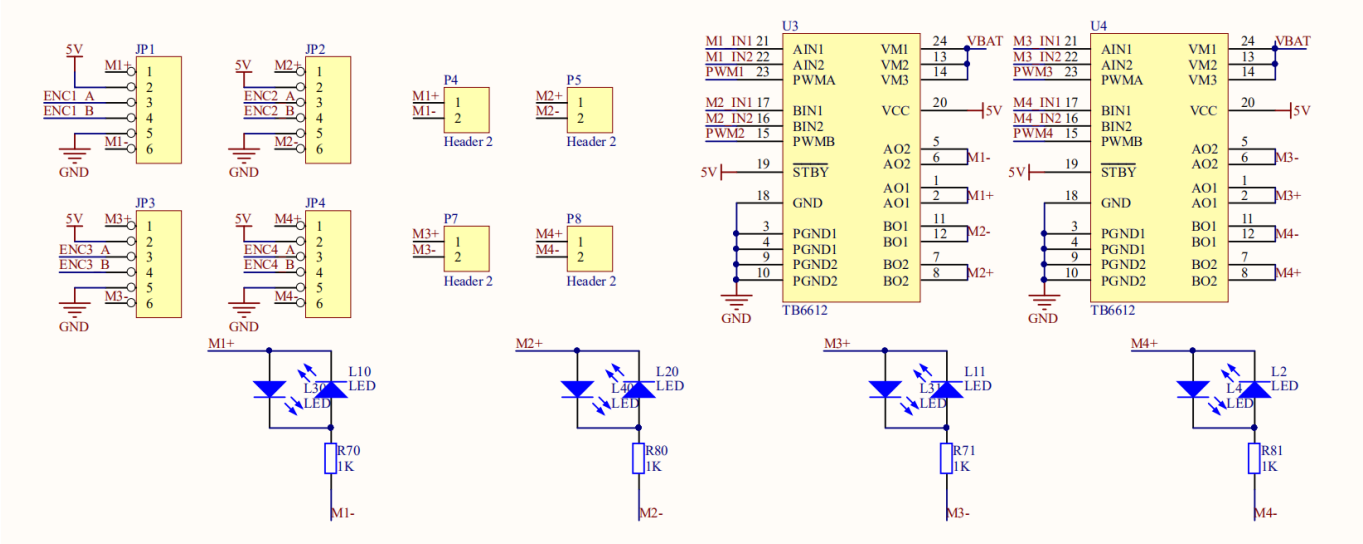


Figure 3.8: TB6612FNG motor drive circuit

3.2.6 Program Download Circuit

For the convenience of downloading the program from PC, a USB conversion circuit needs to be designed.

Commonly used USB-to-serial chips include CH340C, CP2102, PL2303, FT232, etc. The performance of CH340 is sufficient for general applications. The piracy of PL2303 in the current market is serious; the bulk price of CP2102 is more expensive than that of CH340, and it is packaged in QFN, which is not easy to solder. The FT232 is more expensive in bulk and is generally used in industrial-grade environments. To sum up, CH340C is chosen as the USB to serial port chip.

Function of BOOT

The working process of the Cortex M3 core after leaving the reset state is as follows:

1. Get the initial value of the stack pointer MSP from the address 0x00000000, which is the address of the top of the stack;

2. Fetch the initial value of the program pointer PC at address 0x00000004, which points to the first instruction that should be executed after reset;

The above process is automatically set up by the kernel to set the operating environment and execute the main program, so it is called the bootstrap process.

Although the kernel is fixed to access the 0x00000000 and 0x00000004 addresses, these two addresses can actually be remapped to other address spaces. Taking STM32F103 as an example, according to the level of BOOT0 and BOOT1 pins drawn by the chip, these two addresses can be mapped to internal FLASH, internal SRAM and system memory.

IN1	IN2	DC Motor
0	0	Brake
0	1	Forward Rotating
1	0	Reverse Rotating
1	1	Brake

Table 4: TB6612FNG truth table（motor-drive state）

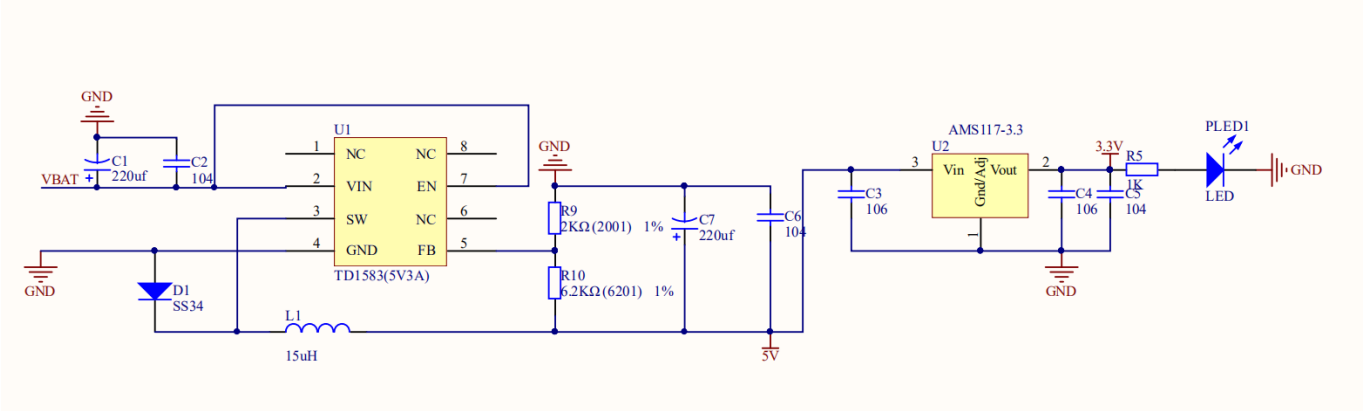


Figure 3.9: Power regulator circuit

Different mapping configurations are shown in the table below.

BOOT1	BOOT0	Registers mapped	0x00000000 maps to	0x00000004 maps to
x	0	Internal FLASH	0x08000000	0x08000004
0	1	System register	0x1FFFB000	0x1FFFB004
1	1	Internal SRAM	0x20000000	0x20000004

Table 5: Different mapping configurations of BOOT

After the kernel leaves the reset state, it will take the value from the mapped address to the stack pointer MSP and the program pointer PC, and then execute the instruction. We generally distinguish the bootstrapping process by the type of memory, such as the internal FLASH startup method, the internal SRAM startup method and System memory boot mode.

Internal FLASH Start method

When the BOOT0 pin is sampled to be low level after the chip is powered on, the addresses 0x00000000 and 0x00000004 are mapped to the first addresses of the internal FLASH 0x08000000 and 0x08000004.

Therefore, after the kernel leaves the reset state, it reads the content stored in the 0x08000000 address space of the internal FLASH, assigns it to the stack pointer MSP as the top address of the stack, and then reads the content stored in the 0x08000004 address space of the internal FLASH, and assigns it to the program pointer PC, as the address of the first instruction to be executed. With these two conditions, the kernel can start to read the instruction execution from the address pointed to by the PC.

System Memory Start Method

When the chip is powered on, the BOOT0 pin is high and BOOT1 is low, the kernel will obtain the MSP and PC values from 0x1FFFF000 and 0x1FFFF004 in the system memory for bootstrapping.

Program functions initiated in this mode are set by the factory. The system memory is a specific area inside the chip, which cannot be accessed by users. ST company solidifies a piece of code in the system memory before the chip leaves the factory, which is what we often call the ISP program, which is a ROM that cannot be modified after leaving the factory.

Therefore, when the system memory boot method is used, the kernel will execute the code. When the code is run, it will provide support for the ISP (In System Program), such as detecting the information transmitted by the

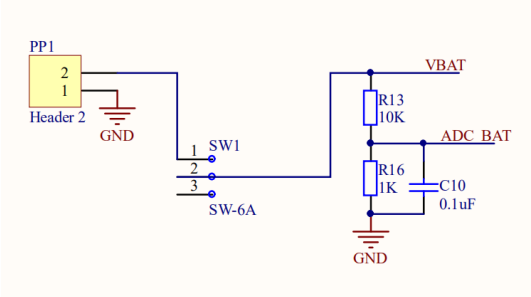


Figure 3.10: Voltage divider circuit

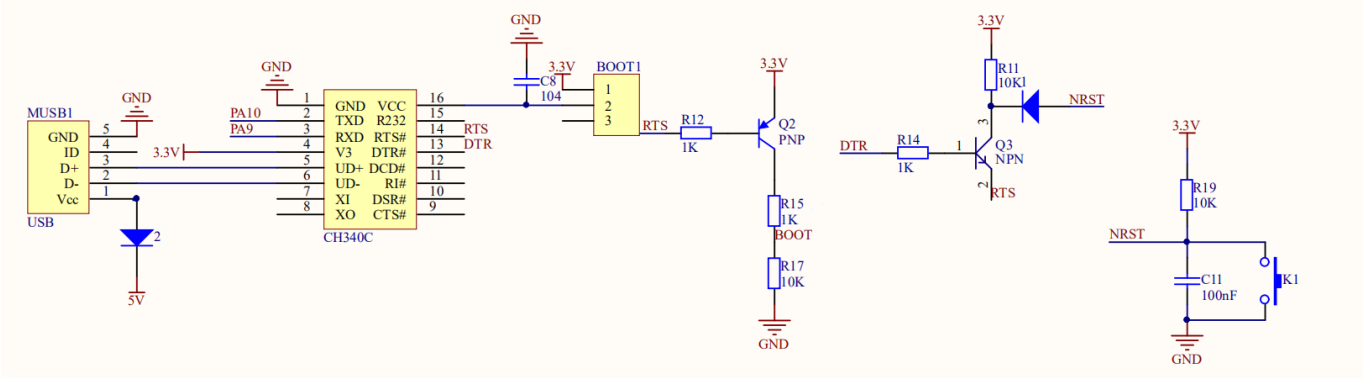


Figure 3.11: Program download circuit

USART1/2, CAN2 and USB communication interfaces, and based on the information, it will updates the content of its own internal FLASH to achieve the purpose of upgrading the product application, so this startup mode is also called ISP startup mode.

Generally speaking, when I chose this startup mode, it is to download the program from the serial port, because the BootLoader provided by the manufacturer provides the firmware of the serial port download program, and the program can be downloaded to the FLASH of the system through this BootLoader.

Internal SRAM Start Method

Similarly, when the BOOT0 and BOOT1 pins are sampled high after the chip is powered on, the addresses 0x00000000 and 0x00000004 are mapped to the first address of the internal SRAM, 0x20000000 and 0x20000004, and the kernel obtains the content from the SRAM space for bootstrapping.

In practical applications, the startup file starttup_stm32f10x.s determines what content is stored at addresses 0x00000000 and 0x00000004. When linking, the scatter-loading file .sct determines the absolute address of these contents, that is, whether to allocate to internal FLASH or internal SRAM. This mode can be used for program debugging.

If only a small place in the code is modified, then the entire FLASH needs to be re-erased, which is time-consuming. Consider starting the code from this mode (that is, in the memory of STM32) for fast program debugging, etc. After debugging, the program can be downloaded to SRAM.

In this project, the internal SRAM startup method will not be used, thus the Pin 28 of the MCU is connected to ground, which means BOOT1 is connected to a low level.

DTR: MODEM Contact output signal, data terminal is ready (DTR# stands for active low);

RTS: MODEM Contact output signal, request to send, (RTS# stands for active low)

When K1 is pressed, NRST outputs a low level and the microcontroller resets. The following discusses the output when K1 is disconnected:

1. When both DTR and RTS are not ready, Q2 is not turned on, Q3 is turned on, BOOT0 outputs a low level, and NRST (RST active low) outputs a high level. The MCU is in the internal FLASH startup mode and executes the program normally;

2. When DTR is ready and RTS is not ready, Q2 and Q3 are not turned on, BOOT0 outputs low level, NRST (RST active low) outputs high level. The MCU is still in the internal FLASH startup mode, and the program is executed normally;

3. When DTR is not ready and RTS is ready, Q2 and Q3 are both turned on, BOOT0 outputs high level, NRST (RST active low) outputs low level, i.e., reset. This state is an abnormal state,and it can be seen that adding Q3 can make the MCU forced to reset in an abnormal state.

3.2.7 Results and Effects

During the project process, it turns out that the circuits we design are sufficient to satisfy the car needs. And thanks to the program download circuit, we can directly download the code into the main control, which greatly adds convenience for car implementation. As for the power regulator circuits, because the car uses many peripherals that require different levels of voltages, the design of this circuirt regulates the power voltages of the various peripherals, such that we do not have to worry about exceeding the rated voltages. As for the decoupling capacitor circuit, it could work properly to reduce the noise produced by other components and produce a relatively stable power output.

It is understood that the circuit design is the fundamental part of the car hardware implementation, therefore, many related circuit design materials should be read and the types of circuits we might need in this car project should be considered. As for the organization of the circuits, it is discussed with our teammate that the circuits are arranged in a way to maximize utilization convenience and use relatively small area.

3.3 PCB Design and Implementation

Main Contributor: Liu Zheng

In this subsection, the process from designing PCB to PCB wiring implementation will be presented. It is mainly divided into two parts. The first part is mainly responsible for drawing the PCB sketch design based on the circuit design introduced in the previous section, and the second part is to convert the designed PCB sketch into a 3D model layout.

3.3.1 Schematic Drawing Introduction

The schematic diagram is considered as the pre-work of the PCB design, and appropriate components are selected according to the circuit design that was introduced previously. Finally, the reproduction of the schematic diagram is represented on the Altium Designer(AD).

The designed circuit is reproduced in AD using the resource library (built in AD and can be searched on the Internet), which is the preparation for the next step of PCB design. Schematic design is not only the logic detection of the designed circuit, but also the preparation for PCB layout. This task is the basis of the car structure and the key to transforming the design ideas into entities.

3.3.2 Comparison of Softwares

There are mainly three PCB drawing tools on the market: Altium Designer, Mentor Pads and Cadence Allegro. These three tools occupy most of the market, and each has their own unique advantages. According to the needs of the car task, Altium Designer is chosen as the tool for schematic drawing.

Altium Designer

The feature of AD is that it integrates all functions, from schematic design to PCB Layout and simulation, into one software that can meet nearly all the required design specifications.

Schematic interface of AD is simple and easy to use, and its hierarchical schematics are very useful when working on complex schematic designs. When calling the schematic diagram, the corresponding PCB package can be directly previewed. After the schematic design is completed, PCB design could be realized without importing or exporting the netlist, which is quite convenient. [3]

The market positioning of AD is for some simple boards, such as single-chip boards, and some relatively ordinary boards. This software is used more in low-end product design, most of which are simple boards with 2 or 4 layers.

Mentor Pads(PADS)

The software interface of PADS is not complicated with only a few options, thus it is not difficult to get started. The main interface is divided into three parts, including: schematic tool PADS Logic, PCB tool PADS Layout and automatic routing tool PADS Router. They are used to make schematics, layout, and wiring respectively. It is easy to operate, quick to use, and can generate high-quality products. However, it does not have simulation itself. When making high-speed boards, it should be used in combination with other special simulation tools, such as hyperlynx. In addition, the resources of PADS is not strong enough, which seems a little bit out of style.

Cadence Allegro

Cadence mainly uses two platforms in PCB design, namely Orcad and Allegro, which are related by netlist. The former is a schematic design which has the most complete library management tool CIS within industry. The library contains the price, cost, data, size and other information of the associated devices. The latter has a rigorous rule management, which ensures high availability of the product after setting the rules. Although the wiring function is not the best, it is also a first-class product, which is more suitable for high-speed and high-density designs.[3]

However, the Allegro software is not very friendly to newcomers. The creation of its package library can be said to be extremely troublesome. PADs and Symbols need to be created in different tools. Moreover, Allegro has poor compatibility with other softwares, and PCB files drawn by other EDA tools cannot be directly opened in Allegro, requiring multiple conversions.[3]

Eventually, Altium Designe is considered the most suitable. Compared with Mentor Pads, AD has a more comprehensive resource library and related resource searcher, and the operation interface is clearer and easier to learn and operate. Compared with Cadence Allegro, AD has a lower difficulty to get started, which can effectively reduce learning costs and improve efficiency.

3.3.3 Drawing Procedure and Results

The outline procedure for schematic design is as follows:

1. Create the schematic diagram file;
2. Find and place components;
3. Draw circuit;
4. Validation and error checking.

Below is the schematic diagram designed showing in Altium Designer.

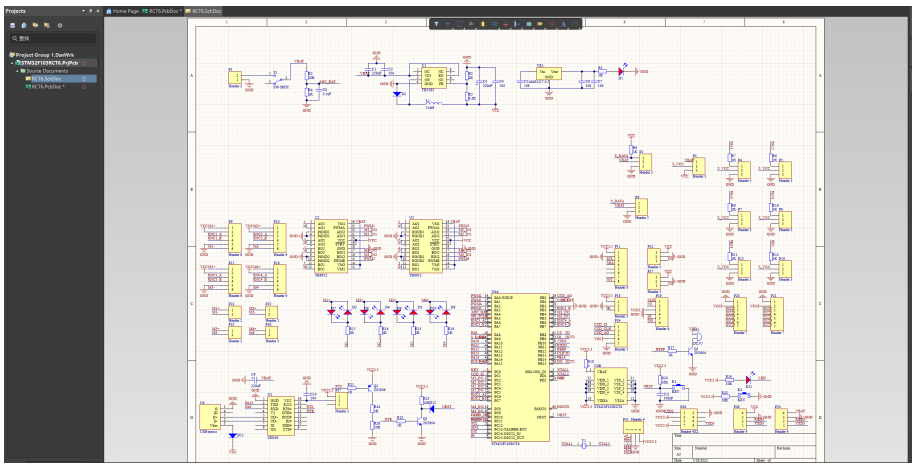


Figure 3.12: PCB schematic drawing

In the validation and error checking stage, ERC check is performed to check the electrical properties of the layout, such as whether the substrate is correctly connected to the power supply or ground, whether the gate is floating and so on.

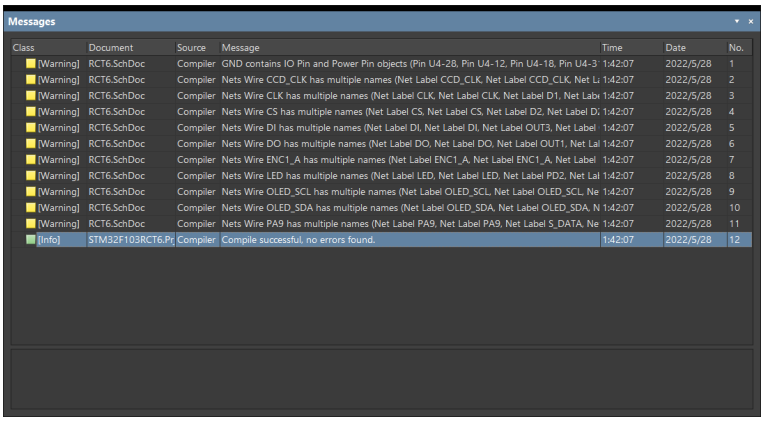


Figure 3.13: Successful ERC check

Figure 3.13 shows that ERC check is successful and thus the design is valid. It is worth- noting that warning does not affect as an error, which means Figure 3.13 shows that the schematic has been successfully verified and no error is reported.

In the step of placing components, some components cannot be found in relevant resources on the network, which requires drawing relevant schematic diagrams by myself, such as the Header 6 component shown in Figure 3.14.

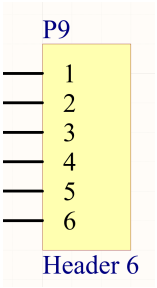


Figure 3.14: Self-drawn component Header 6

In the process of verification and error checking, in order to avoid erroneous error reports, some test points are set to ignored the ERC, thus reducing the workload of checking errors. For example, the error in Figure 3.15 shows that there is a single-ended network net with only one pin. After checking the schematic diagram, it is found that the port in this diagram is indeed a single-ended network, so it is reasonable to place an ignore test point here to eliminate this error.

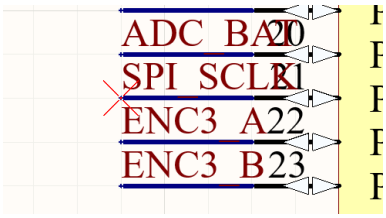


Figure 3.15: ERC test points neglected

The overall PCB schematic design is shown in the Figure 3.16.

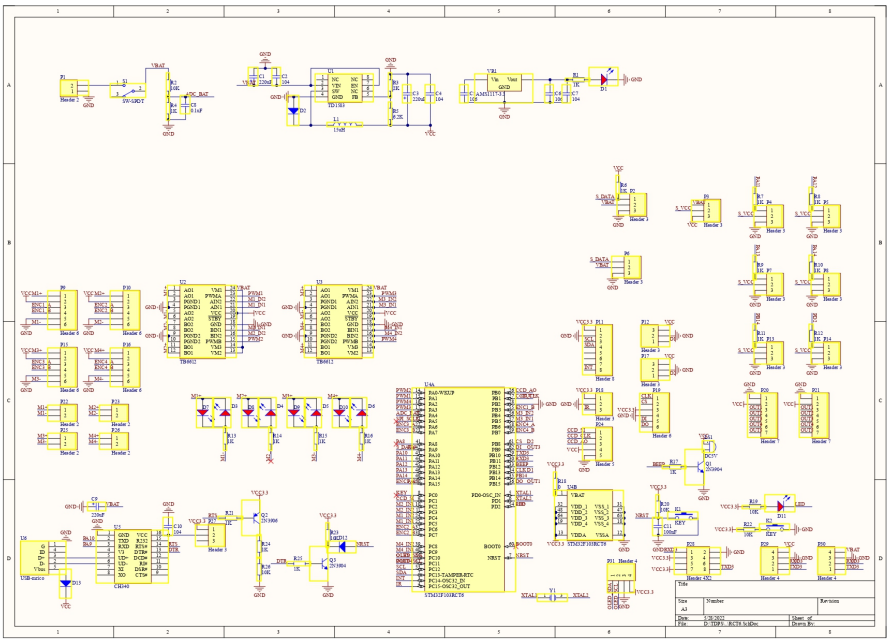


Figure 3.16: The resultant PCB schematic diagram

3.3.4 PCB layout diagram

The PCB layout will be drawn according to the schematic diagram designed in the above part, the package of the appropriate components will be selected supplementing the components that are not available, proper wiring rules will be formulated, and finally the drawing of the PCB is considered completed.

3.3.5 Wiring Choice

In Altium Designer, the usual methods are automatic routing and manual routing.

Manual Routing

Manual wiring requires actual wiring according to real situations, which means it has low wiring speed, requires more work time, and has higher learning costs, but as a trade-off, the flexibility is relatively high. It is more suitable for circuit drawings that need to consider signals and high-frequency interference [4].

Automatic Routing

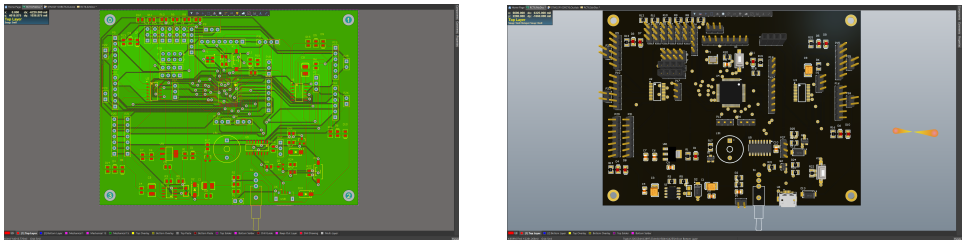
Automatic routing is based on existing rules, and there are original rules in AD, which need to be modified according to the requirements before automatic routing starts. The routing speed is much faster, but the flexibility is relatively low. For circuits that need to consider signals and high-frequency interference, automatic wiring cannot meet the corresponding requirements, and may even cause the circuit to not work properly [4].

At last,a combination of automatic routing and manual routing that is seen probably the most suitable and is the plan chosen in our PCB layout, because automatic routing can do most of the repetitive work, but for the parts that need to consider the signal and interference, manual routing can make the circuit behave better.

3.3.6 Implementing Procedure and Results

The outline procedure for PCB diagram implementation is as follows:

1. Design schematic diagram (completed in the previous section);
2. Define the component package;
3. PCB drawing settings;
4. Generate and load netlist;



(a) 2D PCB layout (b) 3D PCB layout

Figure 3.17: PCB layout in 2D and 3D

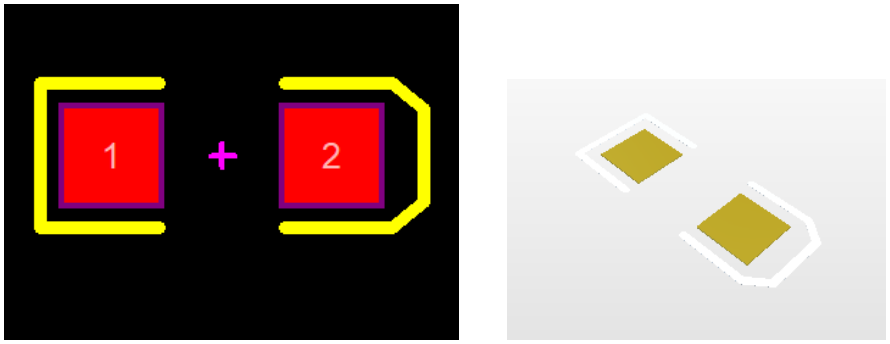
5. Component layout;
6. Set routing rule and route;
7. DRT verification.

Drawing settings: PCB board size and structure are designed first. Then, the number of board layers is designed to be two,as the it can reduce the difficulty of the manufacturing process, and the corresponding manufacturing fees will also be reduced, although it would increase the difficulty of wiring compared with multi-layer boards. However, considering that the number of circuit components connected and wire connections designed, a two-layer board is a good choice.

Generate loading netlist: The schematic is correlated to the PCB, during which the schematic package design mistake can also be checked.

Set routing rules: It is regulated that the conductor spacing should be greater than 4mil, which is considered the larger the better if conditions permit. As for wire width, a 10mil wire is used, which meets the electrical requirements. It should be guaranteed that the economic ground wire should be wide, being greater than 1mm to ensure that the connection potential remains unchanged. The line angle is 45 degrees, avoiding right and acute angles, which can reduce impedance and other problems at corners. Below are the 2D and 3D PCB layouts.

For some devices drawn by ourselves in the schematic diagram, the package definition of the corresponding components also needs to be defined in the PCB manufacturing process, while the existing packages can be defined or modified in the related packages. Figure3.34 shows the 2D and 3D schematic of one package component.



(a) 2D pin header-female header layout (b) 3D pin header-female header layout

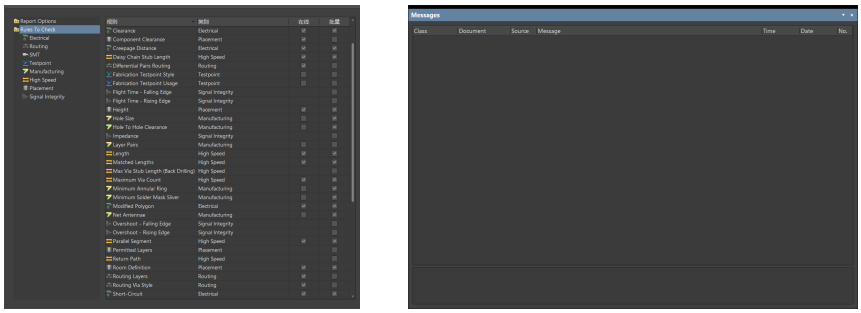
Figure 3.18: pin header-female header layout

DRT design rule check is to check the PCB according to the preset rules, including electric and routing rules and other preset rules check. In the end, the check was passed without violation to rules, and the warning should be adjusted or ignored according to the actual situation.

Results and Analysis

During the DRT verification process, there were many errors reported at the beginning, such as silk to solder mask clearance constraint, hole size constraint, minimum solder mask sliver constraint problems and other warning

errors. After conducting relevant research and consultation, it was found that the original DRT rules are unreasonably set, and thus they should be modified and verified according to actual situation(such as increasing the aperture, reducing the Clearance, adjusted an overlap), and finally zero error was achieved.



(a) DRT windows showing many errors

(b) DRT windows showing 0 error after modification

Figure 3.19: DRT error checking

Below is the resultant PCB layout.

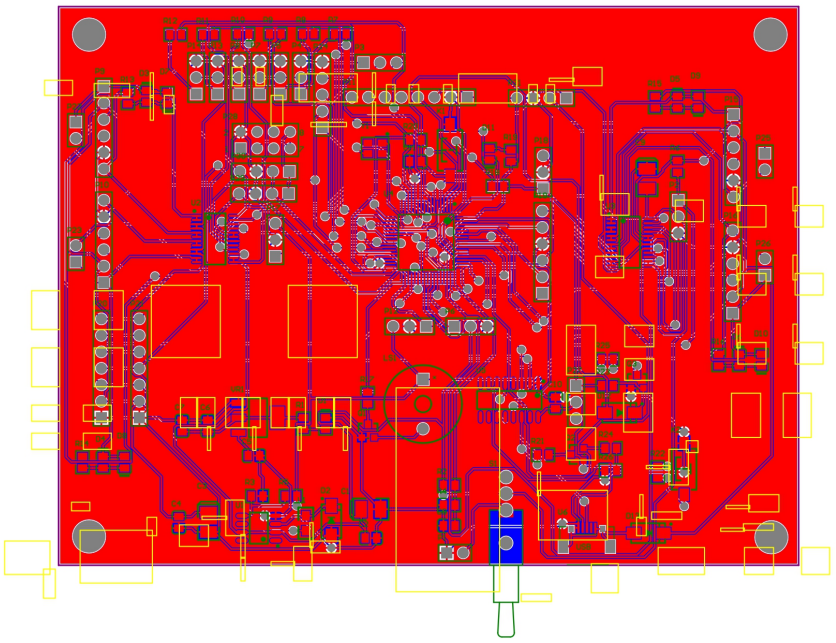


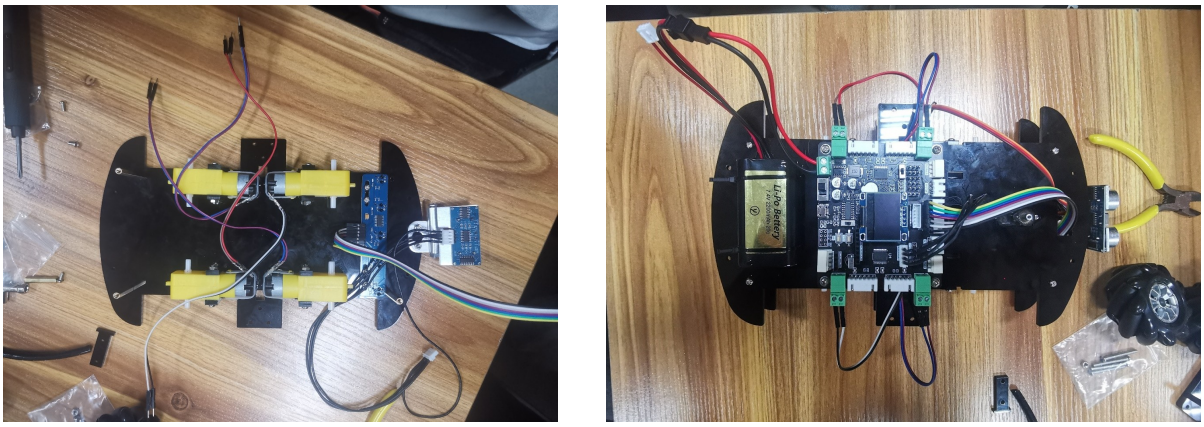
Figure 3.20: The resultant PCB layout

3.4 Mechanical Structure Design

Main Contributor: Shui Jiajun

3.4.1 Plan A

Initially, a car structure with two layers is designed, and the components assembled in this structure will be mentioned obeying an order from bottom to top. At the bottom layer, 4 Mecanum wheels are used, and the layer was spliced with them pushing a cylindrical hole in the middle of the wheel into the corresponding cylindrical head of the motor plastic stick. Moreover, four yellow motors were installed next to the four Mecanum wheels, which are fixed by using 2 screws each, as shown in Figure 3.21. At the right side of the plate, an infrared sensor was set stationary, which at last was not used in the car design. Seen from Figure 3.21, the ultrasonic sensor was also installed at the right side of this layer, and it was pulled in upward vertical direction to the plastic plate.



(a) First layer of the car

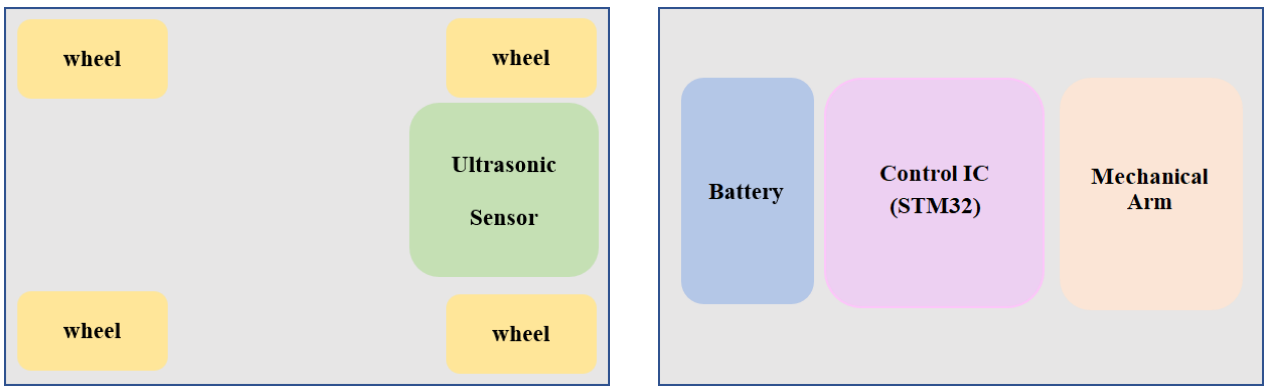
(b) Second layer of the car

Figure 3.21: Composition of the car

As for the second layer, it can be seen from the Figure 3.21 that in the middle of the second layer plastic plate, the main control module including the IC STM32F103, of which the OLED visualizing screen is placed on top. At the left end, the battery module is tied on it using two straps.

As for the wiring, all the components, including those at the first layer, are connected to the chip control board at the second layer using Dupont lines and wires. For wiring of modules at the first layer, small holes existing on the second layer are used as channels for wires to pass through.

The position of each module is shown in the Figure 3.22.



(a) First layer graph of the car

(b) Second layer graph of the car

Figure 3.22: Distribution of modules on the two layers

However, the two-layer design did not satisfy the height required by basket-shooting task after the second field test, thus a third-layer car was then designed, aiming to increase the height of the mechanical arm while the recognizing function of the OpenMV module would not be affected.

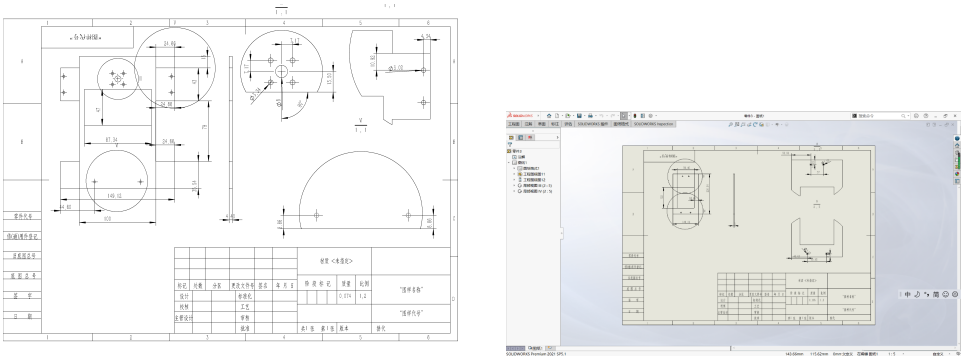
3.4.2 Plan B

SolidWorks is used to redesign the third floor of the trolley, while structure correction of the second floor of the trolley is carried out manually, and the height of the trolley is raised so that the robotic arm can meet the shooting requirements of 25cm high and the OpenMV image recognition is not disturbed.

Version 1: The robotic arm is placed on the second layer, the OpenMV is placed on the third layer, and a rectangular hole is left in the middle of the third-layer board for the debugging of the car control chip. The advantages of this version are: following the design of the second layer and the car are symmetrical and beautiful as a whole. In addition, OpenMV is at a geometric high point, so the identification will not be affected. Moreover, the third layer board is light and thin as a whole, and will not interfere with the overall center of gravity of the car, and has good stability. However, it has a serious shortcoming, that is, it is too thin and has poor toughness, which means its anti-vibration ability is weak. Even worse, it was actually damaged after a field test, and the robotic arm cannot meet the 25cm shooting requirement.

Version 2: Using the method of adjusting the position of the robotic arm and the OpenMV, SolidWorks was used to redesign and manufacture the thickened the third layer board and manually refit the second layer board. After field testing, it was found that the three-layer board could not meet the shooting requirements. Backward shifting and lengthening was needed for mechanical arm to avoid the situation where the center of gravity of the robotic arm is too high and the board might be crushed.

Figure 3.23 shows two versions of the three-layer board sketch design.



(a) Third layer: 1st version (b) Third layer: 2nd version

Figure 3.23: Two versions of the third layer

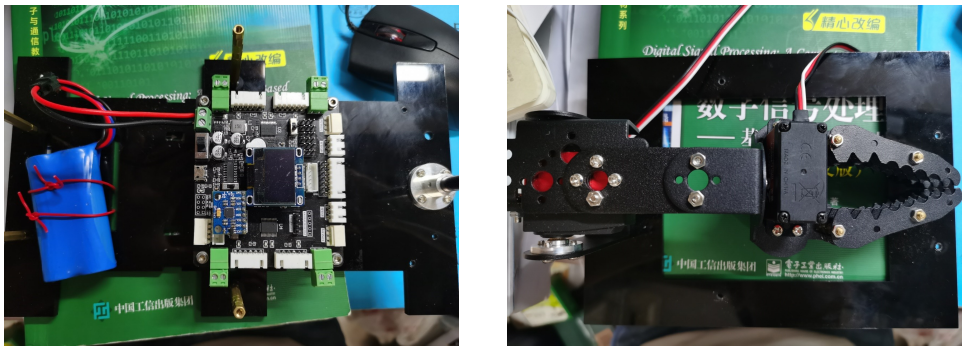
The structure of the first layer of the modified version remains unchanged, which is consistent with the Figure 3.21(a). The physical diagrams of the second and third layers are shown in Figure 3.24. The second layer starting from left to right has modules includes the battery, the main control MCU, the PCB board, and the OpenMV module which only partially enters the image. In the third part, in order to ensure that the function of the robotic arm can be realized, only a lengthened version of the robotic arm with an added in-line bracket was installed.

3.4.3 Plan C

After the third test, the car as a whole has changed a lot.

To begin with, the car wheels, the bottom plate are replaced, and a second-layer board is added, thus the car is more balanced, more stable, and has better off-road capability.

In terms of the height of the car, in order to make the overall height of the trolley higher, the Mecanum wheels with a diameter of 6cm are abandoned and a type of ordinary tires with a diameter of 8.5cm is used. To do this, an



(a) Composition of the 2nd layer (b) Composition of the 3rd layer

Figure 3.24: Modified 2nd and 3rd layer of the car

angle grinder is used to remodel the chassis to accommodate larger wheel diameter tires. And it was also found that Mecanum wheels were likely to cause the car to skid in raining days, thus this became another reason to replace Mecanum wheels with other wheels. At the same time, I used a longer copper column, which brought a height change from 8cm to 12cm, and greatly raised the second and third floors of the car. In addition, an in-line bracket is also added to the robotic arm to increase its height by 10cm. So far, the overall height of the trolley is high enough to complete the robotic arm pitching task.

At the same time, in order to solve the distance detecting problem, two ultrasonic elements are added on the right side of the car and they are controlled by OpenMV. In order to better solve the problem of crossing the bridge of the car, a gyroscope is added and the original TT DC motor is replaced by a Hall encoder with a high horsepower steel electric motor. The chassis of the previous version of the car is too light, and it is easy to roll over on the slope, thus an iron plate was added between the original first and second plate to lower center of gravity of the car and thus car stability could be increased.

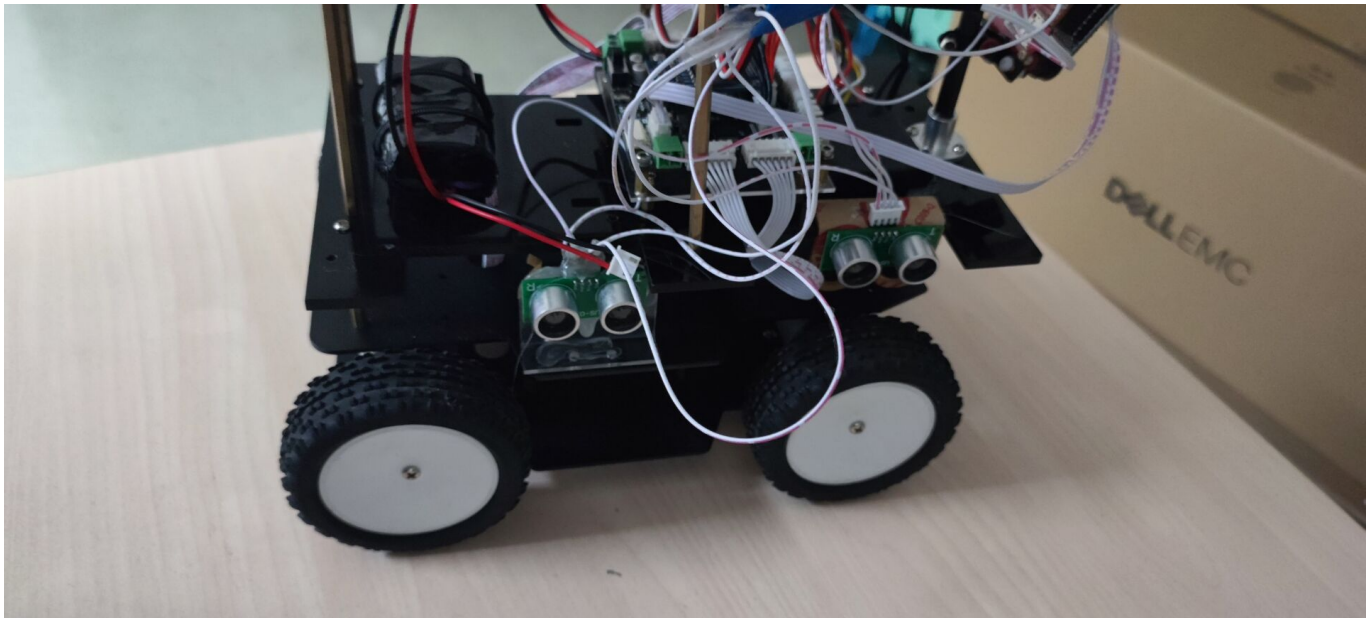


Figure 3.25: Two ultrasonic sensor position in the four-layer car

3.4.4 Results and Analysis

Eventually, a four-layer car is designed. The first layer connects the four wheels and contains the four Hall encoders, the second layer implements no peripherals but many holes to let wires go through it, the third layer has the PCB and the main control STM32F103RCT6 in the middle and the battery placed tightly at the end. As for the

top layer, it is more like a hollow bold rectangular, with the mechanical arm placed at the front side of it.

The battery is placed at the third layer end to balance the car as the mechanical arm with servo is placed at the front side. During the project, many versions of the car structure were made mainly to satisfy the height requirement of the basket and to balance the car. To do this,the car layer number is increased from two layers to four layers, and a lengthened type of cooper pillars is used to connect layers of cars. There are also other reasons why the plates are changed. For example, when it is found the the wheel type needed to be changed, a new bottom layer was designed to fit the new wheels. And when new peripherals are added, such as the gyroscope and the ultrasonic sensors, the peripheral position in different layers would be re-organized. As a result, it is believed in a team collaborative program, it is important to stay in touch with other members and react quickly towards any changes occurred.

3.5 Overview of Peripherals

Below are the detailed introduction of the peripherals we chose to realize functions of cars which help complete the required tasks, including the wiring, the function illustration, the pin description, the code and the code explanation of each peripheral. The peripherals included in this section are ultrasonic sensor, the mechanical arm, and the OLED screen. As for the driver module, it will be introduced in section 2.4.

3.5.1 Basic Components for Car Mode Selection and Monitor

Main Contributor: Liu Zhe, Li Muquan

LED (Li Muquan)

8 LEDs are used to show the direction of the motor. The two LEDs correspond to a motor, showing the direction of the motor's current flow.

LED Code Explanation:

1. First, the GPIO port clock is enabled;
2. Then, initialize the GPIO target pin to push-pull output mode and configure the corresponding speed;
3. Control GPIO pin to output high and low levels.

Generally, hardware-related macro definitions are encapsulated by #define. If the GPIO is changed, it is only required to modify the macros.

The specific code is as follows.

```
#include "led.h"
#include "delay.h"

//LED IO initialization
void LED_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD,ENABLE); //Enable PORTB Clock

    GPIO_InitStructure.GPIO_Pin  = GPIO_Pin_2; //PD2
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;  //set the mode be push-pull output
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
```

Beef Buzzer (Liu Zhe)

The buzzer chosen resembles the one showing in Figure 3.26, and it is used for mode selection and mode hint. For different modes selected, the buzzer will beep differently and help us know the mode the car is in currently.

Figure 3.26: Buzzer outlook

Beep Code Explanation: First, the buzzer pin PB12 is set the push-pull output mode and this pin is then intialized. Afterwards, one function called N_Beep is used to show make the beep buzzer beep N times repeatedly, in which N is a number set by the user.

```

void LED_Init(void)
{
#include "beep.h"
#include "delay.h"

void Beep_Init(void)
{

    GPIO_InitTypeDef GPIO_InitStructure;
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
    GPIO_InitStructure.GPIO_Pin  = GPIO_Pin_12; //PB12
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;  //set mode be push-pull output
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOB, &GPIO_InitStructure); //initialize GPIOB12
    GPIO_ResetBits(GPIOB,GPIO_Pin_12);

}

void N_Beep(char n)
{
    char i=0;
    for(i=0;i<n;i++)
    {
        BEEP(1);
      Delay_ms(150);
        BEEP(0);
      Delay_ms(150);
    }
}
```

Analogue to Digital Converter (Liu Zhe)

ADC measures battery voltage through voltage divider resistors.

The ADC in the micro-controller is the abbreviation of Analog-to-Digital Converter, which refers to an analog-to-digital converter or an analog-to-digital converter.

ADC refers to a device that converts a continuously changing analog signal into a discrete digital signal. Real-world analog signals, such as temperature, pressure, sound, or images, need to be converted into digital forms that are easier to store, process, and transmit. An analog-to-digital converter can achieve this function, and it can be

found in a variety of different products.

The ADC used is 12-bit.In the specific conversion process, the analog range of the 12-bit ADC is 2^{12}, i.e. 4096, which means that the range of the chosen 12-bit ADC analog is 0∼4096.

ADC Code Explanation: First, the ADC is initialized by initializing the IO port and enabling the port clock, and it is the regular channel that is used here. Comparatively, there exists a type channel called injected channels, which has higher processing superiority over the regular ones. The triggering source is the well-set register CR, and the external event selected for starting regular group to convert is set to be SWSTART in CR2, which means for each converesion done, an '1' is written to this bit. Consequently, the triggering mode is set by further defined the register CR2. Then, by setting the register CR1 of ADC1, the working mode is set being independent and non-scanning. By setting the register CR2 of ADC1, the first bit ADON bit is set as 0, which means the ADC will be awaken from off mode by writing '1' into it.

In addition, a sampling event register called SMPR2 is used to decide the sampling time interval from the 0 to 17 channels. Specifically, as the ADC uses multiple channels to convert, the registers SQRX is are used to decide the order of data conversion. A register called ADC_DR is used for converted data storage, in which the data transferred through the regular group is stored in up to 15 bits, and these bits are for read only in order.

At last, the sequence is converted and the battery value is calculated. The code is shown below with detailed code notes.

```
void  Adc_Init(void)
{
    //initialize GPIOA4 port and the clock
    //omitted here

    RCC->CFGR|=2<<14;
    ADC1->CR1&=0XF0FFFF;    //work mode reset
    ADC1->CR1|=0<<16;       //Independent working mode
    ADC1->CR1&=~(1<<8);     //non-scanning mode
    ADC1->CR2&=~(1<<1);     //single conversion mode
    ADC1->CR2&=~(7<<17);
    ADC1->CR2|=7<<17;       //Software Controlled Conversion
    ADC1->CR2|=1<<20;       //Use external trigger (SWSTART)!!! Must use an event to trigger
    ADC1->CR2&=~(1<<11);    // Align right
    ADC1->SQR1&=~(0XF<<20);
    ADC1->SQR1|=0<<20;
    //Set the sampling time of channel 1
    ADC1->SMPR2&=~(3*1);    //Channel 1 sampling time clear
    ADC1->SMPR2|=7<<(3*1);
    ADC1->CR2|=1<<0;        //Turn on AD converter
    ADC1->CR2|=1<<3;        //Enable reset calibration
    while(ADC1->CR2&1<<3);  //Wait for the calibration to end
    //This bit is set by software and cleared by hardware. This bit will be cleared
    after the calibration register is initialized.
    ADC1->CR2|=1<<2;        //Enable AD calibration
    while(ADC1->CR2&1<<2);  //Wait for the calibration to end
    // This bit is set by software to start calibration and cleared by hardware when calibration ends
}
u16 Get_Adc(u8 ch)
int Get_Battery_vot(void)
```

Key (Li Muquan)

Key is the GPIO button initialization, and it is used to select mode, in which long press to select and short press to confirm.

Key Code Explanation: A key initialization function is used to initialize the key, including the port used and the mode selected.

```
#include "key.h"
#include "delay.h"

//key initialization function
void KEY_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC,ENABLE);
    GPIO_InitStructure.GPIO_Pin  = GPIO_Pin_0;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
    GPIO_Init(GPIOC, &GPIO_InitStructure);
}
```

Results and Analysis

The key, beep buzzer and the ADC converter are added in the test field process, as during real testing, much inconvenience has occurred and these elements are added to solve the inconvenience. The key helps to select the mode, which means it acts like a selecting button to help choose whether the car is entering patio 1 or 2, while the beep buzzer is a simple implementation that indicates whether we are selecting mode or just confirming the choice made previously. As for the ADC converter, it gets the remaining voltage percentage and this value is shown through the OLED screen thus I can decide when to charge the car, which would fully utilize the battery and extend its life span.

3.5.2 Ultrasonic Module

Main Contributor: Lin Meihong

In order to realize the ultrasonic ranging function, the HC-SR04 ultrasonic sensor is used. The ranging function is realized by emitting a unique ultrasonic wave which would distinguish itself from ambient noise, and by recording the time it takes to receive the reflected wave.

On the STM32F103RCT6 development board, the ultrasonic module is connected to P13, and the P13 of the board directs to the corresponding interface on the STM32F103 chip, which are PC1 for Trigger, PB0 for Echo and PB2 for Ground, while the 5V is connected to the output of the voltage regulator circuit TD1583, the 3.3V is connected to the output of the voltage regulator circuit AMS1117-3.3 [1].

Module Function Description: The heart of the HC-SR04 is two ultrasonic sensors. One acts as a transmitter, converting the electrical signal into 40 KHz ultrasonic pulses, and the other acts as a receiver, receiving the reflected transmitted pulses back [5]. If received, it will generate an output pulse whose width is proportional to the distance the pulse travels, thus measuring the distance from the object to the module. The HC-SR04 ultrasonic sensor has the advantage of being small in size, easy to use in any robotics project, and providing excellent non-contact range detection between 2 cm and 400 cm with an accuracy of 3mm. Since it operates at 5V, it can be connected directly to any 5V logic micro-controller. The control board used can directly output a voltage of 5V, so the HC-SR04 could be directly connected to the development board.

Pin Function Description:

- VCC: Power pin, and it is connected to 5V pin on the stm32 development board;
- Trig: To trigger ultrasonic pulses;

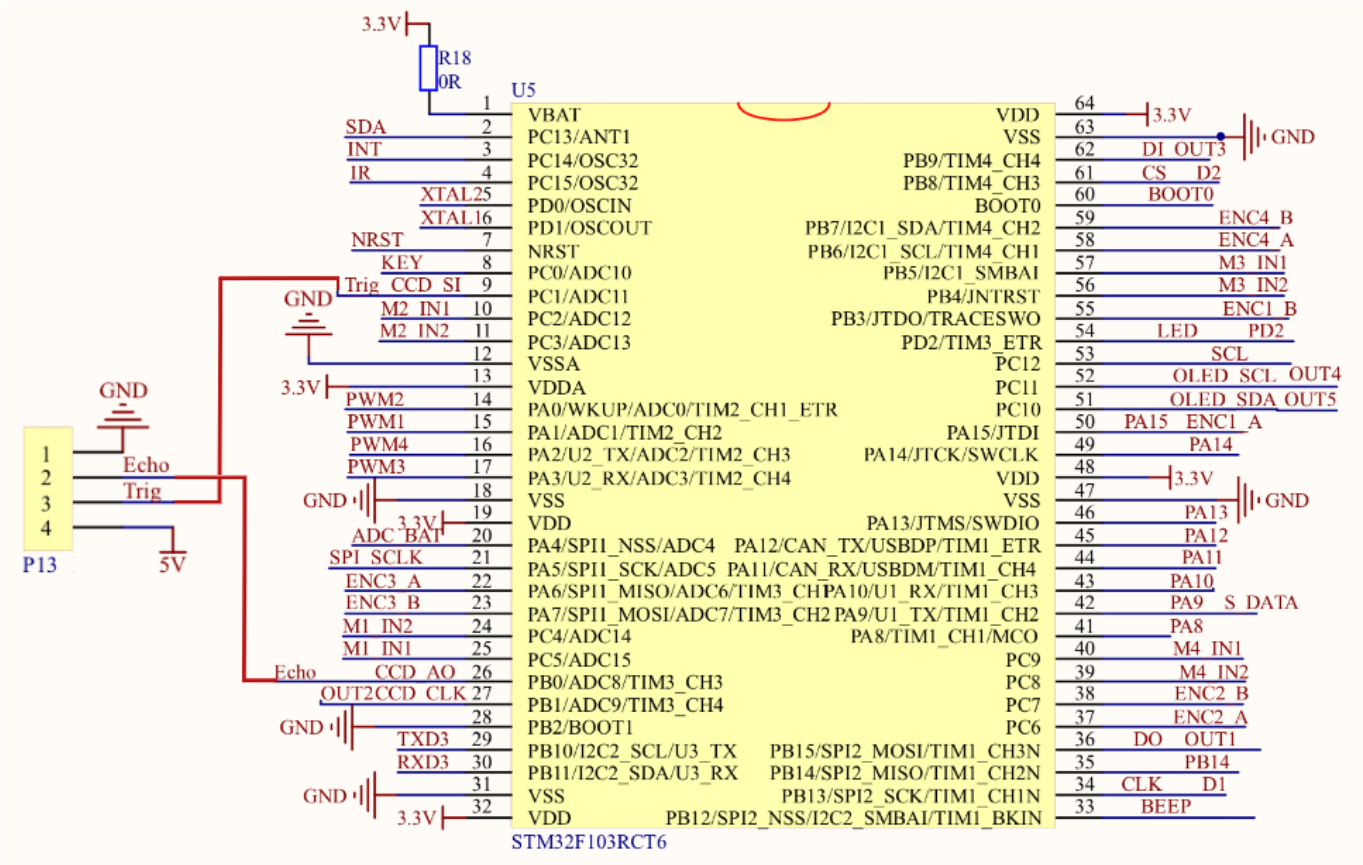


Figure 3.27: HC-SR04 board-IC wiring connection

- Echo: When receiving a reflected signal, it generates a pulse, the length of which is proportional to the time spent to detect the emitted signal;
- GND: connected to the ground pin on the board.

Code Realization: Uln_Trig.c function.

1. Use the pin on board to send the Trig pin with a high level signal lasting at least 10s to trigger the ranging function of the SR04 module;
2. After triggering, the module will automatically send a distinct 8 ultrasonic pulses of 40KHz, and automatically detect whether there is a signal reflected. This step is done automatically inside the module. Sending 8 pulses makes the emitted acoustic signal unique and thus distinguishable from ambient ultrasonic noise;
3. If receiving reflected signals, the Echo pin will output a high voltage level signal and the duration of the high level is the time from the launch to the return of the ultrasonic wave. At this point, the EXTI0IRQ_Handler function is used to obtain the ranging result and calculate the actual distance of the measured object. If the high level output time of Echo exceeds 38ms, it means that there is no obstacle within the range, and the Echo pin returns to low voltage level.

Code Explanation: First, enable the GPIOC clock and the PC1 port, and set the PC1 port as a general push-pull output mode, with a maximum output speed being 50MHz. Then, the register of the timer 7 is enabled, and the time unit is set to 10us by the prescaler psc. Third, a variable named Distance is defined, which points to the value of the timer 7 register CNT.

The following two functions in the code correspond to two situations: if Distance>500, meaning that the object distance exceeds 500cm, the detection range is exceeded,and the register CNT is cleared and starts to count again; if Distance<500, meaning that the distance of the nearest object is less than 500cm, the distance is calculated, the

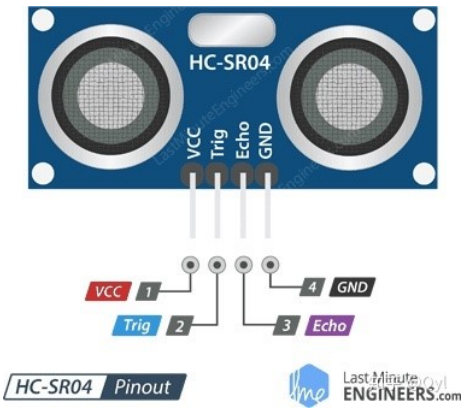


Figure 3.28: Pin of HC-SR04

interrupt occurs, and then the interrupt bit is cleared and the count starts again. The calculated distance is eventually reduced by 7, which is determined according to the installation position of the HC-SR04 on the car. Uln_Trig code is as follows.

```
void Uln_init(void)

void Uln_Trig(void)
{
    Trig=1;Delay_us(100);Trig=0; //Trig pin generates 10us high level
    if(TimeOut_Flag1)Distance1=500;
    TimeOut_Flag1=1; //out of range
    TIM7->CNT=0;                 //Set the automatic reload value of the counter;
    //CNT is the counter register, and the value is set to 0
    TIM7->CR1|=0x01;             //enable counter CNT
}

void EXTI0_IRQHandler(void) //Get the ranging result in the interrupt function

#ifndef __ULN_H
#define __ULN_H
#include "sys.h"
#define Trig PCout(1)
extern unsigned int Distance1;

void Uln_init(void); //External interrupt initialization
void Uln_Trig(void);
#endif
```

Results and Analysis

The ultrasonic sensing controlled through STM32 is used both in patio 1 and 2 and it could fulfill its tasks successfully. In patio 1, the sensor should be used to detect the box before turning right to cross the bridge, and it is noted the car should make physical contact to the box thus the distance calculated by the HC-SR04 should be adjusted. Then in patio 2, the ultrasonic sensor is used to detect the two left turns in the railing area. Finally, it is responsible to detect the basket where the ball will be shot. During the test field, due to the installing position of the ultrasonic sensor on the car, the distance calculated by it should be adjusted according to its position to acquire more

precise data. After many tests, the ultrasonic sensor could function normally and fulfill its tasks.

3.5.3 Self-made Mechanical Arm and Servo

Main Contributor: Feng Guoyu
Servo selection: TBS-K20

Figure 3.29: TBS-K20 outlook

Servo is used with the mechanical arm to control the angle of the arm using PID algorithm. Initially, MG90S servo was used for its wide availability in the market and easy installation. However, during the first trail, it was found that the torque generated by it is 1.8kg/cm(4.8V) [6], which was insufficient to lift a tennis ball. Then, TBS-K20 servo is chosen instead to generate a larger torque, which is 15.5kg/cm(6V) . One of its advantages is that the operating voltage range is 5-8.4V, thus if larger torque is required, TBS-K20 can be connected directly to the 7.4V output on the board, achieving a high sustainability with the car system.

TBS-K20 robot steering gear is a servo unit developed by Zhongling Technology Co., Ltd. that integrates motor, servo drive, and pwm signal interface. It is mainly used for joint drive of robots and mechanical arms, and can also be used for other precise positioning control occasions. The features of TBS-K20 are shown in the table below [7]:

Items	Specifications
Operating Voltage	5-8.4V
Appearance size	40*20*40.5(mm)
No-load speed	0.16 sec/60(6V);0.14 sec/60(6.6V)
Stop torque	15.5kg/cm(6V); 20kg/cm(6.6V)
Stop Current	1.8A(6V); 2A(6.6V)
Gear Material	Metal Gear
Mechanical limit angle	360
Weight	60 ± 1g
Control System	PWM
Operating frequency	50-330Hz
Operating angle	270
Rotation direction	counterclockwise

Table 6: TBS-K20 basic parameters

The main parameters are as follows:

1. Type: PWM single axis;
2. Power supply: 5-8.4V;
3. Angle: 270;
4. Torque: 20kg*cm.

Among them, TBS-K20 does not support voltage feedback, temperature feedback, angle feedback, nor anti-blocking [7].

Mechanical Arm Design - Scheme Selection

Due to the increased shooting height, it is necessary to adjust the three-layer trolley structure, increase the height of the robotic arm, and replace the bearings. In this scheme, the middle of the manipulator is dismantled and an inline bracket is added to extend the length of the manipulator. The advantages of this scheme are as follows:

1. Increased length of the robotic arm, more complex programming and manipulation possibilities;
2. OpenMV can be at a low point, the robotic arm will not affect the sight of OpenMV;
3. The robotic arm can be moved backward, the center of gravity is closer to the center of the car, and the stability is enhanced.

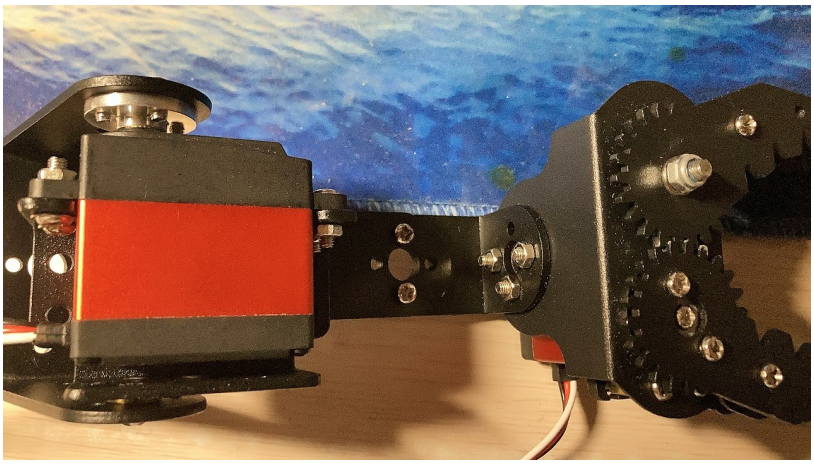


Figure 3.30: Lengthened mechanical arm in Plan 1

This scheme has a significant disadvantage, that is, the growth of the mechanical arm causes the center of the original mechanical pliers to move forward, and the overall in-line bracket has not fully arrived. It is possible to add one in-line bracket, but it may not be feasible to increase several.

Fortunately, the car is raised overall by using a larger wheel and higher copper pillars between layers of the car, thus the required height of the mechanical arm is not that much. One increased in-line bracket is sufficient to complete the basket shooting mission.

Angle Rotation Theory

The steering gear can rotate at different angles through different program designs. Unlike ordinary motors, it can only rotate at a certain speed when powered on. The function of the steering gear is not to rotate in a loop, but to rotate at a certain angle by controlling the PWM duty cycle to represent such angle. The angle of rotation, thereby driving the external mechanical structure. The Figure 3.31 is such an example, both waves having a period of 20ms, but they rotate at different angles (0∼180) because of the different time they are at the high level and the Figure 3.32 shows four different rotation modes. When the STM32 pin inputs a high level of 0.5ms and a low level of 19.5ms, the servo can turn to the 0ř position. Correspondingly, when a high level of 2.5ms and a low level of 17.5ms are input, the servo will turn to the 180ř position.

First, the servo has three wires, two power wires and one signal wire. When we control the steering gear, we need to give the steering gear a suitable working voltage. Usually 5V power supply is applied, but when the number of steering gears is large, it can be powered by 2s lithium battery. Secondly, to drive the steering gear to rotate, a suitable working frequency is required. When controlling the steering gear, a pulse wave with a frequency of 50Hz to control is required, that is, a pulse wave with a period of 20ms (T=1/F). When the period is 0.5ms, the corresponding servo is 0 degrees, and when the period is 2.5ms, the corresponding servo is 180 degrees, that is, the duty ratio is 0.5/20=0.025 to 2.5/20=0.125 The corresponding servo rotation angle 0 degrees to 180 degrees.

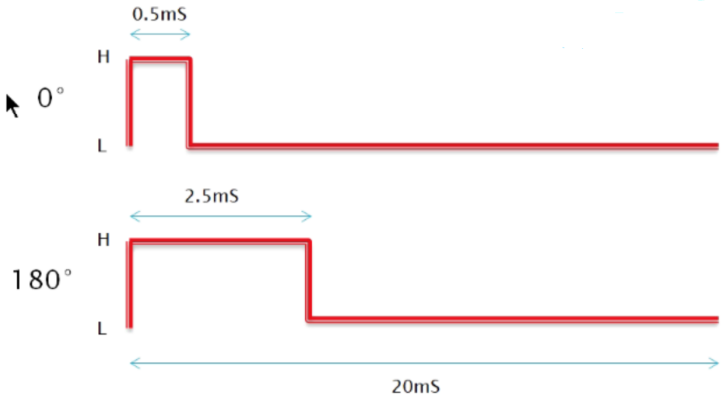


Figure 3.31: Angle and PWM width relation

Time	Angle
0.5ms	0
1ms	45
1.5ms	90
2.0ms	135
2.5ms	180

Table 7: Time and corresponding angle

The angles correspond as follows:

Theoretically, the working mode is to continuously send out the angle waveform to the PWM line until the corresponding angle and under normal circumstances, the maximum angle of the 180-degree steering gear can reach 190 degrees or even 200 degrees. Therefore, it is very important to debug according to the relationship between the waveform and angle of the specified type of servo. Fortunately, the servo has normative error data. A code can be made on the code side to prevent the rotation angle is too large or too small.

The code is therefore designed into three logic part. One is GPIO Level inversion function which enables the STM32 timer to control the operation of the 8-way steering gear and invert IO levels. A definition that clears the PWM cycle level and rotation mode, which is initial the PWM pulse width position of steering gear, represents 1.5ms, 1ms,1.5ms,1.5ms,1.5ms,1.5ms, where a protection function is applied to prevent servo rotation time form deviating from normal expectations, too large or too small. The third parts is composed of comparison of pulse width change and speed control, which is a conditional function that defines whether the steering gears works. The servo is moving at the specified speed to the position corresponding to the specified pulse width. Calculation only occurs when motion is stopped, and the pulse width changes during every 20ms function time. A loop is set to make sure that the practical rotation value is equal to expectation. Then when reaching the set position, the servo stops moving.

The experiment procedure was to put the output under the oscilloscope to see the practical PWM to check the correctness of the waveform output. After confirming the correctness, the trolley is used for field testing.

Below is the mechanical arm core control code.

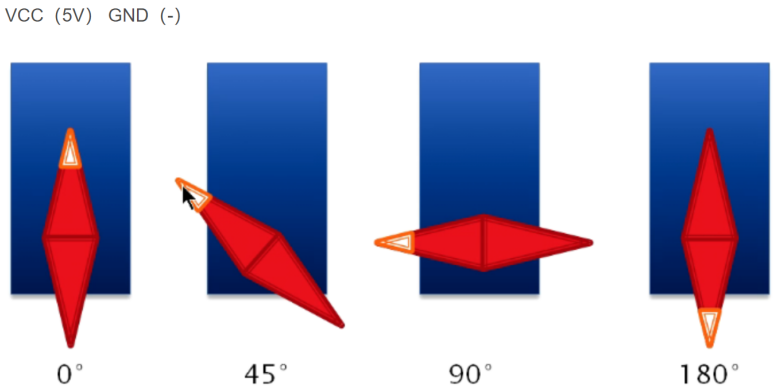


Figure 3.32: Angle and its position

```
1
2  #define MAXPWM 2505        // Macro-definition of maximum PWM width
3  u8 count1;
4  static bool Running = FALSE;       // initial flag of steering gear operation
5  uint16 ServoPwmDuty[8] = {1500,1000,1500,1500,1500,1500,1500,1500};
6       // PWM width of steering gear
7  uint16 ServoPwmDutySet[8] = {1500,1000,1500,1500,1500,1550,1500,1500};
8       // PWM width of steering gear
9  float  ServoPwmDutyInc[8];          //PWM pulse width increases every 2.5ms or 20(2.5*8) ms accordingly
10 bool ServoPwmDutyHaveChange = TRUE;         //Flag bit with pulse width change
11 uint16 ServoTime = 2000; //The time for the servo
12
13 void ServoSetPluseAndTime(unsigned int id,unsigned int p,unsigned int time)
14      //preventing rotation time deviation
15 void ServoPwmDutyCompare(void)  //Comparison of pulse width change and speed control
16
17 bool ServoAction(void)     // Determine if the steering gear is working
18      //GPIO inversion function1
19 void Flip_GPIO_One(void)
20      //servor control function
21 void Servo1(void)
```

3.5.4 OLED Screen

Main Contributor: Lin Meihong

OLED Introduction

OLED, namely Organic Light-Emitting Diode, also known as Organic Electroluminesence Display. As OLED has self-illumination, no need for being against light, high contrast, thin thickness, wide viewing angle, fast response speed, wide operating temperature range, simple structure and process at the same time, it is considered to be the next-generation emerging application technology of flat-panel display [8].

In contrast, LCD requires being against light, while OLED does not, because it is self-illuminating. In this way, the same display, OLED has better effects. The size of OLED is difficult to enlarge, but the resolution can be very high.

An OLED device is chosen with a 0.96-inch I2C interface. Its access voltage range is 3.3∼5V, and it has a built-in

Figure 3.33: OLED Outlook

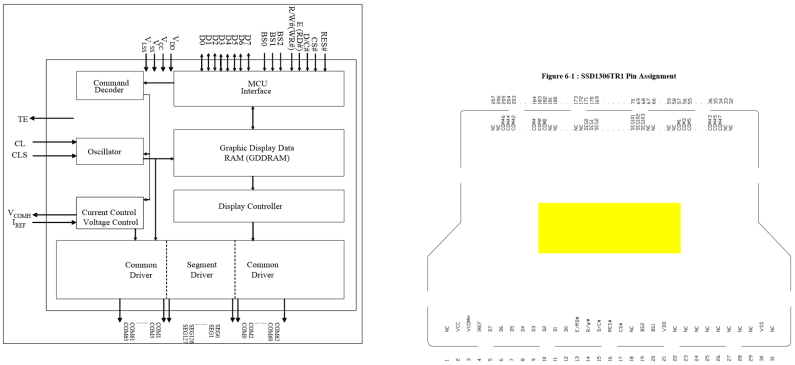
3.3V integrated circuit regulator LDO circuit, because the level of the I2C communication interface requires 3.3V. The chosen OLED has a pixel lattice scale of 128 columns x 64 rows, and the integrated driver IC model is SSD1306 [8].

The pin description is as follows.

Pin name	Function
GND	Ground
VCC	OLED module input voltage, range:3.3V-5V
SCL	I2C module bus clock signal
SDA	I2C module bus data signal

Table 8: OLED pin description

SSD1306 Drive Introduction



(a) SSD1306 inner structure (b) SSD1306 pin position

Figure 3.34: SSD1306 inner and pin structure

SSD1306 is a monolithic CMOS OLED/PLED drive chip consisting of 128 segments and 64 Commons. Its function is to drive the organic/polymeric light emitting diode dot matrix graphic display system. The chip is designed especially for common cathode OLED panels. The resolution is 128*64 (so many pixels), of which 64 lines are divided into 8 pages with 8 lines per page. SSD1306 has been integrated in OLED. It supports multiple bus driving modes, including SPI and parallel ports. The corresponding IO port of the chip is pulled low or high to decide which port is chosen. In the car, the corresponding IO port configuration is solidified and used through resistors, thus I2C interface mode is used.

Below are some common commands used in SSD1306.

1. 0X81: Set the contrast. Contains two bytes, the first 0X81 is the command, and the next byte is to set the contrast, the larger the value, the brighter the screen;

2. 0XAE/0XAF：0XAE is the command to turn off the display, and 0XAF is to turn on;

3. 0X8D: Contains two bytes, the first is the command word, the second is to set value, the BIT2 of the second byte represents the switch state of the charge pump. When the bit A2 is 1 (0x14), the charge pump is turned on , while the charge pump is turned off when A2 is 0 (0x10). When the module is initialized, this must be turned on, otherwise the screen display cannot be seen;

4. 0XB0∼B7: Used to set the page address, the value of the lower three bits corresponds to the GRAM page address;

5. 0X00∼0X0F: Used to set the lower four bits of the starting column address during display;

6. 0X10∼0X1F: Used to set the upper four bits of the starting column address during display.

I2C Communication Protocal

The I2C communication protocol is chosen to establish half-duplex communication between STM32 and SSD1306.

SlaveAdress: The I2C slave address of the module is: 0111 10 [SA0] [RW], SA0 is the hardware address selection bit, when SA0 is connected to high level, SA0 in the address is 1, and when SA0 is connected to low level, SA0 in the address is 0. When the module leaves the factory, SA0 is generally connected to a low level. Users can modify the resistance behind the module to change SA0 to a high level. This also means that the same I2C bus supports up to 2 OLED modules.

Start Condition: During the period when the clock line SCL remains at a high level, the level on the data line SDA is pulled low, that is, a negative transition, which is defined as the start signal of the I2C bus, which marks the beginning of a data transfer. The start signal is a level transition timing signal, not a level signal. The start signal is actively established by the STM32 master, and the I2C bus must be in an idle state before the signal is established. After a Start condition, the bus will be busy until an End condition occurs.

End Condition: During the period when the clock line SCL remains high, the data line SDA is released, so that SDA returns to a high level (that is, a positive transition), which is called the stop signal of the I2C bus, which marks the termination of a data transfer. The stop signal is also a level transition timing signal, not a level signal. The stop signal is also actively established by the master. After the signal is established, the I2C bus will return to the idle state.

Below is a timing diagram of the I2C communication protocol.

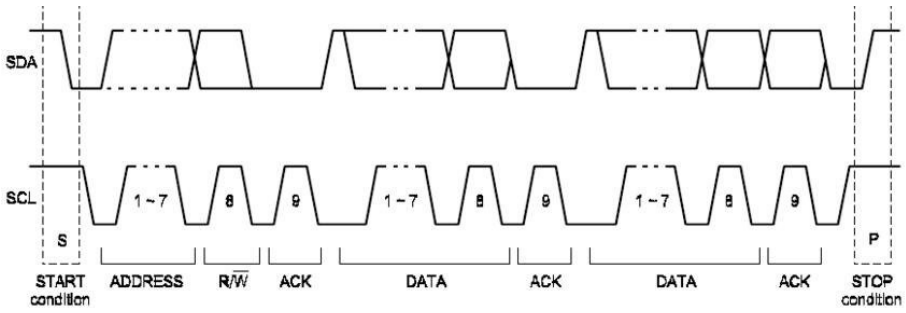


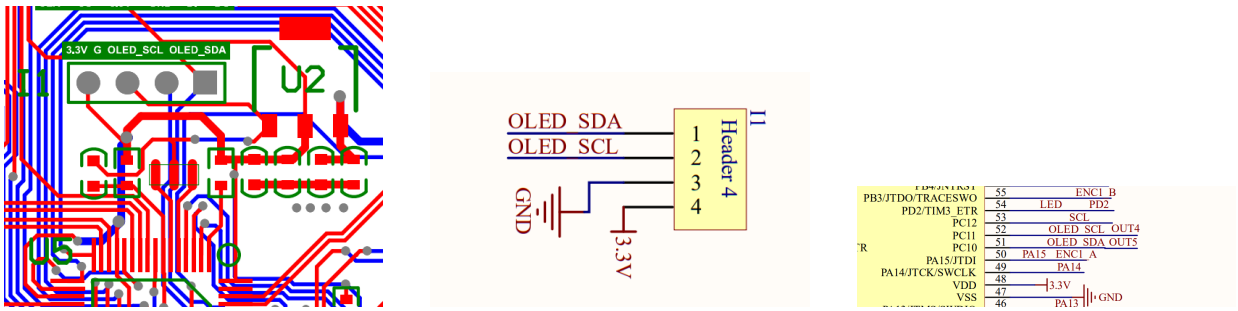
Figure 3.35: I2C timing diagram in SSD1306

PCB Connection

The OLED module is located above the STM32 module in the PCB schematic diagram; as for the connection with the STM32F103RCT6, a 2$\hat{2}$ jumper soldering board of I1 is used to wire the OLED module to the STM32, VCC to 3.3V, GND to ground, SDAout to PC_10 , and SCL to PC_11.

Code Explanation

OLED.c Code: OLED.c contains many functions that OLED needs to use. It implements the communication protocol used by OLED and the functions required to complete the task of the car through code. Among them, first, the OLED_Init function initializes the configuration of the OLED, including setting the brightness, mode, etc. of the OLED screen; second, the OLED_WriteData and OLED_WriteCmd functions describe the OLED as a slave to write



(a) OLED Position on PCB (b) OLED wiring to I2 (c) OLED wiring to STM32

Figure 3.36: On-board connection of OLED

the functions of data and commands to the master, i.e. the main control board. The above three functions initialize OELD and visualize the read and write functions of OLED.

The next three functions simulate the I2C communication protocol adopted by the OLED driver SSD1306 and realize the communication with the OLED. First, the OLED_set_Position function specifies the starting position of the writing displayed on the OLED screen; secondly, the OLED_I2C_Start and OLED_I2C_Stop functions describe the conditions for the start and end of the I2C protocol, namely the corresponding performance of the SDA pin and the SCL pin; fifth, the OLED_I2C_Wait_Ack function visualizes the MCU as the device receiving data to send a low level signal to the data sending device OLED at regular intervals to indicate that the data has been received. function; sixth, the OLED_I2C_Send_Byte function describes how the OLED sends 1-byte data to the master; finally, the OLED_Fill function displays the data on each page (eight pages in total) Specifically, it tells the master what kind of data to write from in which position on which page, and finally displays it on the OLED display.

The code is demonstrated as follows with detailed annotation.

```
#include "oled.h"

void OLED_Init(void)

void OLED_WriteDat(unsigned char dat) //write data

void OLED_WriteCmd(unsigned char cmd) //write command

void OLED_set_Pos(unsigned char x,unsigned char y) //set initial writing position

void OLED_I2C_Start(void) //I2C port drive code

void OLED_I2C_Stop(void) //stop communication

u8 OLED_I2C_Wait_Ack(void) //ACK waiting

void OLED_I2C_Send_Byte(u8 txd)

void OLED_Fill(unsigned char bmp_dat) //show full screen
```

OLED.User.c Code: In the OLED.User.C file, it mainly contains the rules for writing strings of different lengths and formats, such as the color and coordinates when writing 12x24 characters, 7x14 characters, etc. into the main control RAM. In addition, the variable OLED_Buffer[8][128] is defined at the beginning of the file, which means the OLED display buffer. If only the writing rules of strings are defined, the strings are only written into the RAM of the main controller, and cannot appear on the OLED display screen. Therefore, the functions OLED_Refresh_GDRAM

and OLED_Refresh_AllGDRAM are defined at the beginning of the file, so that the data into the main control RAM is written into the OLED and displayed.

Below is the code of OLED.User.c.

```
#define X_WIDTH 128
#define Y_WIDTH 64

//IO definition
//if input, PC_1O Pull-up input;PC_10 Push-Pull with 50MHz speed
#define OLED_SDA_IN() {GPIOC->CRH&=0XFFFFF0FF;GPIOC->CRH|=0X00000800;}
#define OLED_SDA_OUT() {GPIOC->CRH&=0XFFFFF0FF;GPIOC->CRH|=0X00000300;}

#define OLED_I2C_SCL PCout(11)
#define OLED_I2C_SDA PCout(10)

#define READ_OLED_SDA PCin(10)

void OLED_I2C_Init(void);
void OLED_I2C_Start(void);
void OLED_I2C_Stop(void);
u8 OLED_I2C_Wait_Ack(void);
void OLED_I2C_Send_Byte(u8 txd);

void OLED_Init(void);
void OLED_WriteDat(unsigned char dat);
void OLED_WriteCmd(unsigned char cmd);
void OLED_Set_Pos(unsigned char x,unsigned char y);
void OLED_Fill(unsigned char bmp_dat);

#endif
```

Results and Analysis:

The OLED screen could be used to help select mode at the beginning of every patio, and then it could be used to present the devices parameters, such as the motor speed, the ultrasonic sensing distance and so on. More importantly, when switching between the patio 1 and 2, there is no need for the car to load the corresponding code into the main control, instead, we can just select the mode that is going to use shown on the OLED.

3.5.5 Gyroscope

Main Contributor: Liu Zhe

Introduction

MPU6050 is the world's first integrated 6-axis motion processing component launched by InvenSense, with a 3-axis gyroscope and a 3-axis acceleration sensor, and a second IIC interface, which can be used to connect an external magnetic sensor [9]. Its own Digital Motion Processor(DMP) hardware can be used to accelerate the engine, and output complete 9-axis attitude fusion calculation data to the application side through the main IIC interface. With DMP, the motion processing database can be used which is provided by InvenSense, which is very convenient to achieve attitude calculation, reduce the load of motion processing operations on to the operating system, and increase the stability of the car movement.

The MPU6050 can simultaneously detect motion data of three-axis acceleration, three-axis gyroscope (three-axis angular velocity), and temperature data. Using its internal DMP module (Digital Motion Processor), the sensor

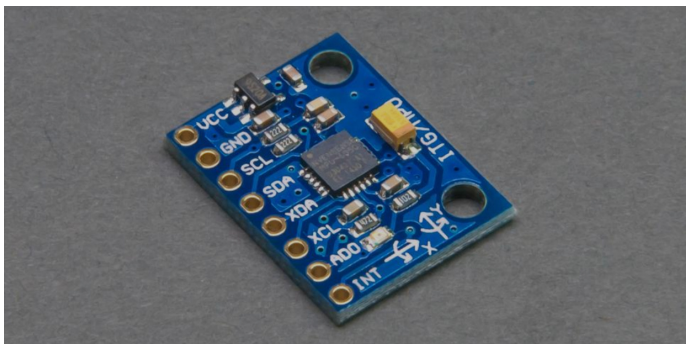


Figure 3.37: MPU6050 outlook

data can be filtered and fused, and the attitude-calculated data can be output to the main controller directly through the IIC interface, reducing the computational workload of the main controller. Its attitude calculation frequency can reach up to 200Hz.

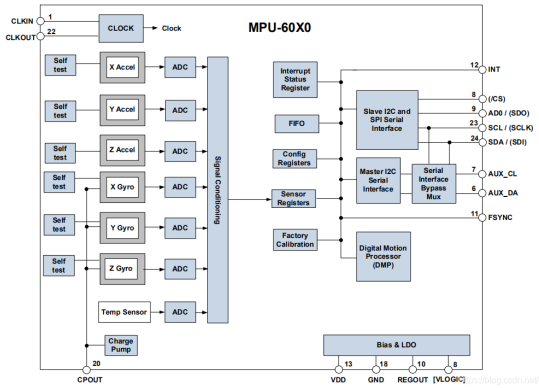


Figure 3.38: Internal structure of MPU6050

The main parameter description is provided in the table below [9].

As for the pin connection, the MPU6050 has 8 pins in total. Apart from the power input and ground pins, 4 pins are used for IIC communication and clock between master and slave devices, and one pin denotes the address of slave. Finally, the last pin is used to intermediate the output pin. A detailed pin illustration is shown in the table below.

Code Explanation

Porting the official MSP430-based DMP MPU6050 drive program, 6 code files are included: anbt_mpu_dmp_-driver.c, anbt_mpu_dmp_driver.h, anbt_mpu_dmp_mpu6050.c, anbt_mpu_dmp_mpu6050.h, mpu_i2c.h, mpu_i2c.c, in which anbt_mpu_dmp_mpu6050.c is used mainly for setting and initializing register, mpu_i2c.h is the library function file used for MPU6050 IIC communication.

As for the initialization function, firstly MPU is initialized by waking up 6050, reading the power consumption state of accelerator and entering high-power-consumption state, setting the initialized standard bit of configuration data, determing relevant parameters, shutting down passby mode, and configuring the sensor including its clock, itself and its power consumption mode. Secondly, there is a return value justification function. If the return value is greater than 0, the contol unit STM32F103RCT6 will restarted. Thirdly, the clock of gyroscope and the accelerator and their corresponding work state is set, then the First Input First Output(FIFO) mode is set by writing I2C commands to activate the FIFO channel in accelerator and gyroscope.

Then, the sampling rate is set $1000/(1+data)$, and the DPL is set to be 42Hz. Afterwards, the frequency sampling range function and the range function of the gyroscope are obtained, as well as the range function of the accelerator. The array hal which is used as a standard bit, is cleared to zero, and the DMP mapping function is loaded and verified. In the next step, the directional matrix from gyoscope and the accelerator is pushed to DMP, and the corresponding functions of DMP are enabled and used. Finally, the presently set state is marked as the origin point in the coordinate system through DMP self-testing.

Parameter	Description
Power Supply	3.3-5V
Communication Port	IIC(maximum clock frequency 400KHz)
Measuring Dimension	acceleration: 3-D gyroscope
ADC resolution	Acceleration: 16-bit gyroscope
Acceleration measurement range	±2g, ±4g, ±8g, ±16g(gravitational acceleration constant, g=9.8m/s^2)
Acceleration highest resolution	16384 LSB/g
Acceleration measurement accuracy	0.1g
Acceleration output frequency	1KHz maximum
Gyroscope measurement range	±250/s, ±500/s,±1000/s, ±2000/s
Gyroscope highest resolution	131 LSB/(/s)
Gyroscope measurement accuracy	0.1/s
Gyroscope output frequency	8KHz maximum
DMP Attitude Resolution Frequency	200Hz maximum
Temperature sensor measuring range	-40 +85°C
Temperature sensor resolution	340 LSB°C
Temperature sensor accuracy	±1°C
Working temperature	-40-+85°C
Power consumption	500uA-3.9mA(working voltage 3.3V)

Table 9: Main parameter description of MPU6050

Pin Name	Illustration
VCC	3.3/5V power input
GND	ground
SCL	I2C slave clock signal line(pull-up resistor is connected to the module)
SDA	I2C slave data signal line(pull-up resistor is connected to the module)
XDA	I2C master data signal line(pull-up resistor is connected to the module)
XCL	I2C master clock signal line(pull-up resistor is connected to the module)
AD0	slave address: 0X68(GND or floating) or 0X69(connecting VCC)
INT	Intermediate output pin

Table 10: Pin illustration of MOU6050

Below is the initialization code.

```
    u8 AnBT_DMP_MPU6050_Init(void)
{
    MPU_I2C_Init_IO();                          //I2C initialization
    Ex_NVIC_Config(GPIO_C,14,FTIR);                 //falling-edge triggered
    MY_NVIC_Init(0,0,EXTI15_10_IRQn,2);       //subpriority2
    AnBT_DMP_MPU6050_DEV_CFG();
    if(!mpu_set_sensors(INV_XYZ_GYRO | INV_XYZ_ACCEL));
    if(!mpu_configure_fifo(INV_XYZ_GYRO | INV_XYZ_ACCEL));
    if(!mpu_set_sample_rate(DEFAULT_MPU_HZ));
    if(!dmp_load_motion_driver_firmware());
    if(!dmp_set_orientation(inv_orientation_matrix_to_scalar(gyro_orientation)));
    if(!dmp_set_fifo_rate(DEFAULT_MPU_HZ));
    run_self_test(0);
    if(!mpu_set_dmp_state(1));
    return 0;}
```

As to update the heading angle, a function MPU6050_Pose is used. In it, the callback function is collected completely and the value of hal.gyro is set 1. Then, the value in FIFO is outputted as dmp.fiforead. To ensure there is nothing left in FIFO, a query function is used, and if there is none, the value of hal.gyro is 0. Finally, the quaternion

is converted to Euler angles through formulas. Below is the corresponding code.

```
    u8 AnBT_DMP_MPU6050_Init(void)
{
    void MPU6050_Pose()
    {
     dmp_read_fifo(MPU6050_gyro, MPU6050_accel, quat, &sensor_timestamp, &sensors,&
         more);
     if (sensors & INV_WXYZ_QUAT )
    {
        q0=quat[0] / q30;
        q1=quat[1] / q30;
        q2=quat[2] / q30;
        q3=quat[3] / q30;
        Pitch = asin(-2 * q1 * q3 + 2 * q0* q2)* 57.3;
        Roll =  -atan2(2 * q2 * q3 + 2 * q0 * q1, -2 * q1 * q1 - 2 * q2* q2 + 1)*
            57.3;
       Yaw =   atan2(2*(q1*q2 + q0*q3),q0*q0+q1*q1-q2*q2-q3*q3) * 57.3;
    // printf("pitch:
 }
}
```

Results and Analysis

The gyroscope was added during the test field process after our team found that there existed a turning problem of the car, which means that the car would sometimes overturn or turn less than desired, and this has brought tracing problems that the car had a rather high probability losing the path and would drive off the routine. After adding the gyroscope, the turning angle could be captured by the gyroscope and processed, thus it was able to get the desired turning angle and the car would stay in the path.

3.5.6 Hall encoder

Main Contributor: Liu Zhe

An encoder is a device that compiles and converts signals or data into signals that can be used for communication, transmission and storage. The encoder converts angular displacement or linear displacement into electrical signals. It is a commonly used motor positioning device in the industry, which can accurately test the angular displacement and rotational position of the motor. The most direct function of the encoder is to measure the displacement. Once the displacement is known, the speed can be calculated.

One category of encoders is the Hall encoder and this is the type chosen. It is composed of a Hall code disc and a Hall element. The Hall code disc is equally arranged with different magnetic poles on a circular plate with a certain diameter. The Hall code disc is coaxial with the motor. When the motor rotates, the Hall element detects and outputs several pulse signals. In order to judge the steering, two groups of square wave signals with a certain phase difference are generally output (usually a difference of 90).

It can be seen from the figure below, when rotating clockwise, when phase A is on the falling edge, phase B is on high level, and when phase A is on the rising edge, phase B is on low. When rotating counterclockwise (that is, looking at the above waveform from right to left), phase B is low when phase A is on the falling edge. When phase A is on the rising edge, phase B is high.

The pin description of the Hall encoder is shown in the table below [10].

Encoder Control Motor-GB37520

By using the Hall encoder, the original TT motor as shown in Figure3.21 could be replaced, which is the yellow TT motor. Since the Hall encoder can provide motor data PID feedback control and generate a larger torque, the Hall

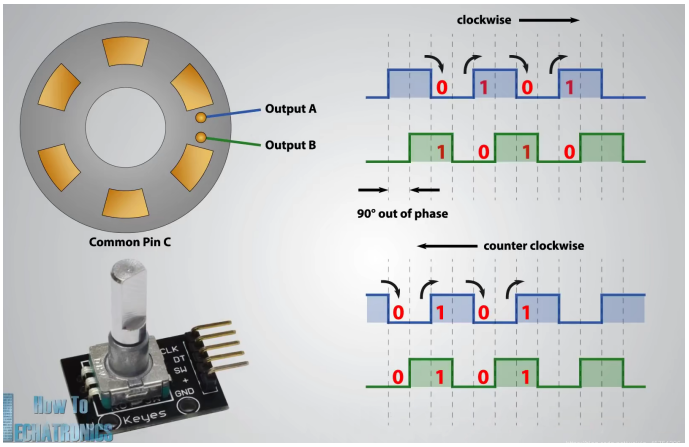


Figure 3.39: Hall disc outlook

Encoder	STM32
A Phase	Pulse detection interface(in STM32)
B Phase	Pulse detection interface
+5V	+5V
GND	GND

Table 11: Encoder-STM32 pin connection

encoder is chosen which combined DC motor GB37520.

The GB37520 motor is a geared motor, which means that there is a reducer at the head of the motor, and the motor rotates to drive the gears in the reducer, and then outputs the corresponding speed through the output shaft of the reducer. For example, the reduction ratio is 1:30, which means that the motor rotates 30 times, and the reducer of the motor rotates once.

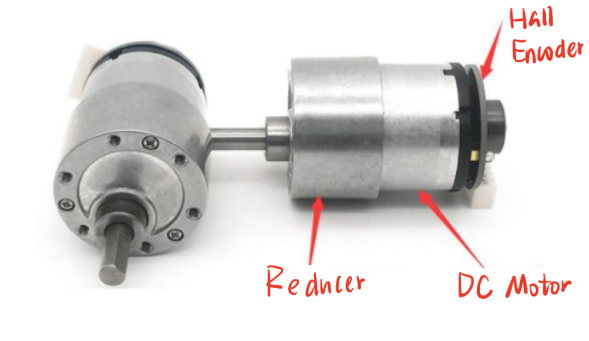


Figure 3.40: General structure of Hall encoder

The table below is a detailed description of the DC gear motor GB37520BMQ.

M-method speed measurement

The equation below is used to calculate the rotating speed of the DC motor.

$$n = \frac{M_0}{C \times T_0}$$

in which,

- T_0 is a user-set time;
- $M_0$is the number of pulses in the set time;
- C is the total number of pulses per revolution.

Encoder	Motor Drive
M+	The output OUT1 of the motor drive
M-	The output OUT2 of the motor drive

Table 12: Encoder-Motor drive pin connection

Parameter	Illustration
No-load rotating speed	333 RPM
No-load current	120mA
Rated torque	3.5N/cm
Rated voltage	12V
Rated current	1A
Rated power	12W
Weight	190g
Stall current	2.3A

Table 13: Encoder-Motor drive pin connection

The shaft of the DC motor rotates once, and each pin of the Hall sensor has 13 pulse signal outputs.

The magnetic ring here is fixed on the shaft of the DC motor, which is different from the output shaft of the gear motor. The output shaft of the deceleration motor is transformed by its gear. The deceleration ratio of the motor was introduced before as 1:30. Therefore, if the output shaft of the deceleration motor rotates once, the encoder pulse per phase can actually be detected. The quantity is: $13 \times 30 = 390$ pieces. Then 4 times the frequency through the encoder interface of the micro-controller is 1560 pulse signals, which is very useful for us to calculate the actual displacement and speed.

The calculated n is the speed of the DC motor, which can be used normally, but a geared motor is used. In order to get the real speed of the output shaft, it is necessary to divide the reduction ratio.

Code Explanation

Below is a part of the PID algorithm, which is responsible for speed adjustment, and it is included in the control.c file. The speed adjustment of the four motors are generally controlled using PID algorithm, which uses proportion, integral and differential adjustment to utilize better control of the 4 car motors. For a more specific introduction of the PID algorithm, please refer to Section 4.5.4.

```
1  if(Contorl_stick==4)
2          {
3               Contorl_stick=0;
4               Read_Encoder(); //Read the Encoder
5               if(ARMED)
6               {
7                    for(nmr=0;nmr<4;nmr++)
8                    {
9                         PID_Motor[nmr].error=Encoder[nmr]-Motor_Speed[nmr];
10                        //output P value
11                        PID_Motor[nmr].pout = PID_Motor[nmr].Pdat * PID_Motor[nmr].error;
12                        //Output the value of I
13                        PID_Motor[nmr].iout += (PID_Motor[nmr].Idat*0.05) * PID_Motor[nmr
                              ].error;
14                        PID_Motor[nmr].iout = Get_MxMi(PID_Motor[nmr].iout,200,-200);
15                        PID_Motor[nmr].dout = PID_Motor[nmr].Ddat*0.05*(PID_Motor[nmr].
                              error-PID_Motor[nmr].Last_error);
16                        PID_Motor[nmr].Last_error=PID_Motor[nmr].error;
```

```
                PID_Motor[nmr].OUT += PID_Motor[nmr].pout + PID_Motor[nmr].iout +
                    PID_Motor[nmr].dout; ////P,I,D values added
                    together
                PID_Motor[nmr].OUT=Get_MxMi(PID_Motor[nmr].OUT,4599,-4599);
            }
            Set_Motor(PID_Motor[0].OUT, PID_Motor[1].OUT, PID_Motor[2].OUT,
                PID_Motor[3].OUT);
        }
        else
        {
        }
    }
```

As for the timer encoder count part, it initializes TIM1, TIM2, TIM3 and TIM4 to encoder interface mode by enabling the corresponding clocks, setting the CRL registers to set the port into floating input mode. As an example, only one motor initialization is chosen, because the initialization process of the other three motors is quite similar. As for the clock, the clock division is defined as no frequency division and the count mode is chosen to be the edge-aligned mode. A register called CCMR is used, and it is set as CC1 and CC2 channel are configured as input, IC1 and IC2 are mapped on TI1 and TI2 correspondingly. Then, the input sampling frequencies of the input capture filter 1 and 2 are defined, and finally the timer is enabled. In a function called Encoder_Sum, the four timers are distributed to the four wheels of the car accordingly, with TIM2 being the left rear wheel, TIM8 being the right rear wheel, TIM3 being the right front wheel and TIM4 being the left front wheel. Finally, to read the values stored in the 4 encoders, the corresponding register CNT is used, in which the converted values are stored, and these values are given to the array named Encoder_Sum with length 4.

Below is the specified code with detailed notation.

```
#include "encoder.h"
short int Encoder[4];
void Encoder_Init(void)
{
    Encoder_Init_TIM8();
    Encoder_Init_TIM2();
    Encoder_Init_TIM3();
    Encoder_Init_TIM4();
}

void Encoder_Init_TIM8(void)
{
    RCC->APB1ENR|=1<<0;         //enable TIM2
    RCC->APB2ENR|=1<<13;         //enable TIM8
    RCC->APB2ENR|=1<<4;        //enable PC clock
    GPIOC->CRL&=0X00FFFFFF; //PC6,7
    GPIOC->CRL|=0X44000000; //floating input

    TIM8->PSC = 0x0; //prescaler enabled
    TIM8->ARR = 0xFFFF; //set counter auto-reload value
    TIM8->CR1 &=~(3<<8); // no frequency division
    TIM8->CR1 &=~(3<<5); // select counting mode: edge-aligned mode
    TIM8->CCMR1 |= 1<<0; //CC1S='01' IC1FP1 maps to TI1
    TIM8->CCMR1 |= 1<<8; //CC2S='01' IC2FP2 maps to TI2
```

```
    TIM8->CCER &= ~(1<<1);      //CC1P='0' IC1FP1 no phase inversion, IC1FP1=TI1
    TIM8->CCER &= ~(1<<5);      //CC2P='0' IC2FP2 no phase inversion, IC2FP2=TI2
    TIM8->CCMR1 |= 3<<4;       // IC1F='1000', input capture 1 filter
    TIM8->CCMR1 |= 3<<12;  // IC1F='1000', Input capture 2 filter
    TIM8->SMCR |= 3<<0;    //SMS='011', all inputs are valid on rising and falling edges
// TIM8 -> CNT=5000;
    TIM8->CR1 |= 1<<0;      //CEN=1, enable the timer
}

short int Encoder_Sum[4]={0,0,0,0};
void Read_Encoder(void)
{
    Encoder[0]= (short int)TIM2 -> CNT;   TIM2 -> CNT=0; //left gear wheel
    Encoder[1]= (short int)TIM8 -> CNT;   TIM8 -> CNT=0; //right gear wheel
    Encoder[2]= (short int)TIM3 -> CNT;   TIM3 -> CNT=0; //right front wheel
    Encoder[3]= (short int)TIM4 -> CNT;   TIM4 -> CNT=0; //left front wheel
    Encoder_Sum[0]+=Encoder[0];
    Encoder_Sum[1]+=Encoder[1];
    Encoder_Sum[2]+=Encoder[2];
    Encoder_Sum[3]+=Encoder[3];
}
```

Results and Analysis

The Hall encoder was not chosen as the motor of the car at first, because it is thought that the TT gear motor could provide enough torque and power for the car to cross the bridge. Unfortunately, during the test field process, it was found that the car did not have enough power to cross the bridge and the vibration of the car and its tracing was quite obvious. To obtain larger torque and better control of the drive system of the car, a Hall encoder is chosen as the corresponding motor. Then, when tested in the field, the car can successfully rush to the top of the bridge.

3.5.7 Wheel Selection

Main Contributor: Shui Jiajun

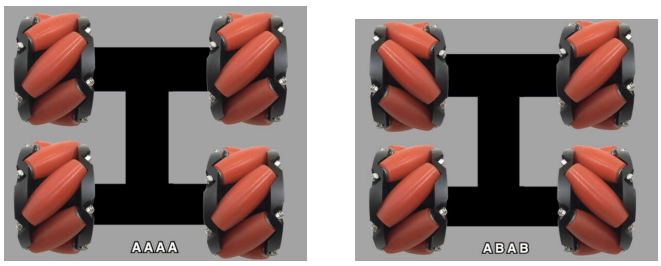
Initially, the Mecanum wheel is chosen to complete the driving of the car, as it is a type of an omnidirectional mobile wheeled structure invented by the Swedish Mecanum company. It is mainly composed of a main body hub and a set of uneven rows of rotating rollers. The outer envelope of the rotating roller is a cylindrical surface, so the wheel can roll forward continuously. The arrangement of the rollers and the main hub will present a specific angle, generally 45 degrees [11].

Mecanum wheels can be divided into two types: A and B. Since the rollers and the main hub are at 45 degrees with each other, both types of the wheels can have horizontal speed component which is perpendicular to the movement direction, so as to realize the steering function of the trolley.

From Figure 3.41, When the A wheel rotates forward, a left horizontal speed component could be decomposed, while when it rotates backward, a right horizontal speed component would occur. If four A wheels are chosen, the car will drift to the left while moving forward. Therefore, the AAAA scheme is not a stable scheme and is not desirable. The B type has the opposite horizontal component compared to the A type, which means that a right horizontal speed component would be derived when moving forward.

If four wires are chosen as ABAB as shown in Figure 3.41, this scheme could effectively offset the horizontal components generated by the four wheels of the trolley when moving forward or backward, so that the trolley can move in a stable and controlled manner.

Mecanum wheels possess some advantages as well as some drawbacks. First, it is universal and can easily



(a) AAAA wheel choice using Mecanum wheels

(b) ABAB wheel choice using Mecanum wheels

Figure 3.41: Mecanum wheel selection scheme

change direction. Meanwhile, it also has high stability, which makes the operation of the car more stable. These beneftis consist of reasons why we initially chose the Mecanum wheels. However, as for the disadvantages, the operation efficiency of the Mecanum wheel is relatively low, which is due to the fact that when it works, it will consume a part of the speed component in a certain direction, so that the wheel loses more power. Secondly, the ground contact of the wheel is poor. Since rollers on the Mecanum wheels are cylindrical, if the ground is not level, the wheels will slide and deviate from the preset track. Finally, the wheel is only suitable for use at low speeds to reduce the disturbance caused by the rollers. Despite the drawbacks mentioned above, one is lethal for the completion of this car task, which is that the vibration and bumpiness brought by Mecanum wheels are quite severe that has caused deviation of route and brought great inconvenicen for debugging. In addition, the Mecaum wheel is not large enough, thus if we insist on using it, the mechanical arm will need to be increased to a height that might influence the stability of the car. Therefore, we did not choose the famous Mecanum wheels, instead, we chose a normal type of wheels made by a Chinese company called WHEELTEC with a larger diameter.

Figure 3.42: Outlook of the wheel we choose

Compared with the Mecanum wheel, this wheel has a larger wheel diameter, uses electricity more efficiently, and at the same time, the effect of shock absorption is better, so that the task can be better completed. The vibration of the car will be relatively small, and it will not deviate from the route. Standing from a economic point, using this type of wheels saves our budget significantly. However, it also has some disadvantages. For example, the ground friction of this wheel is quite large, resulting in the turning insensitivity problem and the car body vibration problem. This is considered as a trade-off between the sensitivity problem and successfully crossing the bridge and more accurate distance control.

The following are some parameters of this wheel.

Below is a picture showing the wheels being installed on our car.

As mentioned in Section 2.2, the motor drive circuit chosen is the TB6612FNG. Below is the code implementation of the drive, thus the wheel could be controlled and drived properly. To begin with, the IO port is set into PWM mode. In detail, TIM5 and GPIOA external clock is enabled, and the 8 ports mode of GPIOA is set as multiplexed output

Parameter	Value
Wheel width	30.5mm
Diameter	85mm
Weight	48g
Coupling device	5mm, 6mm, 8mm
Coupling weight	23.6g, 21.4g, 19.1g

Table 14: Wheel parameters

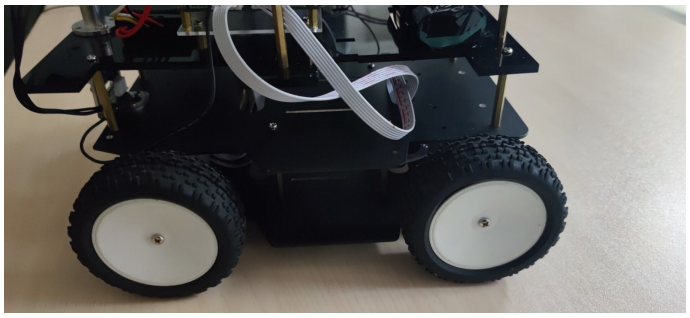


Figure 3.43: The chosen wheels on the car

mode by setting the CRL register. Then, to set the timer TIM5, the maximum value counted by counter is automatically loaded in the register ARR while the prescaler is set as no division. Correspondingly, the ARPE bit is high to enable auto-reloaded prescaler, and the CEN bit is set as 1 to enable the counter. In the register CCMR, the four channels of the TIM5, which are mapped to the four motors, are set in the PWM mode1, which means that high voltages will be outputted if the CNT values are greater that the CCR1 value in count-up mode and less for high in count-down mode. As a result, the PWM width is controlled by adjusting the CCR1 value. Then, by setting the register CCER, the four channels are output enabled. Then, in the second function Motor_Init, the four ports of the two motor are set into push-pull output mode, and the clock frequency is set as 8kHz. Finally, in the third function, Set_Motor, after referring to the TB6612 table, the motor mode of the two motors controlled are specified. For simplicity, we show the motor control of motor A only.

```
1
2  #include "motor.h"
3
4  void TIM5_PWM_Init(u16 arr,u16 psc)
5  void Motor_Init(void)
6  {
7      TIM5_PWM_Init(4499,1);  //PWM frequency =8khz
8  TIM5->CCR1=0;                 //CH1 Set duty cycle
9
10     RCC->APB2ENR|=1<<4;          //GPIOC clock enable
11     GPIOC->CRL&=0XFF0000FF;      //PC2,3,4,5 Set as strong push-pull output
12     GPIOC->CRL|=0X00333300;
13     GPIOC->ODR|=0xF<<2;            //PC2,3,4,5 pull-forward
14 }
15 // control motor speed 1000-2000
16 void Set_Motor(int MotorA,int MotorB,int MotorC,int MotorD)
17 {
18     if(MotorA>0)
19     {
20         MotorA1=0,            MotorA2=1;
```

```
21          TIM5->CCR2=MotorA;
22      }
23      else
24      {
25          MotorA1=1,              MotorA2=0;
26          TIM5->CCR2=-MotorA;
27      }
28  }
29  void Motor_Init(void); //Motor PWM initializes and sets the motor's refresh rate 50-499
30  void Set_Motor(int MotorA,int MotorB,int MotorC,int MotorD);
31
32  #endif
```

Results and Analysis

The wheel was changed mainly because that the during test field, the Mecanum wheels would generate much vibration due to the uneven road surface. As a result, it became quite often for the car to drive off the path. In addition, the radius of the Mecanum wheels is not large enough to allow the car to reach the basket, although the increased height given by the new wheels is not very significant, about 2.5cm. However, considering both of the reasons, this normal type of wheels is chosen. After conducting the test field, it is found that the vibration was reduced dramatically, and the car could finally stay into the path. Additionally, after changing to a larger type of wheels, along with replacing the taller cooper pillar, increasing the car layers and extending the mechanical arm, the car could finally reach the height of the basket and the ball shooting task could be finally accomplished, which indeed is a collaborative achievement.

3.6 Car Group Contribution Summary

In summary, the Car Control group completed hardware tasks including the circuit and PCB design, the car mechanical structure design and assembling the car. Besides, the group also managed to solve the code implementation of peripherals controlled by the STM32F103RCT6, which are the ultrasonic sensor HC-SR04, the self-designed mechanical arm combined with servo, the OLED display screen, the gyroscope, the Hall encoder and the motor drive TB6612FNG. As for the effects of our car, in the first two field tests, there existed some problems related to the car structure, especially the height of the car and the wheel we chose. To solve the height problem, the number of the car layers is changed from two to four, while to solve the problem of not being able to cross the bridge, we replaced the Mecanum with normal wheels, and used the Hall encoder to generate larger torque. It can be said that a majority of problems faced by the Car Control Group are solved through conducting field tests and debugging and raising solutions afterwards. At last, the appearance of our car might not be very beautiful, but it can finish the required tasks.

4 Image Recognition: Visual Recognition and Patrol

In this section, the tasks related to visual recognition and patrol will be finished based on the implementation of OpenMV. To be specific, firstly the theoretical basis of camera imaging is introduced including overall imaging principle, field distortion and image digitization. Then the recognition of shape, basket and Apriltag are carried out to satisfy the requirement of Partio 2. As for the most complex patrol module, firstly two alternatives of infrared and mmWave detection are evaluated but abandoned due to cost or imperfect performance. The method of real-time imaging for target path searching is proposed and has been verified to be generally valid through the field test in spite of the possible human shadow interfering. After that, the ultrasonic module is designed to detect distance as an assistance for controlling. In the end, the communication between OpenMV and STM32 are decribed in detail.

4.1 Theoretical Basis of Camera Imaging

In this part, theoretical basis of camera imaging principles and conceptions of field curvature and image digitization are briefly introduced before our design.

4.1.1 Overall Imaging Principle

The camera module always uses the lens group to realize image imaging based on the principle of light refraction and propagation through lens [12]. The refracted light will hit on a photosensitive chip, which can convert light information including wavelength and intensity into a digital electrical signal, and this can be recognized by a digital circuit. The photosensitive element on the camera is shown in the figure, and the photosensitive chip and related circuits inside convert the captured optical signal and transmit image information [13].

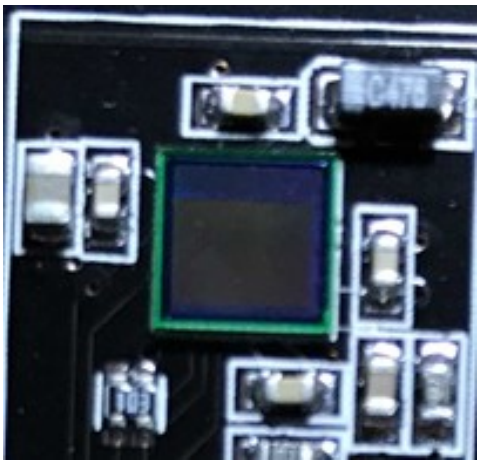


Figure 4.1: Light-sensitive element (in the center) [12].

4.1.2 Focal Length

The focal length refers to the distance between the two optical focal points of the lens. The light reflected by the object is refracted by the lens, and then hits the photosensitive element for imaging [13]. Thus the parameters of the lens, especially the focal length, will determine the imaging size, the field angle, the field depth, the perspective strength of the picture and other important imaging parameters. In general, for the same subject at the same distance, the longer focal length implies larger the image size and smaller field angle [14].

4.1.3 Field Curvature and Distortion

The simplest model for camera imaging analysis is the pin-hole model, which is an ideal optical system. However, the pin-hole model does not take into account the field curvature, distortion, etc., which are produced in the edge of lens for different imaging distances in actual cameras.

These problems are always corrected by using an code algorithm in practice. For example, OpenMV uses $image.lens\_corr(1.8)$ to correct a 2.8mm focal length lens [15]. In addition, it is also possible to directly use a distortion-free lens with complex additional optical modules to correct the distortion [12]. Figure 4.2 is the imaging comparison of different lenses when the camera in OpenMV is 20cm away from the desktop.

4.1.4 Image digitization

Image digitization is the most basic step of computer image processing. Its significance is to transform the real image into a format acceptable to the computer, that is, a series of specific numbers. Common scanners are this process. Usually, this digitization process can also be divided into two steps: sampling and quantization. The result

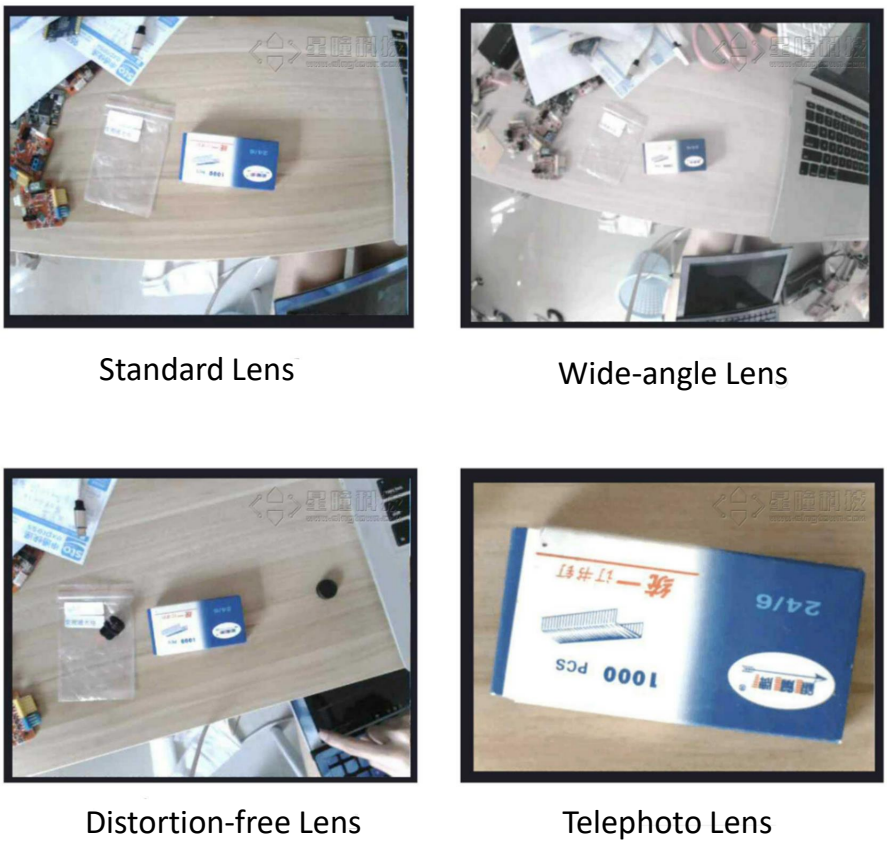


Figure 4.2: Imaging comparison of different lenses when the camera in OpenMV is 20cm away from the desktop.

of "sampling" is usually called image resolution, while the result of "quantization" is the total number of colors that the image can hold [13].

Sampling is a process of using pixels to represent an image. For example, a 640 * 480 image is composed of 307200 points. Of course, if you want to have clearer image quality, you have to use more points to represent the image, that is, to make the image have higher resolution [14].

Quantization is a process to use a wide range of values (color number) to represent each point after image sampling. This numerical range includes the total number of colors that can be used on the image. Eg: storing a point with 4 bits means that the image can only have 16 colors. The larger the numerical range, it means that the image can have more colors, which can naturally produce more realistic image effects.

4.2 Shape Recognition

Main Contributor: Li Muquan

In this part, we use OpenMv to carry out the shape recognition to complete the corresponding tasks in Patio 2.

4.2.1 Proposed Methods

Method A: Directly call functions of $find\_circles()$ and $find\_rects()$ in OpenMv. These two functions use Hough transformation to identify circles and quaternion detection to identify rectangles [15]. Thus we can directly call these functions to finish the task. The advantage of this method is the convenience. However, there is no algorithm for identifying triangles stored in OpenMV, thus the algorithm needs to be further redesigned.

Method B: Use the proportion of the color block to the minimum circumscribed rectangle area for identification. To be specific, firstly use color recognition of OpenMV to identify black block, then frame the minimum circumscribed rectangle of the identified color block. Finally we can call the $blob.density()$ function to calculate the proportion of the

color block to the minimum circumscribed rectangle area [15]. In particular, these three shapes, round, rectangle and triangle, all correspond to a range of different proportions. Therefore, calculating the shape occupancy can identify the corresponding shape. The advantage of this method is that it is very accurate, as well as also convenient since only one single function is used to finish the recognition of three shapes.

We finally chose **Method B** to carry out the identification of these three shapes with consideration of both convenience and performance.

4.2.2 Recognition Process

Firstly we print pictures with these three graphs, and then adjust the black color threshold according to the light interference of the environment (here the graphics are all black).

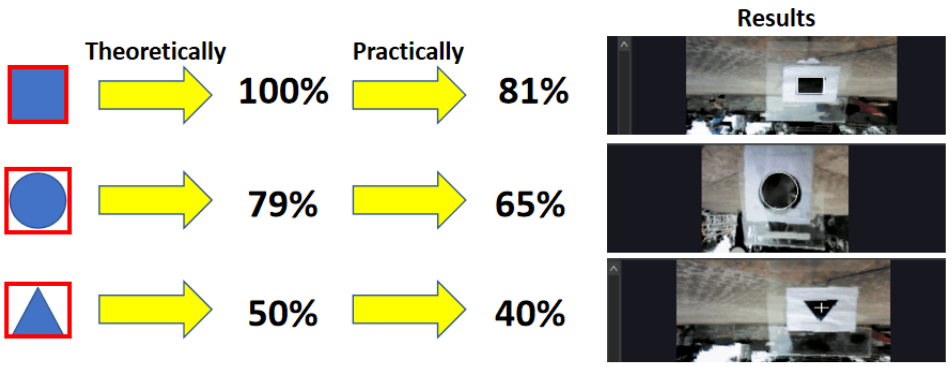


Figure 4.3: Shape recognition process.

After initializing the settings, OpenMV will keep taking pictures. We write codes to use the $find\_blobs$ function to find blobs within the set threshold [15]. In particular, OpenMV will automatically frame the smallest bounding rectangle of the identified color block when performing color recognition, which provides great convenience for us to find the target.

After consulting the technical manual, the $blob.density()$ function can calculate the ratio of the color block area to the area of its smallest bounding rectangle [15]. Thus for rectangles, triangles and circles, the ratios corresponding to each shape are quite different, which can be considered as a powerful evidence to differentiate them.

For a rectangle, theoretically it should completely coincide with its enclosing rectangle. However, taking into account a slight error the threshold for determining a rectangle is set between 0.805 and 1. As for circle, theoretically the ratio of the circle to its circumscribed rectangle being 0.79, thus the threshold is set to be within 0.65 to 0.805. Finally, the ratio of equilateral triangle accounting for its circumscribed rectangle is varied since it is too sensitive to the direction influence cannot be determined stringently. However, our test results represent that shapes with a ratio less than 0.65 and greater than 0.4 can be judged as triangles. Additionally, in our tests, the ratio less than 0.4 is mostly caused by the interference, thus this interval is discarded.

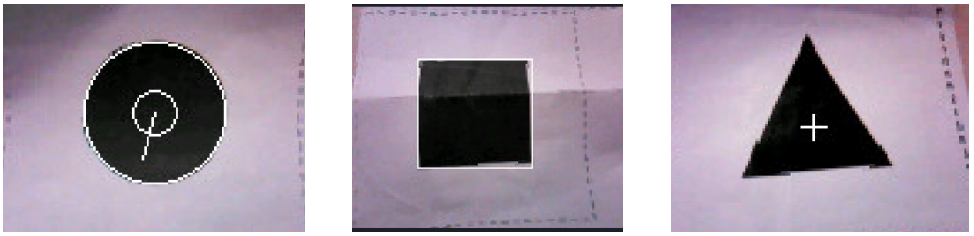
We give the final threshold result after multiple tests. The corresponding code is shown below.

```
import sensor, image, time, math
thresholds = [(0, 40, -128, 13, -94, 22)]

sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QQVGA)
sensor.skip_frames(30)
sensor.set_auto_gain(False)
sensor.set_auto_whitebal(False)
clock = time.clock()

while(True):
    clock.tick()
```

```
    img = sensor.snapshot().lens_corr(1.8)
    for blob in img.find_blobs(thresholds,pixels_threshold=200,area_threshold=200):
        if blob.density()>0.805:
            print("retangle")
            img.draw_rectangle(blob.rect())
        elif blob.density()>0.65:
            print("circle")
            img.draw_keypoints([(blob.cx(), blob.cy(), int(math.degrees(blob.rotation
                ()))),], size=20)
            img.draw_circle((blob.cx(), blob.cy(),int((blob.w()+blob.h())/4)))
        elif blob.density()>0.40:
            print("triangle")
            img.draw_cross(blob.cx(), blob.cy())
        else:
            print("no dectedtion")

    print(clock.fps())
```

4.2.3 Test Results

After many field tests, perfect shape recognition performance is achieved in our system. As shown in Figure 4.4, when one of these three shapes appears in the field of view, the target shape will be automatically detected and framed, then the serial port terminal will display the recognized shape. Particularly, when all three shapes are shown simultaneously in the camera's field of view, each shape will be also detected, and the serial terminal will output all the recognized results, which is shown in Figure 4.5.



(a) Circle (b) Rectangle (c) Triangle

Figure 4.4: Shape recognition.

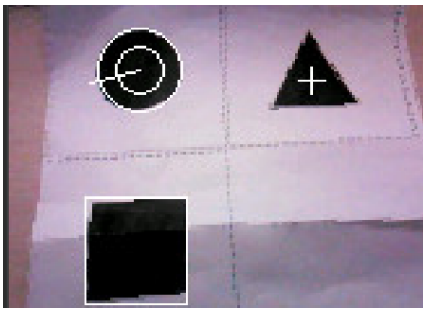


Figure 4.5: Recognition when three shapes appear simultaneously.

4.3 Basket Recognition

Main Contributor: Li Muquan

In this part, the design of basket recognition is illustrated to make the car find the basket thus can use the robotic arm to shoot the ball into the basket.

4.3.1 Proposed Methods

Method A: Only ultrasonic ranging. The reflected ultrasonic wave demonstrates the distance between the car and the basket and shows the position of the basket. However, this method requires the accurate direction calibration between the car and the basket to avoid the ranging deviation, which may not be perfectly satisfied in the pratical senerio.

Method B: Combining color recognition and ultrasonic ranging. Firstly OpenMV's color recognition module is used to detect the black block with the minimum circumscribed rectangle. Then functions *blob.cx()* and *blob.cx()* are called to return the center x coordinate and y coordinate of the color block's outer frame, thus the center coordinates of the basket can be obtained. This center coordinates contribute to make the front of the car align with the basket, calibrating the car and the basket. Then ultrasonic ranging can be uses. The advantage are not only the calibration alignment, but also the distance evaluation between the car with the basket.

Thus in order to achieve the function of shooting, we choose the second option.

4.3.2 Design process

Firstly we make the graphic pictures of the target basket (the color is black), and then adjust the black threshold according to these pictures. After that, the function *find_max* is called to find the largest black block in the field of view, which corresponds to the target basket. Then it will be framed with the smallest bounding rectangle and functions *blob.cx()* and *blob.cy()* will be used to return the center x coordinate and y coordinate of the rectangle. In this way, the center coordinates of the basket in the field of view are obtained, which will be sent to the control module. Based on this position information, the front of the car will be controlled to align with the basket.

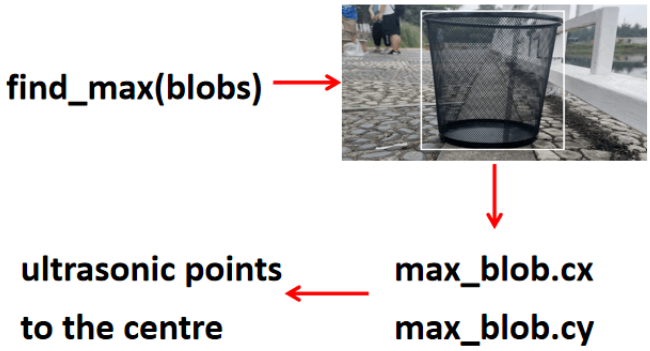


Figure 4.6: Basket recognition process.

The codes below demonstrate the details of basket recognition.

```
import sensor, image, time, math

black_threshold = (8, 22, -60, -3, 127, -128) #Set the black threshold of the basket (unadjusted)

sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QVGA) #Set the image size QVGA size to 320*240, so the center coordinate
    should be (160, 120)
sensor.skip_frames(time = 2000)
sensor.set_auto_gain(False) #Turn off the gain
sensor.set_auto_whitebal(False) #Turn off white balance
clock = time.clock()

def find_max(blobs):
    max_size=0
    for blob in blobs:
        if blob.pixels() > max_size: #blob.pixels() returns the number of pixels (int) that are part of the
            blob
```

```
            max_blob=blob
            max_size = blob.pixels()
    return max_blob

while(True):
    img = sensor.snapshot() #Take a picture and return the image.
    blobs = img.find_blobs([black_threshold])
    if blobs:
        max_blob=find_max(blobs)
        img.draw_rectangle(max_blob.rect()) #Draw the minimum bounding rectangle of the basket
        #blob.cx() returns the center x coordinate (int) of the outer frame of the color block, blob.cy() returns the
            center y coordinate (int) of the outer frame of the color block
        img.draw_cross(max_blob.cx(), max_blob.cy()) #Draw a cross at the center coordinates of
            the rectangle
        output_str="[%d,%d]" % (max_blob.cx(),max_blob.cy())

        print('Center point coordinate:',output_str) #Print the coordinates of the center point of
            the color block to the serial terminal for data verification
        sending_data(max_blob.cx(),max_blob.cy()) #Send the coordinates of the center point of the
            color block frame How to send? ? ?
        print(clock.fps()) #Print frame rate

    else:
        print('404 not found!')
        sending_data(567,789)#If no matching color block is found, send an impossible coordinate
```

4.3.3 Testing results

Figure 4.7 shows the recognition and detection result of the basket. It can be seen that the target basket is successfully detected, which provide the necessary location information for the car to carry out the subsequent process for ball throwing.

Figure 4.7: Basket recognition result.

4.4 Apriltag Recognition

Main Contributor: Li Muquan

In this part, the design of Apriltag recognition is illustrated in detail. This is served as a location assistance for range and direction measurement to prevent deviations.

4.4.1 Proposed Methods

Method A: Identifying the TAG16H5 family in Apriltag: The effective area of Apriltag is the 4*4 block inside, thus it can be identified directly by the function *find_apriltags()*. The advantage of this method is that the TAG16H5 family has the high capacity of long-distance identification, achieving the perfect performance of long-distance positioning.

However, the non-neglectable disadvantage is the low positioning and ranging accuracy resulted from the inefficient collected information and effective area of Apriltag.

Method B: Identifying the TAG36H11 family in Apriltag: The effective area of Apriltag in this method is a 6*6 block, which is larger than that of the TAG16H5 family. Thus this family contributes to have more data and verification information, achieving a larger identification accuracy. However, the shortcoming of this method is that the effective working distance is smaller than that of the TAG16H5 family.

Since the position where the Apriltag is placed is not far from the car, i.e., the working distance is not required to be large, we choose the **Method B** with higher identification performance.

4.4.2 Design process

The process of Apriltag recognition is shown in Figure 4.8.

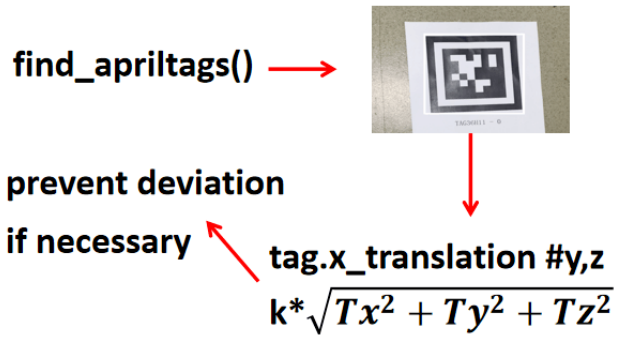


Figure 4.8: Apriltag recognition process.

Firstly, the function *find_apriltags()* can be called directly in OpenMV, through which we can obtain the the apriltag family. Then a converting function, *degrees()*, is defined to transform the deflection radians in each dimension into angles (angles are more convenient for us to understand in the designed system). In this way, the three-dimensional coordinates of the target apriltag and the deflection angles of three directions can be easily obtained. See the code below for details.

```
#identifying the Apriltag and ranging
import sensor, image, time, math

sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QQVGA)  # the iamg size
sensor.skip_frames(30)
sensor.set_auto_gain(False)
sensor.set_auto_whitebal(False)
clock = time.clock()

f_x = (2.8 / 3.984) * 160 # (Default)
f_y = (2.8 / 2.952) * 120 # (Default)
c_x = 160 * 0.5 # X-coordinate of the central position of the image (Default)
c_y = 120 * 0.5 # Y-coordinate of the central position of the image (Default)

def degrees(radians):
    return (180 * radians) / math.pi

while(True):
    clock.tick()
    img = sensor.snapshot()
    for tag in img.find_apriltags(fx=f_x, fy=f_y, cx=c_x, cy=c_y):
        img.draw_rectangle(tag.rect(), color = (255, 0, 0))
```

```
26          img.draw_cross(tag.cx(), tag.cy(), color = (0, 255, 0))
27          print_args = (tag.x_translation(), tag.y_translation(), tag.z_translation(),
                \
28              degrees(tag.x_rotation()), degrees(tag.y_rotation()), degrees(tag.
                    z_rotation()))
```

The obtained three-dimensional coordinates will be used to calculate the actual distance between the camera and the apriltag through the following distance formula:

$$D = \sqrt{D_x^2 + D_y^2 + D_z^2} \tag{4.1}$$

However, this calculated distance has distortion compared with the real distance. Thus we shoud correct this distortion. According to the manual of OpenMV, such distortion is approximately linear, determined by the scale coefficient k. Thus we can obtain k by testing several sets of data. The testing distance is set to be 200mm, and the three-dimensional coordinates of the Apriltag displayed by the terminals is $D_x = 0.56, D_y = 0.75, D_z = 3.58$. Thus k and $D_corrected$ can be calculated as:

$$k = 200/(0.56 * 0.56 + 0.75 * 0.75 + 3.58 * 3.58) = 54 D_corrected = k * D \tag{4.2}$$

Finally, the corrected distance will be displayed on the serial terminal. The code below shows the details of this part.

```
1  from math import sqrt #Introduce square root operation
2
3  def square(x): #Define the square operation
4      return x**2
5
6  k=54
7  distance = k*sqrt(square(tag.x_translation())+square(tag.y_translation())+square(tag.
       z_translation())) #The corrected distance from the camera to the center of
       apriltag
8  print("Tx: %f, Ty %f, Tz %f, Rx %f, Ry %f, Rz %f" % print_args)
9  print("Actual distance: %f" % distance)
10 print(clock.fps())
```

4.4.3 Testing results

Figure 4.9 shows the recognition and detection result of the Apriltag. It can be seen that the target Apriltag is successfully detectedin the frame in Figure 4.9.

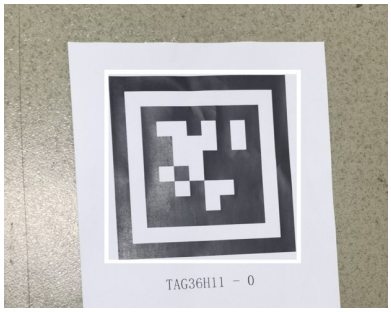


Figure 4.9: Apriltag recognition result.

4.5 Patrol

Main Contributor: Li Yuetai

In this part, the design of car patrolling is illustrated in detail. Firstly we choose real-time imaging method. Particularly, edge-color joint detection is used for the searching of target path, and PID is used for deviation controlling. finally, field test shows that we generally have a perfect car patrolling performance, which satisfies the task requirements.

4.5.1 Proposed Methods and Our Choice

The car patrol may have different solutions to meet the satisfaction, which are described and discussed as follow, respectively.

Method A: Infrared detection Infrared detection method is based on the principle that infrared rays have different reflection intensities on the surface of objects of different colors, and the car continuously emits infrared light to the ground during the driving process [16]. When the infrared light encounters the white paper floor, diffuse reflection occurs, and the reflected infrared light is received by the receiving tube mounted on the trolley; when the infrared light encounters the black line, the infrared light is absorbed, and the receiving tube on the trolley cannot receive the infrared light. Light. Therefore, the single-chip microcomputer determines the relative position of the black line according to whether it receives the reflected infrared light, and uses the feedback program to design the walking route of the car [14].

The advantages of this method include simple circuit design, fast information detection rate and low cost [14]. In addition, because the radiation of the natural environment in this band is significantly weak, the infrared reflective sensor is less affected by external interference and has high reliability.

However, this method has the non-negligible drawback, i.e., the fewer judgement parameters causes less accurate and only responds to roads with high black and white contrast [13].

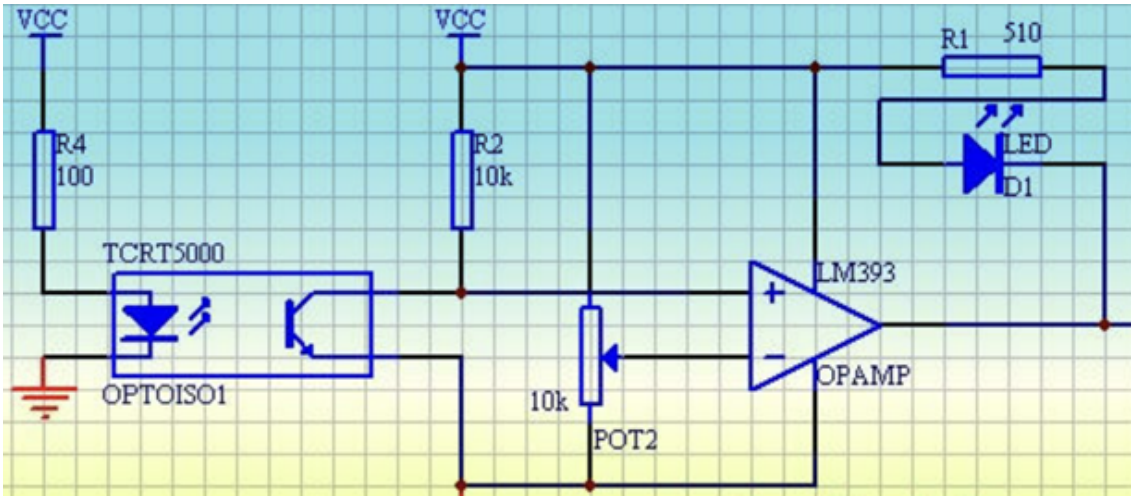


Figure 4.10: Circuit design diagram for infrared car patrol based on TCRT500 infrared transmitter and receiver (the LED part is designed to get percept of the feedback signal).

Method B: Real-time imaging systems Real-time imaging systems, such as OpenMV, require a dual power supply system to provide power to the single-chip microcomputer and the visual recognition module, then carry out imaging by an internal sensor based on visible light band [13]. The advantage of this system is that there are a large number of successful video recognition cases available for reference. Additionally, it has a simple structure, high sensitivity, and a high degree of integration, allowing to achieve various of functions. However, because it can only work in the visible light band, it is prone to interference from outside light.

Method C: 3D reconstruction of Lidar/mmWave Radar As the current state-of-the-art imaging technique, Lidar/mmWave Radar analyzes the object distance, spatial orientation, and relative speed information carried by the echoes, and using algorithms to draw the surrounding terrain map, regarded as an effective visual detection method for autonomous driving.

The advantage of this system is that the imaging field is wide and non-line-of-sight imaging can be performed, thus avoiding the catastrophic consequences that may occur when the line of sight is blocked. In addition, millimeter-wave imaging is stable and reliable in the natural environment, and is not prone to be interfered by signals in other bands. However, mmWave 3D reconstruction requires a lot of difficult technical assistance, is significantly hard to deploy, and is expensive far beyond the project budget [14].

Our choice: Considering the detection performance, scheme A is not sufficient for our task. For the project budget, since the recognition module costs is allowed only within 300rmb, we still use OpenMV and use the real-time image recognition method based on visible light. It is worth noting that if there is more funding and deployment time, the scheme of 3D reconstruction of Lidar/mmWave Radar might be the most attractive and interesting choice.

4.5.2 Initial Design: Patrol Based on Color Detection

Based on the imaging system, the car can obtain the real-time situation of the ground. Since the target patrol path is darker than other grounds, the initial idea is based on color recognition to identify the darker path in the field of view of the car [17]. The car then changes the travel speed and direction, and finally complete the patrol task. To be specific, we set the appropriate interval of the LAB value of the target color and then filter the other undesired color whose LAB value is outside our expected interval. Through the binaryzation, the desired color stands out in the image. The corresponding code is shown as below.

```
1  import sensor, image, time
2
3  green_threshold   = ( 0, 80, -70, -10, -0, 30)
4  #corresponding (minL, maxL, minA, maxA, minB, maxB) , while for grey images only (min, max) is necessary.
5  sensor.reset() # camera initialization
6  sensor.set_pixformat(sensor.RGB565)
7  sensor.set_framesize(sensor.QQVGA)
8  sensor.skip_frames(time = 2000)
9  sensor.set_auto_gain(False)
10 sensor.set_auto_whitebal(False)
11 clock = time.clock()
12 while(True):
13     clock.tick() # Track elapsed milliseconds between snapshots().
14     img = sensor.snapshot()
15     blobs = img.find_blobs([green_threshold])
16     #find_blobs(thresholds, invert=False, roi=Auto),thresholds is the color threshold.
17     if blobs:
18         for b in blobs:
19             # Draw a rect around the blob.
20             img.draw_rectangle(b[0:4]) # rect
21             img.draw_cross(b[5], b[6]) # cx, cy
22             print(clock.fps())
```

We provide a demo of the feasibility for the color recognition proposed in this section, which is shown in Figure 4.11. In this demo, green is taken as an example and the corresponding LAB threshold, i.e., line 3 of the code below, is set to be the threshold. Other colors in the image (non-green) is successfully filtered by our color recognition algorithm, while the target color (green) is completely and obviously retained.

4.5.3 Accuracy Improvement: Edge-Color Joint Detection

In this section, edge detection has been used as the supplement of color detection to improve the recognition accuracy of target path searching.

Firstly, we use three different edge-detection operators, i.e., Prewitt, Sobel and Canny operators to smooth the image and then filter the edge. To be specific, Prewitt and Sobel operators use the first-order differentiation as the index to judge whether one pixel is on the edge. The difference is that Sobel operator has larger weight in the central pixal thus the difference between edge and non-edge stands out more obviously. As for the Canny operator, the

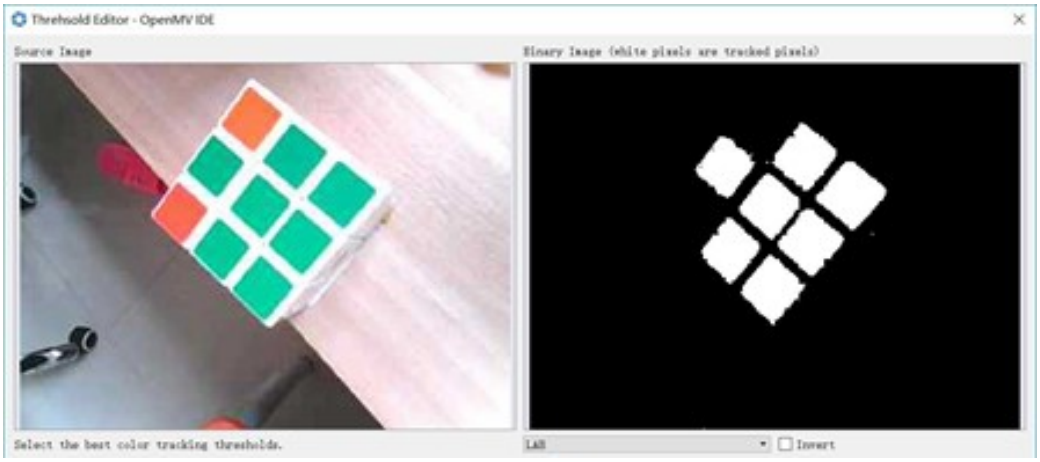


Figure 4.11: Demo for color recognition.

Gaussian de-noising filter is firstly used, and then first-order derivative is used. The largest difference is that Canny operator has two more stages, i.e., non-maximum suppression and double threshold mechanism.

Figure 4.12 shows the edge detection results. Although the Canny operator is mostly complex, the performance is not perfect as another two operators. As for Sobel and Prewitt operators, it can be seen that the target road is generally detected, and both the number of white points is suitable for path detecting. However, it is worth noting that the number of white points of Sobel operator processing is generally less than that of Prewitt operator processing. Thus it can be inferred that Sobel operator might have more perfect performance especially in presence of environmental interference. Thus we finally choose Sobel operator for both effectiveness and simple realization.

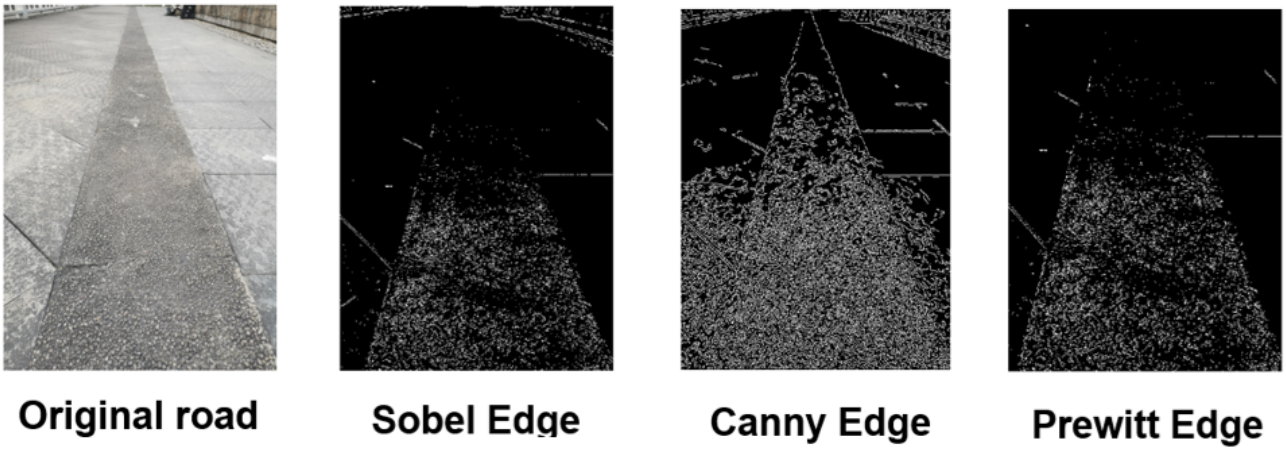


Figure 4.12: edge detection results for three different operators.

Finally, Figure 4.13 shows the main process of our Edge-Color Joint Path Searching algorithm. To be specific, both results of edge detection and color detection are combined by the operation of pixel bitwise, then a linear regression is carried out to illustrate the target path as [18] mentioned. It is worth noting that we use image processing of erosion and dilation in our procedure to reduce identification errors. The field test result for the joint detection is shown in section 4.5.5.

4.5.4 PID Control

According to Edge-Color Joint Detection, the target path can be found and the car can adjust the running direction and speed to follow this path. However, in a control system, the actual behaviour might always deviates from the required behaviour due to the system error [19]. Specifically, in the case of car patrol system, the main deviation is the error between the actual car travelling route and the ideal route obtained by image processing. One solution of this problem is the PID algorithm, which consists of a proportional unit (P), integral unit (I) and differential unit (D) with

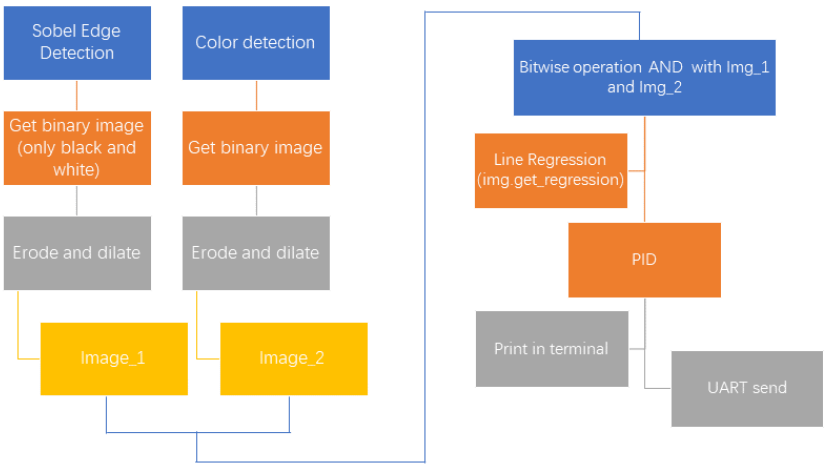


Figure 4.13: Diagram of joint detection.

corresponding adjustable gain $scale1$, $scale2$ and $scale3$ [20]. In particular, PID is suitable for the approximately linear system with time-invariant dynamic characteristics. These characteristics are exactly in consistent with the motion of our car patrol, thus we use PID to correct for our system offset.

In our system, the PID algorithm is used to calculate the rotational speed offset of the left and right wheels of the car, and then feedback to adjust the driving direction to make the target route and the actual route consistent. We can firstly obtain the error value of a certain period from the main function through the get_pid, and then calculated a reasonable offset through integral and differential operations to feed back to the main function, which is shown in Figure 4.14. Furthermore, the code of the revised PID class and our implementation is shown as below, which has been stored in the OpenMV.

```
from pyb import millis
from math import pi, isnan
class PID:
    _kp = _ki = _kd = _integrator = _imax = 0
    _last_error = _last_derivative = _last_t = 0
    _RC = 1/(2 * pi * 20)
    def initial(self, p=0, i=0, d=0, imax=0):
        self._kp = float(p)
        self._ki = float(i)
        self._kd = float(d)
        self._imax = abs(imax)
        self._last_derivative = float('nan')
    def get_pid(self, error, scaler):
        tnow = millis()
        dt = tnow - self._last_t
        output = 0
        if self._last_t == 0 or dt > 1000:
            dt = 0
            self.reset_I()
        self._last_t = tnow
        delta_time = float(dt) / float(1000)
        output += error * self._kp
```

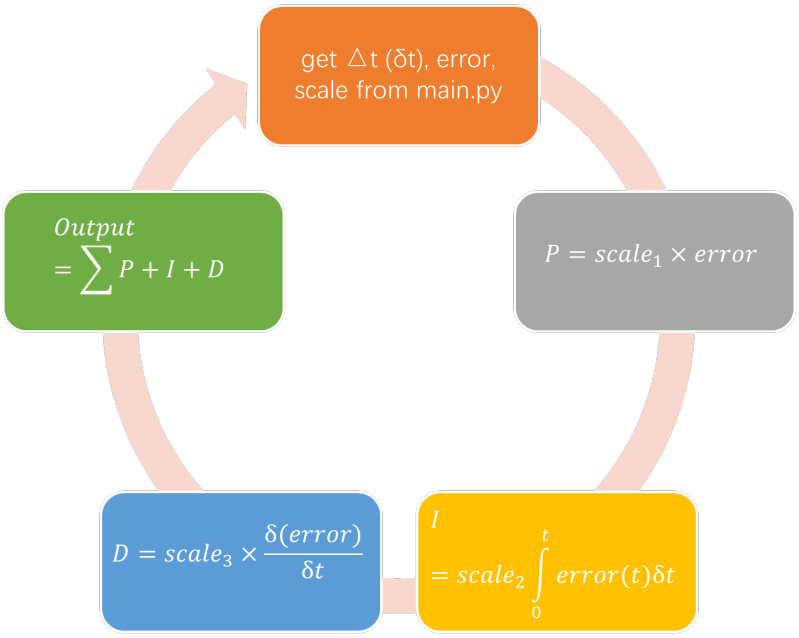


Figure 4.14: PID control implementation

```
        if abs(self._kd) > 0 and dt > 0:
            if isnan(self._last_derivative):
                derivative = 0
                self._last_derivative = 0
            else:
                derivative = (error - self._last_error) / delta_time
            derivative = self._last_derivative + ((delta_time / (self._RC +
                delta_time))     * (derivative - self._last_derivative))
            self._last_error = error
            self._last_derivative = derivative
            output += self._kd * derivative
        output *= scaler
        if abs(self._ki) > 0 and dt > 0:
            self._integrator += (error * self._ki) * scaler * delta_time
            if self._integrator < -self._imax: self._integrator = -self._imax
            elif self._integrator > self._imax: self._integrator = self._imax
            output += self._integrator
        return output
    def reset_I(self):
        self._integrator = 0
        self._last_derivative = float('nan')
```

Finally, let (rho, theta) represent the bias distance and angle of the target path at this time to the car, which is returned from the code as below. Thus the car can adjust its speed and running direction to follow the target path and finish the patrol task.

```
THRESHOLD = (5, 70, -23, 15, -57, 0)
import sensor, image, time
from pyb import LED
from pid import PID

rho_pid = PID(p=0.4, i=0)
theta_pid = PID(p=0.001, i=0)

```

```
LED(1).on()
LED(2).on()
LED(3).on()

sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QQQVGA)
sensor.skip_frames(time = 2000)
clock = time.clock()

while(True):
    clock.tick()
    img = sensor.snapshot().binary([THRESHOLD])
    line = img.get_regression([(100,100)], robust = True)

    if (line):
        rho_err = abs(line.rho())-img.width()/2
        if line.theta()>90:
            theta_err = line.theta()-180
        else:
            theta_err = line.theta()

        img.draw_line(line.line(), color = 127)
        print(rho_err,line.magnitude(),rho_err)
        if line.magnitude()>8:
            rho_output = rho_pid.get_pid(rho_err,1)
            theta_output = theta_pid.get_pid(theta_err,1)
            output = rho_output+theta_output
#car.run(50+output, 50-output)
#car.run(0,0) otherwise(fail to get enough clear line), stop
#car.run(50,-50) rotate on the spot to search in sight for lines pass

    print(clock.fps())
```

4.5.5 Field Test

We test the feasibility and performance of the edge-color joint path detection. As shown in Figure 4.15, the joint recognition is generally valid to find the target path. The linear regression is shown to find a line centered within the expected area (bright area in Figure 4.15), which demonstrates the car can generally obtain the target path. Although another area with scattered points produced by the light environment or human shadow interference also appears, this does not affect our target path searching.

4.6 Ultrasonic Module Connected to OpenMV

Main Contributor: He Yuechen

In this section, the ultrasonic module connected to OpenMV is designed to be served as an assistance of ranging to complete tasks in the Patio 2. Firstly we briefly introduce the US-015 Module and then describe our implementation in detail.

4.6.1 US-015 Module

Ultrasonic sensors are analog to another pair of eyes of the OpenMV camera. The obtained distance feedback is used to determine subsequent instructions. In this section, the design of the ultrasonic sensor US-015 module is illustrated in details. The purpose of designing this two ultrasonic sensors module is to trace along handrails in patio

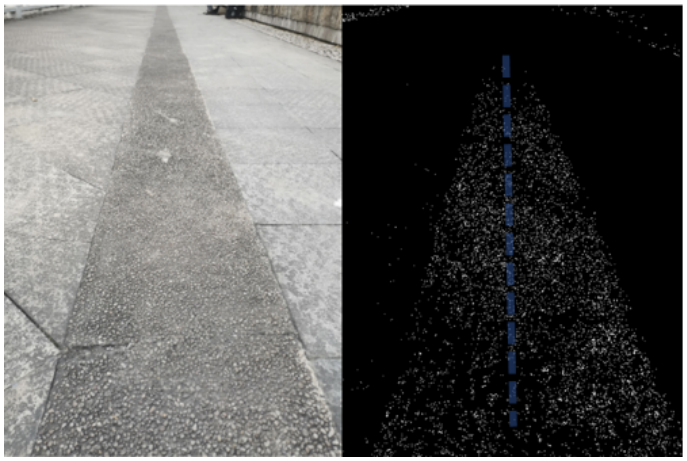
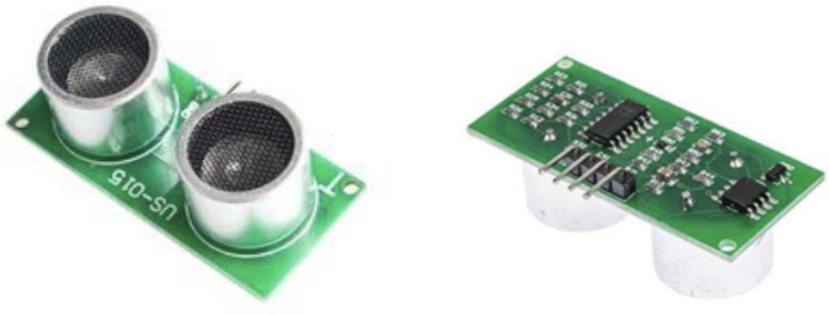


Figure 4.15: Searching the target patrol path

2. Since there are not enough I/O interfaces on the control board, we connected the two ultrasonic modules to the OpenMV module.

The ultrasonic sensor US-015 module has four pins, i.e., VCC, GND, ECHO, and TRIG, which is shown in Figure 4.16. In this project, VCC and GND pins of 2 sensors were connected to the power supply pin of OpenMV and ground of OpenMV respectively. TRIGs are connected to output pins P2s and P7 on OpenMV, and ECHOs are for input pins P3 and P8 refer to Code xx attached below.



(a) Front feature of US-015 (b) Back feature of US-015

Figure 4.16: Front and Back Feature of US-015

	Pin Symbol	Pin Function Description illustration
1	VCC	5V power suplly
2	Trig	Trigger input pin
3	Echo	Receiver output pin
4	GND	Power ground

Table 15: Pin Assignment

4.6.2 OpenMV Program Implementation

In this part we describe our implementation in detail.

As in Figure 4.17, the working flow of the ultrasonic sensor is illustrated. Initially, the Trig pin on the OpenMV triggers a minimum 10us High level pulse and sends it to the sensor. The sensor transmits an ultrasonic signal and receives from the echo. The Echo pin is connected with an input pin on OpenMV, which should be set to remain to

the High level until sensor receives reflected. As soon as a high-level voltage is received, a timer in OpenMV should be initialized to count the time. The high-level duration is the ultrasonic propagation time in the air, including the travelling and reflection time. The duration corresponds to the distance, which can be calculated as follows.

$$disstance = tim_counter * 0.017(cm) \tag{4.3}$$

where ultrasonic wave velocity in air is set to be $340m/s$. Thus, after capturing the period of high-level, the time is multiplied by 0.017 in our program, and the distance is in unit of centimeter.

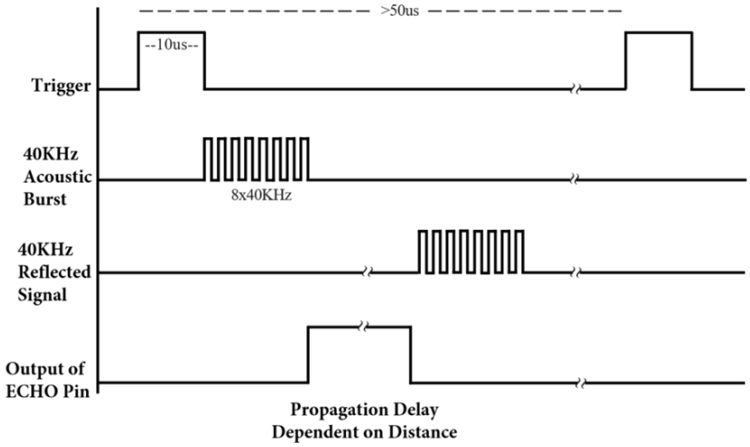


Figure 4.17: Summary of working principle for US-015 paraphrasing from datasheet.

More in detail, the program is packaged into a Class called "WAVE" in an individual file, which aims at conveniently callback in the main function of OpenMV. Consequently, whenever the class is instantiated, the pins for the ECHO and TRIG are defined and *wave_patrol* method is called in the main program, the method can be adopted to calculate the distance to the objects and calculate the right rotation speed for the car. Then OpenMV will send a message to STM32 to change the speed of left and right wheels (this will be shown in the code below). The TRIG pin is first set to "0" for 3 us, then "1" for 20 us, and finally "0", to trigger the sensor. Method *machine.time_pulse_us* is used to count the duration time of high-level "1", and it returns the time lasting in us unit. If the time cannot be counted, a sentence of "error" will be printed out in the serial output window for convenient debug process. Then the distance error will be calculated and transmitted to PID method to obtain a convergent transient distance error. Consequently, this error will be used to calculate the modulation of speed of wheel and send it to main control board with method *sending_data*.

The code below shows the implementation of the ultrasonic module.

```
1
2  import time,utime,pyb, machine
3  from pyb import Pin
4  from pid import PID
5  from pyb import UART
6  from pyb import LED
7  import math
8  import ustruct
9  class WAVE:
10     a=0
11     distance_initial=6      #unit cm
12     distance_pid = PID(p=0.9, i=0.2)
13     distance_6cmpid=PID(p=0.05,i=0)
14     scaler=5
15     _uart=UART(3,115200)
16 # Ultrasonic Sensor Distance Calculation
17 def wave_distance_process(self,wave_echo_pin,wave_trig_pin):
18     # Wave Start
```

```
    wave_trig_pin.value(0)
    utime.sleep_us(3)
    wave_trig_pin.value(1)
    utime.sleep_us(20)
    wave_trig_pin.value(0)

    # Receive echo signal
    try:
        tim_counter = machine.time_pulse_us(wave_echo_pin, 1, 30000)   # pin,
            timeout=30000us (30000*0.017=510cm)
    except:
        print('error',wave_echo_pin.value())

    # calculate distance
    wave_distance = tim_counter*0.017
    return wave_distance

# Ultrasonic Sensor Initialization
# Ultrasonic Sensor Pins
def sending_data(self,x,y):
    #global uart
    data=ustruct.pack("<bbbbbb",
            0xAA,
            0xFF,
            0xA1,
            int(x),
            int(y),
            0x0d)
    self._uart.write(data)
# Send data, uart.write data
def wave_patrol(self):
    while (True):
        while self.a%2==0:
            wave_echo_pin_1 = Pin('P1', Pin.IN, Pin.PULL_NONE)     # A0 P13, P7 D12,
            wave_trig_pin_1= Pin('P2', Pin.OUT_PP, Pin.PULL_DOWN)   # A1 P14, P8 D13,
            distance1=self.wave_distance_process(wave_echo_pin=wave_echo_pin_1,
                wave_trig_pin=wave_trig_pin_1)
            #print('distance1',distance1,'cm')
            self.a+=1
            #print('a1=',a)
        else:
            wave_echo_pin_2 = Pin('P7', Pin.IN, Pin.PULL_NONE)     # A0 P13, P7 D12,
            wave_trig_pin_2 = Pin('P8', Pin.OUT_PP, Pin.PULL_DOWN)   # A1 P14, P8 D13,
            distance2=self.wave_distance_process(wave_echo_pin=wave_echo_pin_2,
                wave_trig_pin=wave_trig_pin_2)
            #print('distance2',distance2,'cm')
            self.a+=1
            #print('a2=',a)
        distance_error=distance1-distance2
        distance_6cm=distance1+distance2-2*self.distance_initial
        #print('error=',distance_error)
        error_pid=self.distance_pid.get_pid(distance_error,1)
        drift_pid=self.distance_6cmpid.get_pid(distance_6cm,1)
        total_error=error_pid+drift_pid
        left=int(50+self.scaler*total_error)
        right=int(50-self.scaler*total_error)
        print('speed,left',left,'right',right)
        LED(1).on()
        self.sending_data(x=left,y=right)
        LED(1).off()
        time.sleep_ms(100)
```

```
77 def __init__(self,uart):
78     self._uart=uart
```

4.7 Communication between OpenMV and STM32

Main Contributor: He Yuechen

Inter-board Communication is significant in our smart robot system as it' s composed of two boards: The STM32F103RCT6 board is responsible for controlling the movement and giving orders while OpenMV is responsible for calculating and patrolling. In order to control the car to move according to OpenMV's commands, highly reliable inter-board communication needs to be realized. Therefore, some failures caused by noise from the friction between line and solder and wrong encoding or wrong decoding need to be avoid through a careful design of communication protocol.

Generally, UART was chosen as the communication protocol. Compared to other protocols like SPI or I2C, it has the outstanding advantage of fewer lines (only TX, RX and ground required, no clock needed). Although UART has the limitations such as slow speed,the speed it can provide is enough for our project as the data sending and receiving rate between both sides is just required to be more that 3000Hz, and Baud rate was set to be 115200.

In this section, we demonstrate our design for communications between OpenMV and STM32 in three part, including how OpenMV Sends data, how STM32 Receives data and how OpenMV sends orders to HC-12 and Clock module.

4.7.1 OpenMV Sending data

To avoid the error caused by noise and garbled code, parity bytes was added to check the validity of messages. The terrible confusing behaviors of the car caused by messages without parity bytes will be described in "Debug process of communication" . In addition, as shown in the code below, in order to realize packaged message sending, *ustruct.pack* method was utilized. The package includes 4 bytes parity header, 2 bytes mode selection message, 4 bytes sending message and 2 ending parity bytes.

Ustruct class can pack and unpack primitive data types. *Ustruct.pack* (fmt, v1, v2, ...) method can Pack the values v1, v2, ···according to the format string fmt. The return value is a bytes object encoding the values. Additionally, P4 and P5 are set to be used for UART communication (P4: TX; P5: RX) on OpenMV. Therefore, the pin on STM32 for TX should be connected to P5 and pin for RX should be connected to P4.

See the code below for details of the sending part.

```
1  def sending_data(self,x,y):
2      #global uart
3      data=ustruct.pack("<bbbbbb", # The format is two characters, two short integers (2 bytes)
4              0xAA, # Frame header 1
5              0xFF, # Frame header 1
6              0xA1, #mode byte
7              int(x),
8              int(y),
9              0xfe) # Last frame oxfe
10     self._uart.write(data)
```

4.7.2 STM32 Receiving data

Similar to sending data pack, the data received from STM32 inclueds 3 parts: 4 bytes parity header (AA and FF), 2 bytes mode selection message (0x01-0x05). As shown in the line 31 of code below, openmv reads the data from RX port for uart1 (P5), then store one message into a stack. This message will be validated to have an "AA FF 0x" structure, and the third byte of it will then be extracted into a variable– *ctrl.work_mode*. The method to examine

the parity bits and extract mode bits is called *change_mod()*. Next, according to the mode bits extracted, loops in main function will instantiate different objects according to self-written class, call different methods, and finish different tasks in different modes.

From the line 45 to line 64 in the code below, we could identify that the correspondence between codes and openmv behaviors. The relationship is shown in the table below.

Code	Class File / Method	Function in Tasks
01	PATROL	Patio 1. Patrol
02	SHAPE	Patio 2. Shape classification
03	WAVE	Patio 2. Ultrasonic rail patrolling/ Patio 1. Gate Finding and End to stop
04	APRILTAG	Patio 2. Can Identification
05	HC-12	Patio 2. Send Messages

Table 16: Relationship between codes and openmv behaviors

```
# main.py  Main function

import sensor, image, time,pyb, lcd
from pyb import LED
from pyb import UART
from pid import PID
from patrol import PATROL
from shape import SHAPE
from wave import WAVE
from apriltag import APRILTAG
import math
#import struct
sensor.reset()
#sensor.set_vflip(True)
#sensor.set_hmirror(True)
sensor.set_pixformat(sensor.RGB565) #set to be the grey image
sensor.set_framesize(sensor.QQVGA)
sensor.set_auto_gain(False)
sensor.set_auto_whitebal(False)
#sensor.set_windowing([0,20,80,40])
#sensor.skip_frames(time = 2000)
uart1=UART(3,115200)
uart2=UART(1,115200)
clock = time.clock()                    # to process a frame sometimes.
led = pyb.LED(2)
clock = time.clock()                    # to process a frame sometimes.
class ctrl(object):
    work_mode = 0x00
ctrl=ctrl()
def change_mod():
    if(uart.any()):
        txt=uart.read()
        if(len(txt)>=6):
            if(txt[0]==0xAA):
                if(txt[1]==0xff):
                    ctrl.work_mode=txt[4]

uart1 = UART(3,115200)
uart_b= UART(1,115200)
while(True):
    clock.tick()
    img = sensor.snapshot() #.binary([THRESHOLD])
    #img.lens_corr(1.2)
    change_mod()
    if (ctrl.work_mode==0x00): #MODE1 waiting
```

```
        True
    if (ctrl.work_mode==0x01):#MODE1 start patrolling
        car_patrol=PATROL(dec_bit=1,uart=uart1)
        retu_val=car_patrol.patrol_line()
        print('the distance is: %d',retu_val)
    if (ctrl.work_mode==0x02):#MODE2 shape recognition
        car_shape=SHAPE(uart=uart1)
        retu_val=car_shape.shape_rec()
        print ('shape is ',retu_val)
    if (ctrl.work_mode==0x03):#MODE3 ultrasonic module
        car_wave=WAVE(uart=uart1)
        retu_val=car_wave.wave_patrol()
        print ('wave is ',retu_val)
    if (ctrl.work_mode==0x04):#MODE4 Apriltag recognition
        car_april=APRILTAG(uart=uart1)
        retu_val=car_april.apriltag_det()
        print ('april is ',retu_val)
    else
        True
```

4.7.3 Sending orders to HC-12 Clock module from OpenMV

Another STM32 single chip microcomputer was designed to control our clock module and HC-12 signal sending module due to lack of ports in main control panel. In previous section, it is illustrated that HC-12 mode has only one command– *uart2.sending_data(1,1)*. It should be mentioned that the UART serial port called in this line of code is different from that of previous. Uart2 is used, which was connected to the STM32 that controls HC-12 module.

In OpenMV, it is set that P0 and P1 can also be used for UART communication (P0: RX; P1: TX). In our project, we agreed that the communication format with hc-12 control board is: AA FF BB 01 01 FE. When receiving this message, the controller will let HC-12 send messages as required.

5 Wireless Communication to Complete tasks in Patio 2

Main Contributor: Ma Yibo

Patio 2 requires that when the car travels to the preassigned area, the microprocessor will send a wireless communication message containing the team number, team member names, and time of the day to the computer in real-time. In this wireless communication module, we plan to use Wavesens HC-12 wireless transceiver to complete data transmission and reception at 433MHz, and DS1302 clock module to obtain clock time. Three key sub-tasks are listed below.

(1). Achieve the data transmission between OpenMV and microprocessor with UART.

(2). Implement the Wavesens HC-12 communication module to achieve data transmission.

(3). Obtain Beijing time in real-time.

Their theoretical bases and the work details are described in detial in the following sections.

5.1 Data Transmission

A. Background information on UART

In UART communication, only two wires are used to transmit data between two modules. The UART communication allows the serial form transmission of data. After receiving the serial data, the receiver converts the serial data back into parallel data. Data flows from the TX pin of the transmitting UART to the RX pin of the receiving UART, which can be explained by Figure 5.1 [21]。

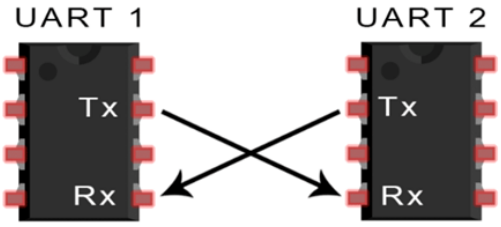


Figure 5.1: Data Transmission of UART

UARTs transmit data asynchronously, which means there is no clock signal to synchronize the output of bits. During the transmission of data, the transmitting UART uses start and stop bits to the data packet instead of a clock signal. Therefore, the receiving UART can know when to start reading the bits and when to stop based on the start bit and stop bit [22]. In summary, UART transmitted data is organized into packets. Each packet contains 1 start bit, 5 to 9 data bits (depending on the UART), an optional parity bit, and 1 or 2 stop bits, which is shown in Figure 5.2.

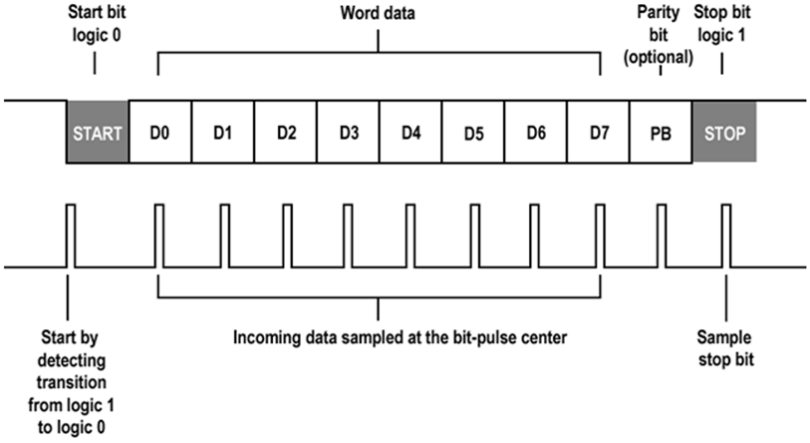


Figure 5.2: Form of the Data in the UART Transmission

B. Wireless Communication between Microcontroller and OpenMV

When the vehicle arrives at the preassigned area, OpenMV will send a message **'Oxaa Oxff Oxbb'** to the microcontroller. Then the microcontroller will read the current time from the DS1302 Clock Module and control the HC-12 module to send out the message containing the time and team information.

5.2 HC-12 Wireless Communication

A. Overview

According to the task requirement, wireless communication should be established between the laptop and the vehicle we design. A large amount of data including team introduction and the real-time should be transmitted rapidly with no error. HC-12 is an applicable module to achieve these tasks.

HC-12 is a half-duplex 20 dBm (100mW) wireless serial communication module with 100 channels in the 433.4-473.0 MHz range. Paired with an external antenna, these two transceivers are capable of wireless communication with transmission slightly beyond 1 km in the open area. The experimental setup and its representative block diagram are displayed in Figure 5.3.

The maximum operating current is 100mA. There are five pins on HC-12 module in total. Two pins (VCC and GND) are used to achieve voltage control. The other two pins (TX and RX) are UART communication ports to exchange data between HC-12 and another module. The "SET" pin is used to reset the settings of HC-12. According to the datasheet, HC-12 has a total of 4 serial port transmission modes FU1, FU2, FU3, FU4, corresponding to

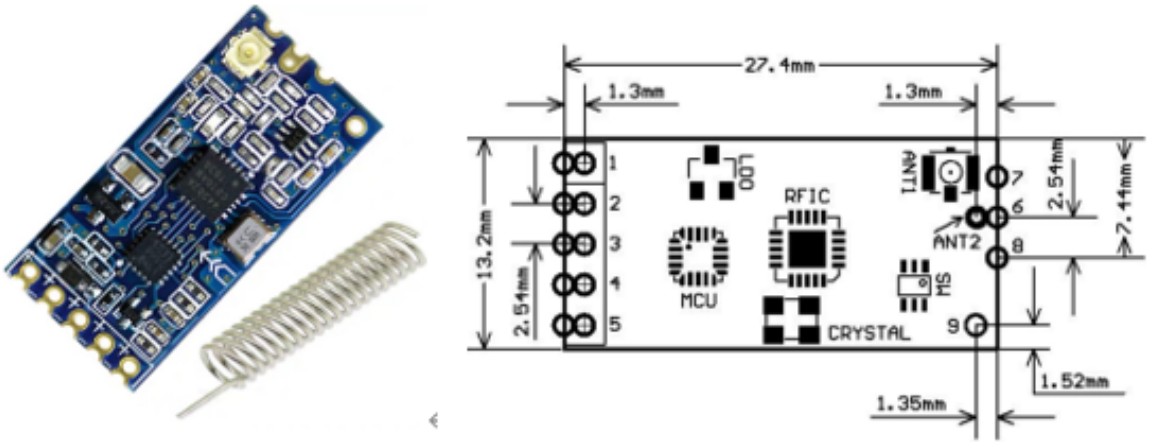


Figure 5.3: Experimental setup (left) and the representative block diagram (right).

different serial port baud rates.

B. Design Process

As for the designing and wiring, the TX pin of HC-12 is connected to the RX pin of the microcontroller. At the same time, the RX pin of HC-12 is connected to the TX pin of the microcontroller. Additionally, these two modules should be set to the same ground reference. The overall wiring diagram is displayed in Figure 5.4.

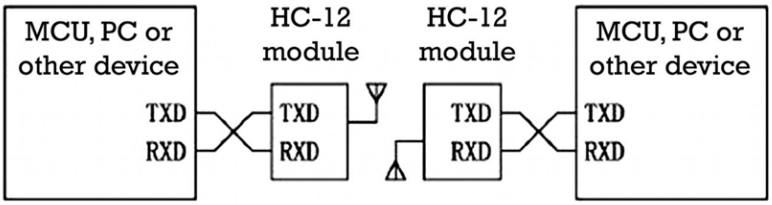


Figure 5.4: Wiring Diagram of HC-12

After testing, a low baud rate is suitable for long communication distances, but it will also lead to lower transmission efficiency and less transmission capacity in a short time. Since this task requires HC-12 to transmit a relatively large amount of data in a short period with a communication distance of about 10m, we finally choose the high baud rate transmission. The module settings of HC-12 are FU3, 9600bps (8-bit data, no parity, 1-bit stop bit), and CH001 (433.4MHz). The final experimental setup is displayed in Fig. 5.5.

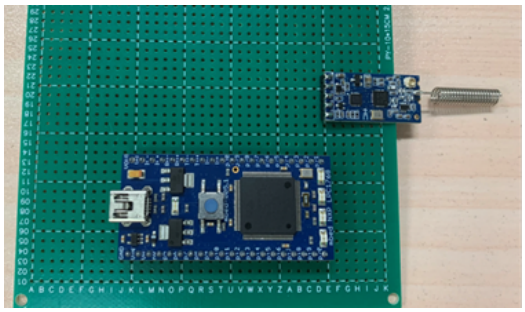


Figure 5.5: Physical Setup between HC-12 and Microcontroller

C. Programming Implementation

After the physical wiring of the HC-12 module and micro-controller, we can easily control the HC-12 module to send messages from the rover to our laptop. The codes below show the details.

```
#Initialization of HC-12 settings
void setup() {
```

```
  Serial.begin(9600);               // Serial port to computer
  HC12.begin(9600);                 // Serial port to HC12
}

//Sending and Receiving Message
#include "mbed.h"
#include "DS1302.h"

Serial pc(USBTX, USBRX);
Serial hc12(D4, D3);        //D4 for TX, D3 for RX
int main()
{
    while (HC12.available()) {          // If HC-12 has data
    Serial.write(HC12.read());        // Send the data to Serial monitor
  }
    while(Serial.available()){          // recieving data from vehicle
    DS1302_SubMain();
    hc12.baud(9600);
    hc12.format(8, SerialBase::None, 1);
    hc12.printf("Team 10: Forza Horizon\n\r");
    hc12.printf("Team Members:\n\r Li Muquan; Zhang Shikun; Liu Zhe; He Yuechen; Ma
        Yibo;\n\r Li Yuetai; Lin Meihong; Feng Guoyu; Liu Zheng; Shui Jiajun;\n\r");
    hc12.printf("Time of the day: %x:%x:%x\n\r", hour, minute, second);
    //Get the time from DS1302
    hc12.printf("Date of the day: 20%x-%x-%x \n\r", year, month, day);
    //Get the day from DS1302

    switch(week){
        case 1: hc12.printf("Monday \n\r");break;
        case 2: hc12.printf("Tuesday \n\r");break;
        case 3: hc12.printf("Wednesday \n\r");break;
        case 4: hc12.printf("Thursday \n\r");break;
        case 5: hc12.printf("Friday \n\r");break;
        case 6: hc12.printf("Saturday \n\r");break;
        case 7: hc12.printf("Sunday \n\r");break;
        }
```

5.3 Clock Module

A. DS1302 Clock Module Overview

According to the task requirement, the transmission message should include the real-time of the day. DS1302 is an applicable module to obtain and transmit the real-time for further wireless communications.

DS1302 Charge Timekeeping module contains a real-time clock and 31 bytes of static RAM, ensuring that the time could be stored in the register for further use. The experimental setup and its pins assignment diagrams are displayed in Figure 5.6.

The communication between DS1302 and the microcontroller is achieved by a simple serial interface. There are 8 pins on the module, but only 5 pins are supposed to be connected. (1) and (2): "VCC" and "GND" pins are connected to supply and control the voltage of the module. (3) "RST" is the reset/chip select line. All data transfers are started by driving the RST input to a high level[5]. Data transmission can only be performed when the level is high. (4) "DAT" is a two-way communication pin to achieve data transmission. (5) "CLK" is the clock pin, ensuring the overall time synchronization of the entire 8 pins. Data can be transferred to and from the clock/RAM 1 byte at a time or in a burst of up to 31 bytes.

The real-time clock provides seconds, minutes, hours, day, date, month, and year information. The inside AM/PM indicator helps the clock operate in either the 24–hour or 12–hour format. Its power consumption is designed to be

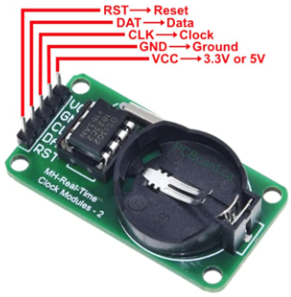


Figure 5.6: Experimental Setup of DS1302

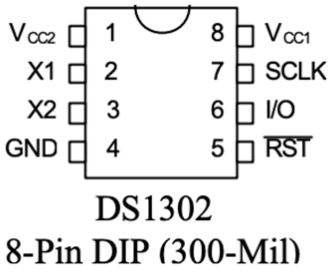


Figure 5.7: Pins Assignments of DS1302

very low (less than 1 mW). With the power supply from an external button battery, it can keep timing and retain data information for a long time.

B. Design Process

As for the designing and wiring, the "CLK", "DAT" and "RST" pins of DS1302 are connected to the PWM pins on the microcontroller respectively. Additionally, the two modules should be set to the same ground reference. The overall wiring diagram is displayed in Figure 5.8. The register style of the DS1302 module is displayed in Figure 5.9.

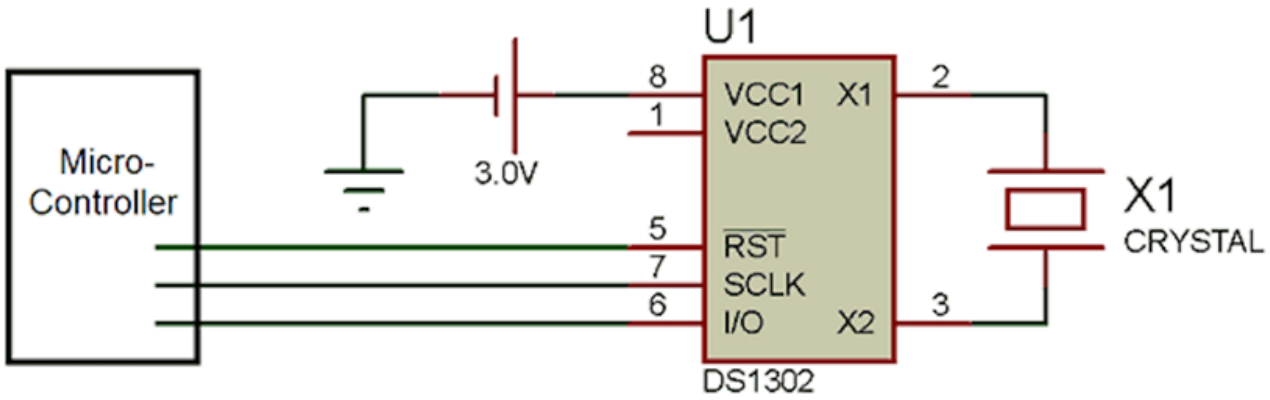


Figure 5.8: Block Diagram of DS1302

From this, we see that the 5th to 1st bits are the address of RAM or register, controlling the "read" or "write" function. The sixth bit, 1 means RAM, addressing internal memory address; 0 means CK, addressing internal register.

The addresses of reading and writing clock data of the DS1302 module are demonstrated in Figure 5.10. To control the module, we need to write or read data to these addresses. For example,

(1). the address for reading is: 0x81 (second), 0x83 (minute), 0x85 (hour), 0x87 (day), 0x89 (month), 0x8b (week), 0x8d (year);

(2). the address for writing is: 0x80 (second), 0x82 (minute), 0x84 (hour), 0x86 (day), 0x88 (month), 0x8a (week), 0x8c (year).

ADDRESS/COMMAND BYTE

7	6	5	4	3	2	1	0
1	RAM / $\overline{CK}$	A4	A3	A2	A1	A0	RD / $\overline{W}$

Figure 5.9: Register Style of DS1302

READ	WRITE	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	RANGE
81h	80h	CH	10 Seconds			Seconds				00–59
83h	82h		10 Minutes			Minutes				00–59
85h	84h	12/$\overline{24}$	0	10 $\overline{AM}$/PM	Hour	Hour				1–12/0–23
87h	86h	0	0	10 Date		Date				1–31
89h	88h	0	0	0	10 Month	Month				1–12
8Bh	8Ah	0	0	0	0	0	Day			1–7
8Dh	8Ch	10 Year				Year				00–99
8Fh	8Eh	WP	0	0	0	0	0	0	0	—
91h	90h	TCS	TCS	TCS	TCS	DS	DS	RS	RS	—

Figure 5.10: Addresses of Reading and Writing Clock Data

The final experimental setup is displayed in Fig. 5.11.

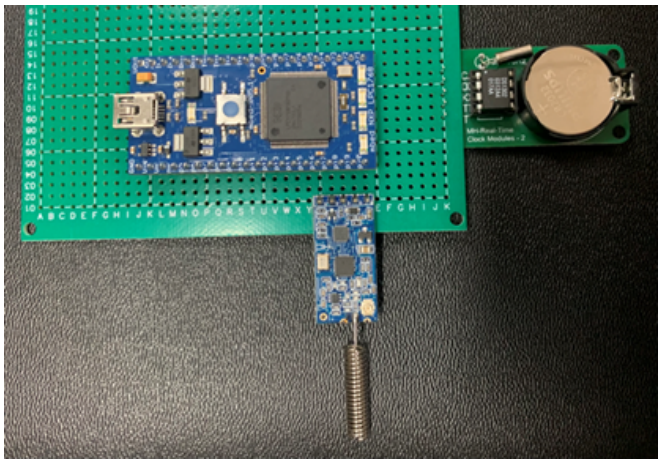


Figure 5.11: Physical Setup between DS1302 and Microcontroller

C. Programming Implementation

After wiring it up, firstly we need to initialize its current time based on the register address above, so that the DS1302 could start to operate. Then the DS1302 clock will run according to the setting. Therefore, we can use the microcontroller to read out the data in its register. The codes below show the details.

```
#include "mbed.h"
#include "DS1302.h"

DigitalOut sclk(D5); //clk pin
DigitalInOut io(D6); //data pin
DigitalOut ce(D7); //reset pin

uint8_t time[7]={0x58, 0x35, 0x20, 0x04, 0x06, 0x06, 0x22}; //Initialize the time of
    DS1302.
```

```
uint8_t write[7]={0x80, 0x82, 0x84, 0x86, 0x88, 0x8a, 0x8c};//The address of wirting
    time
uint8_t read[7]={0x81, 0x83, 0x85, 0x87, 0x89, 0x8b, 0x8d};//The address of reading
    time
uint8_t second, minute, hour, day, month, week, year;

void writetime(uint8_t dat)
{
    uint8_t m, value;
    sclk=0;
    wait_us(20);
    io.output();
    for(m=0;m<7;m++)
    {
        value=dat&0x01; //select the first bit of data and write it into DS1302
        if(value)
        io=1;
        else
        io=0;
        dat>>=1;//right shift 1 bit of dat
    }
}

//Read time from DS1302
unsigned char readtime(unsigned char Clo)
{
    unsigned char i,Data=0x00;
    ce=1;
    for(i=0;i<8;i++)
    {
        io=Clo&(0x01<<i);//Here, the first for loop is used to send the first bit in
            the Command by shifting, and so on
        sclk=0;
        sclk=1;
    }
    for(i=0;i<8;i++)
    {
        sclk=1;
        sclk=0;
        if(io){Data|=(0x01<<i);}//Here, if IO=1, then the i-th bit corresponding to
            Data will be 1 if IO=1, and if IO=0, then the i-th bit corresponding to
            Data will be 0 if it does not enter if.
    }
    ce=0;    //turn off CE in the end
    io=0;    //Set IO to 0 after reading, otherwise the read data will be wrong
    return Data;
}
```

5.4 Wireless Communication Result

Figure 5.12 shows the final outcome of the wireless communication. The team information details and time of the day are displayed on the port screen respectively.

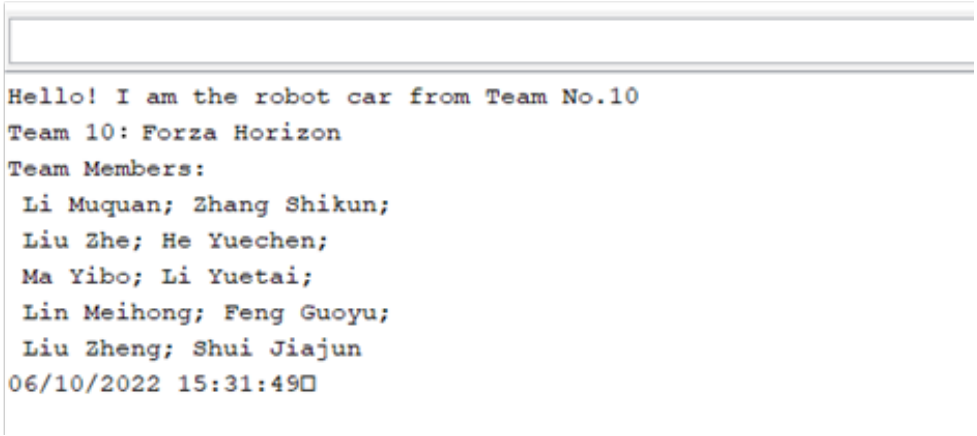


Figure 5.12: Wireless Communication Result

6 Conclusion

In this report, we have designed, implemented and tested an intelligent robot car to complete all the required tasks. The experimental results demonstrated the feasibility of our design. All the design are generally divided into two parts, i.e., car control in section 3 and image recognition in section 4, while the wireless communication in section 5 is considered as the supplement for the tasks in Patio 2.

In the Car Control Group, we initially decided the STM32F103RCT6 being the main control of our car, and based on this, the circuits and PCB are designed correspondingly. To withstand the road vibration and achieve the height of the basket, we chose a four-layer car structure with lengthened cooper pillars. As for the peripherals, we use Hall encoders, a gyroscope and a normal type of wheels to satisfy the dynamic control requirements of the car, which can generate a larger torque and better control the car movement respectively. To achieve ultrasonic distance detecting, we use an ultrasonic sensor HC-SR04, while to lift and shoot the tennis ball, a mechanical arm is designed and a servo is used to control the rotating angle. In order to better control the car, an OLED screen is implemented and other components, such as a key, a buzz beeper are used to assist mode selection and parameters display.

As for the image recognition group, we firstly determine OpenMv as the core architecture in the image part, and choose the ultrasonic US-015 module as a powerful assistance for ranging. Then the shape recognition, basket recognition and Apriltag recognition are implemented based on OpenMv and the US-015, respectively. Particularly, according to the traits of the object to be recognized and the practical environment, we select the corresponding appropriate methods for these three recognition tasks: using the proportion of the color block to the minimum circumscribed rectangle area, combining color recognition and ultrasonic ranging, and identifying the TAG36H11 family in Apriltag. As for the patrolling part, we use the edge-color joint detection method to search the target path, and then use PID algorithm to alleviate the deviations and maintain the tracking of target path when the robot car is running. Finally, the connection and communication between OpenMv and STM32 is implemented through UART protocol. How OpenMV sends data and how STM32 receives data are designed and discussed in detail.

As the supplement part, wireless communication is implemented in section 5 to complete the required tasks in Patio 2. Functions are successfully implemented including data transmission between OpenMV and microprocessor, Wavesens HC-12 communication module, and obtaining Beijing clock in real-time.

The test result has been appended in each part after design process, which shows our work generally achieve the design requirements. Additionally, some future improvements are mentioned in each section. It is worth noting that in section 4, the idea for one alternative of OpenMV is implementing mm-Wave system for sensing and recognition. This scheme might have significantly stability, higher accuracy and more flexibility, which can be considered as the future challenging with sufficient funding and time.

References

[1] STMicroelectronics. *STM32F103xC, STM32F103xD, STM32F103xE*. https://www.st.com/resource/en/datasheet/cd00191185.pdf.

[2] TOSHIBA Corporation. *TB6612FNG*. https://www.mouser.com/datasheet/2/408/TB6612FNG_datasheet_en_20141001-708260.pdf.

[3] *Comparison Of Several Mainstream Pcb Design Software*. https://www.rocket-pcb.com/comparison-of-several-mainstream-pcb-design-software.

[4] *Manual vs. Automated Assembly: When Does Your PCB Need a Human Touch? - VSE*. https://www.vse.com/blog/2019/06/27/manual-vs-automated-assembly-when-does-your-pcb-need-a-human-touch/.

[5] Arduino Corporation. *Arduino Documentation*. https://docs.arduino.cc/.

[6] *Micro Servo Motor Tower Pro MG90s Metal Gear - Microdaz Blog*.

[7] *Zhongling Technology Servo User Manual.pdf-original document*. https://max.book118.com/html/2021/0222/5234030202003131.shtm.

[8] *Working with STM32 and Displays: SSD1306 I2C OLED display - EmbeddedExpertIO*. https://embeddedexpert.io/?p=613.

[9] *In-Depth: Interface MPU6050 Accelerometer and Gyroscope Sensor with Arduino*. https://lastminuteengineers.com/mpu6050-accel-gyro-arduino-tutorial/.

[10] *STM32CubeMXHall encoder, L298N drive motor_W_oilpicture blog-CSDN blog_stm32 Hall encoder*. https://blog.csdn.net/WandZ123/article/details/124583452.

[11] Admin. *What's the Mecanum wheel and how to use it?* en-US. Oct. 2019.

[12] Singtown. *Openmv embedded image processing*. https://book.openmv.cc/.

[13] Graeme J Awcock and Ray Thomas. *Applied image processing*. Macmillan International Higher Education, 1995.

[14] Maria MP Petrou and Costas Petrou. *Image processing: the fundamentals*. John Wiley & Sons, 2010.

[15] Singtown. *Openmv embedded image processing*. https://docs.singtown.com/micropython/zh/latest/openmvcam/index.html.

[16] Gloria Bueno Garca et al. *Learning image processing with OpenCV*. Packt Publishing Birmingham, 2015.

[17] Kiryung Lee, Dong Sik Kim, and Taejeong Kim. "Regression-based prediction for blocking artifact reduction in JPEG-compressed images". In: *IEEE Transactions on Image Processing* 14.1 (2004), pp. 36–48.

[18] Mauricio Marengoni and Denise Stringhini. "High level computer vision using opencv". In: *2011 24th SIBGRAPI Conference on Graphics, Patterns, and Images Tutorials*. IEEE. 2011, pp. 11–24.

[19] M Araki. "PID control". In: *Control Systems, Robotics and Automation: System Analysis and Control: Classical Approaches II* (2009), pp. 58–79.

[20] Mituhiko Araki and Hidefumi Taguchi. "Two-degree-of-freedom PID controllers". In: *International Journal of Control, Automation, and Systems* 1.4 (2003), pp. 401–411.

[21] AB Wilkinson. "An introduction to microcomputers. Vol 1 —Basic concepts and Vol 2 —Some real products. 8080 programming for logic design: A Osborne. Adam Osborne Associates (obtainable from Sybex Publications Department, Paris) (1976) (2nd ed.), 289 pp, 368 pp and 273 pp, respectively, $7.50 each (paperback)$". eng. In: *Microprocessors* 1.5 (1977), pp. 339–339. ISSN: 0308-5953.

[22] Konstantin Mikhaylov et al. "Impact of IEEE 802.15.4 Communication Settings on Performance in Asynchronous Two Way UWB Ranging". eng. In: *International journal of wireless information networks* 24.2 (2017), pp. 124–139. ISSN: 1068-9605.