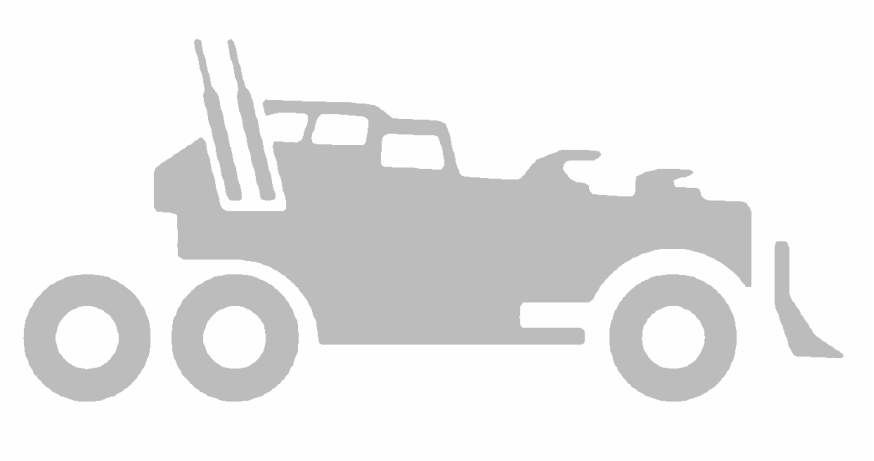
电子科技大学
格拉斯哥学院
Glasgow College, UESTC

Team Design Project and Skills

2020-21, semester II

Final Report



Team: Mad Max (No.37)
Supervisor: Dr Wasim Ahmad

Abstract

With the continuous deepening of robotics research, a new discipline, robotics has emerged, which relates to a lot of technologies including artificially intelligence, controlling, etc. In the course, we are required to design an intelligent car which is capable of performing tasks including line tracking, moving across the bridge, passing through the gate, color detection and item releasing. This report illustrates our smart rover with functions of line tracking, color recognition, distance measurement, object delivery, inter-board communication, dynamic self-adaptive direction control and turning on the spot. The electrical hierarchy of this vehicle consists of STM32 and OpenMV together with their corresponding peripherals. The experiment details present the idea and design process of how these two microcontrollers cooperate with each other to fulfill the assignment. OpenMV is applied for image processing and send instructions to other parts of our car, while STM32 is used to receive the instructions from OpenMV and adjust the modes of the robot car. There are mainly 7 modes of the car movement: stop, receiving PWM control from OpenMV, turning on the spot, turning while moving, moving in a straight line on pebble ground, moving in a straight line across the bridge and operating the robotic arm. More precise movement and better performances can be presented on all the tasks of both patios accordingly with the 7 controlling modes. Under a limited budget, the robotic car achieves the expected requirements and performs unexpectedly well.

Table of Contents

List of Figures and Tables

Acknowledgements

Thanks for the guidance and help from team supervisor Dr Wasim Ahmad as well as the financial support from UESTC Glasgow College for this project.

Thanks to the seller of the components for the technical support.

Last but not the least, thanks to all team members. We have been working together for over 3 months since the team was formed on March 3rd. It was a very solid group, and each member took his or her task very seriously and responsibly. We have suffered setbacks. We burned out 16 circuit boards by different reasons. However, there is no complain or blame, but only everyone united in the laboratory together to troubleshoot problems until midnight. We have experienced success. We were all happy and proud when our little car was perfectly tracing and passed through the gate in Patio 1. Three month's shared delights and common hardships has enhanced our friendship. I'm so sure that this TDPS project will be an unforgettable memory of our university life.

1 Introduction

1.1 Background

Team Design Project and Skills (TDPS) is one of the Year-3 courses aiming to enhance engineering capabilities as well as gain teamwork techniques. In this course, 10 team members are asked to design and assemble a robot car which is able to finish given tasks on 2 different patios. At the end of the course, students need to give a final demonstration of their robotic car in the field in a competition.

1.1.1 Design Tasks Overview

Based on the location of the field, different tasks are divided into two patios. And students are asked to design a robotic car which can complete all tasks of the two patios.

Patio 1 has three tasks, where the first task is to recognize and follow the path composed of dark floor tiles, the second task is to find the bridge and cross on top of it and the third task is to find the gate, go through and finally stop. Figure 1 shows the overview of patio 1.

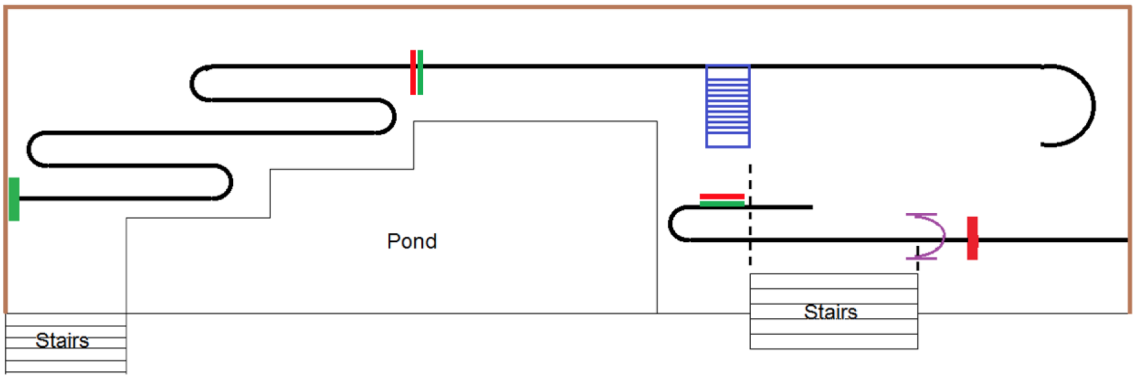


Figure 1 Patio 1

Patio 2 also has three main tasks. The mission of task 1 is to detect a particular color and automatically move in a specific direction. Task 2 is to automatically move from the red line in task 1 to a position near the lake and release an item to the lake. Task 3 requires car to return back to planter area and transmit a message to a laptop. After receiving reply message from the laptop, the car needs to keep moving until the destination. The overview of patio 2 is shown below in Figure 2.

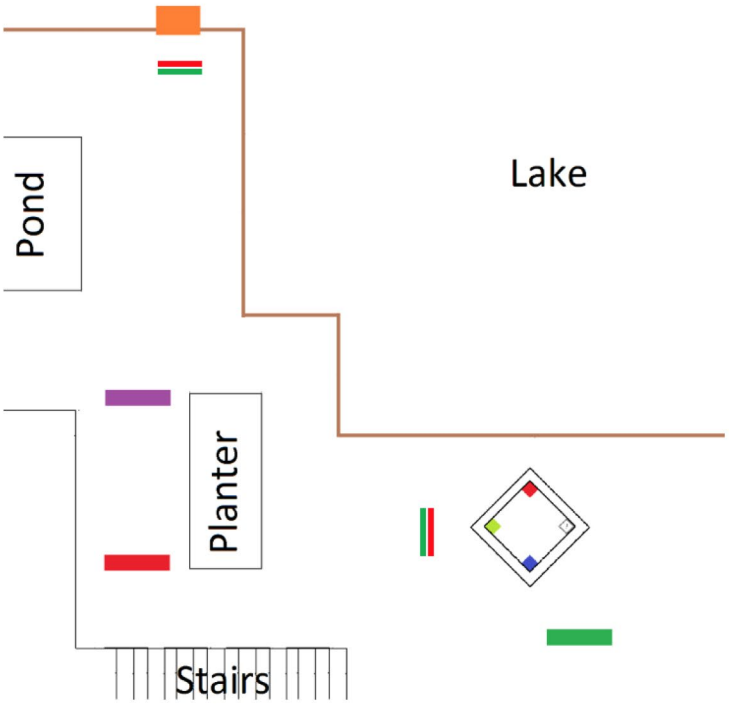


Figure 2 Patio 2

1.1.2 Team Information

Established on 3 March 2021, our team consists of 10 students from both CE and IE. To achieve high efficiency in the project, our group is divided into two sub-teams: **Car Group** and **Image Group**. The work of the Car Group is to control the car movement, covering the work of assembling the car, wiring, driving circuit design and manufacturing, PID algorithm, etc. The Image group is responsible for tracing, color and beacon recognition, communication and control of the claws. Each group has a coordinator who is responsible for coordinating the work within the group and communicating with the other group. The team leader is responsible for coordinating the overall project progress. The team information as well as rough responsibilities of team members can be found in Table 1, where The table in blue is for image group members, and the table in green is for car group members. Figure 3 show the group members.

Team Number		**37**				
Team Name		**Mad Max**				
No	**UESTC ID**	**UoG ID**	**Name**	**Role/Responsibility**	**Degree program**	**Gender**
1	2018190606034	2429391X	Zhangchen Xu	**Team Leader** + Communication	CE	Male

2	2018190502024	2429497X	Ziyi Xie	**Image Group Coordinator** + Route Tracing	IE	Male
3	2018190607005	2429400Y	Shubo Yang	Image: Tracing + Ultrasonic Sensors	CE	Female
4	2018190505002	2429592Y	Jingluan Yang	Image: Color Recognition	IE	Female
5	2018190505005	2429595H	Qinlin Han	Image: Detecting Railing+Route Planning	IE	Female
6	2018190505006	2429596L	Xinyue Li	Car: Wiring+Claw	IE	Female
7	2018190602001	2429207Y	Yuchen Yao	Car: PID Algorithm	CE	Female
8	2018190606004	2429361Z	Weizhe Zhao	Car: L298 Driver Design + Gyroscope	CE	Female
9	2018190502030	2429503L	Ziyang Long	**Car Group Coordinator** + PCB design	IE	Male
10	2018190505036	2429626F	Chenyang Fan	Car: Speed Feedback	IE	Male

Table 1 Team Information

Figure 3 Group Members

1.1.3 Technical Background

Microcontroller is the brain of the car which controls the car's speed, direction and robotic arm. A microcontroller is an integrated circuit used for controlling other portions of an electronic system, usually via a microprocessor unit (MPU), memory, and some peripherals [1]. The selection of the microcontroller should consider the task requirements (e.g., power, memory, costs, number of pins). In this project, we choose STM32F411 [2] as the main microcontroller. As one of the models of High Performance F4 Series, STM32F411 offer the best balance of dynamic power consumption and processing performance. Normally, we use C language to control the microcontroller. Although C language is powerful and mature in embedded systems, it is difficult to use. So, we choose **microPython** as the programming language. MicroPython is a stripped-down version of Python that is designed to run on microcontrollers with limited performance, with a minimum requirement of only 16K memory [3]. We can use microPython to develop applications such as automation and robotics very easily.

Image recognition is essential to complete the tracking process in Patio 1 and color recognition in Patio 2. In the past, implementing image recognition in low-power embedded systems was a pipe dream. However, nowadays, with the evolution of technology, powerful development boards such as the Raspberry Pi [4], or FPGAs [5] can easily do this. In this task, we used the OpenMV development board [6] for image processing due to the limitations of the requirements. OpenMV is a machine vision platform that includes a camera and a powerful STM32 microcontroller. It runs MicroPython and includes an application programming interface (API) with various machine vision functions that make interpreting images much easier.

The robotic car’s movement is powered by four DC electric motors. Driving the motors requires high voltages and currents that cannot be provided by a microcontroller [7]. Therefore, a **motor driver** is needed. The purpose of the motor drivers is to use control signals with low current and convert it into a higher current signal that can drive a motor. In order to have a complete control over DC motor, one should control both speed and rotation direction. This can be done by two techniques: PWM for controlling the speed and H-Bridge for controlling the rotation direction. In the project, we choose L298N [8], a dual-channel H-Bridge motor driver which is capable of driving a pair of DC motors. We designed a circuit which have two L298N and 4 outputs, so 4 motors can be driven using only one driver board.

The direction and distance feedback from gyroscope and ultrasonic distance sensors is also important in the project. The direction feedback helps to make accurate turns to achieve a stable and correct journey for the car, and the distance feedback can be used with beacon to guide the cart to turn. A **gyroscope** is a device used for measuring or maintaining orientation and angular velocity. Commonly used Gyroscopes in embedded systems such as MPU-6050 can return six-axis data, while more powerful ones (such as the JY901S) can return nine-axis data including

geomagnetic data. The **ultrasonic distance sensor** is an electronic device that measures the distance by emitting ultrasonic sound waves and converts the reflected sound waves into an electrical signal. A widely used HC-SR04 ultrasonic module can provides 2cm to 400cm of non-contact measurement functionality with a ranging accuracy reach up to 3mm [9].

1.2 Scope and Limitations

The scope of this report is the tasks in two Patios of the TDPS course. Experiments and tests have proven that our robotic car is able to complete each task very well.

However, there are still limitations to our design and production. One of the biggest limitations is that our algorithm is too sensitive to weather conditions. Both the algorithms of patio 1 and patio 2 need to collect thresholds during testing to get better results. Otherwise, there is a high probability that good results will not be achieved. Let's take tracing algorithm in Patio 1 as an example. If the edge detection threshold of a sunny day is used in a cloudy day, the algorithm may not able to calculate the center of the path, thus the car is likely to be off route.

The second limitation is the stability of the system. In our tests, the microcontroller occasionally gets burned out. Some of these times were human mistakes (such as incorrect wiring), but there are some cases that cannot be explained by human error. We can only attribute these situations to occasional failures.

Another limitation is that human intervention may be required to complete all tasks. For both Patio 1 and Patio 2, we may need one intervention to get the car to complete all tasks successfully. This is probably because the robustness of our system is not strong enough. In the competition on June 20th, we also did an intervention in Patio 2 and ended up with 29 points.

2 Overall System Design Approach

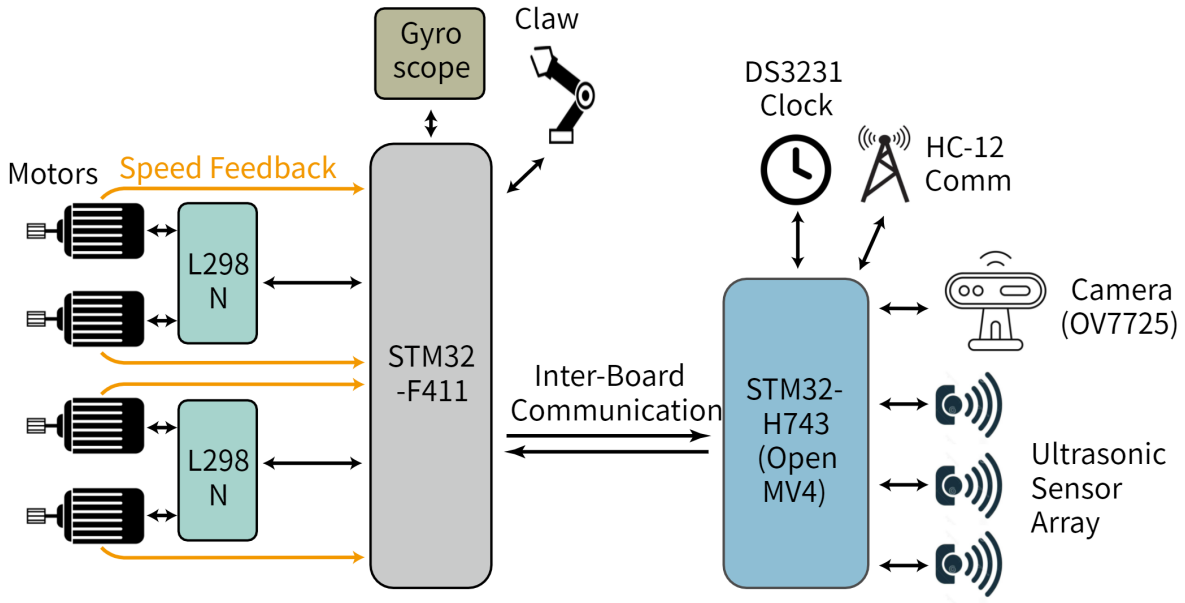


Figure 4 Overall System Structure

Figure 4 illustrates the overall system structure of our robotic car. As shown in the figure, we used two major boards control the car. One is a STM32F411 development board which is responsible for controlling the car movement, adjusting car's position and operating a robotic arm for feeding. Motors with Hall velocity sensor and two L298N driver modules are connected with this board, and there is also a gyroscope module which provides position for PID algorithm. We can regard these modules together as a motion controller.

The other board is a STM32H743 development board with OpenMV4 firmware, aiming at providing operation instructions to the system, and it's like a direction controller (or a wheel) that send direction order to the car. The OpenMV4 is connected to an OV7725 camera and processes image data for tracing and color detection. Furthermore, it collects distance information via ultrasonic sensor array for synthetical decision-making and is responsible for communication with laptops via a HC-12 communication module. There is also a DS3231 clock module to save time under power off conditions. In terms of inter-board communication, we use UART, an asynchronous serial communication protocol.

In terms of software, we choose **microPython** as project's embedded development language, since it is a lean and efficient implementation of the Python 3 programming language that includes a small subset of the Python standard library and is optimized to run on microcontrollers as well as in constrained environments.

3 Subsystem Design and Solutions

3.1 Car Group: Car Structure Design

Section Author	**Chenyang Fan** **(UESTC ID:2018190505036 UofG ID:2429626F)**
Technically Assisted by	Xinyue Li (UESTC ID:2018190505006, UofG ID:2429596L)

3.1.1 Part1: Design and Construction of Car Structure

Whether the structure of the car is stable or not is an important factor for the smooth movement of the car. Therefore, in the process of building the structure of the car, We chose the appropriate screws, two solid acrylic plates to place the image module and the hardware module respectively, and four flexible universal wheels.

In terms of drive scheme of the robot car, we evaluated two-wheel drive and three-wheel (with one omni-directional wheel), and finally chose **4WD (four-wheel drive)** scheme. There are three main reasons for the choice. First, as the car is expected to drive across the cobblestone pavement in Patio 2, 4WD is preferred for its outstanding off-roading performance, with better traction on rough roads than its 2WD counterparts. Second, 4WD cars are able to make sharp turns, since both the front and rear wheels can be designed to take part in turning. Moreover, 4WD is switchable to 2WD when necessary. 4WD scheme has some shortcomings, though, such as higher cost and energy consumption. However, the cons are almost immaterial when compared with the pros. Consequently, we decided to follow the 4WD scheme.

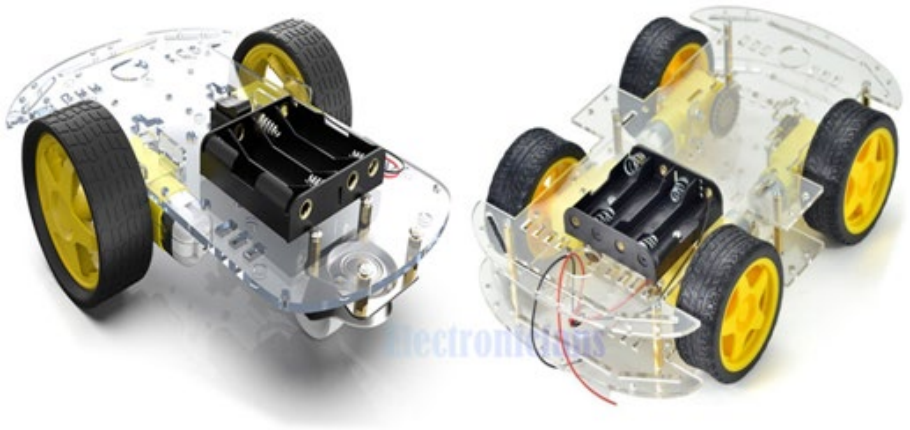


Figure 5 Comparison of 4WD and 2WD

Initially, I need to get the screws, screwdrivers, wheels, motor shaft and some other components ready to build the car.

Figure 6 The parts needed to build the car

The following step is using M3x12 screws and M3 nuts to connect the motor bracket and the acrylic base plate. Then, I utilized M3x4 screws to connect the motor and the motor bracket (the motor shaft is connected at the end far away from the acrylic base plate). After that, M3x6 screws were used to connect the motor and the coupling while I need to ensure that the screws is aligned with the cut surface of the motor shaft Finally, M4x6 screws and M4 washers were utilized to connect the wheels and couplings, and the frame of a car with four-wheel drive has been installed.

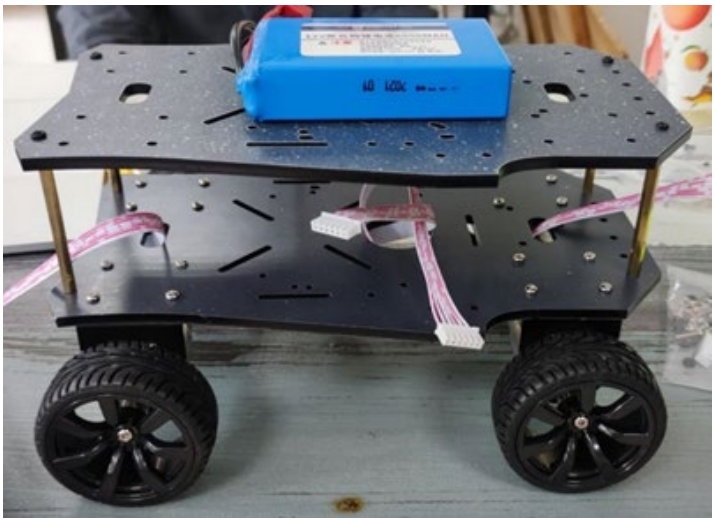


Figure 7 The rudiment of car before wiring

3.1.2 Part2: Wiring between Different Modules

As for wiring between each module and STM32, our aim is to enable each module operate normally. After checking the function of each pin of STM32 and the user manual of each module, we connected these modules to the pin and draw the schematic diagram for use when writing code.

Initially, we refer to the graph of the micro control unit in order to understand the function of each pin. Then we look through the user manual of each module to comprehend how to connect modules in a correct way.

There are mainly five modules, which are Driver module, Motor module, Motion Processor and Power module.

As for the Driver module, we combined 2 basic driver circuit in one driver module to control 4 motors. The Figure 4 shows the structure of the driver module.

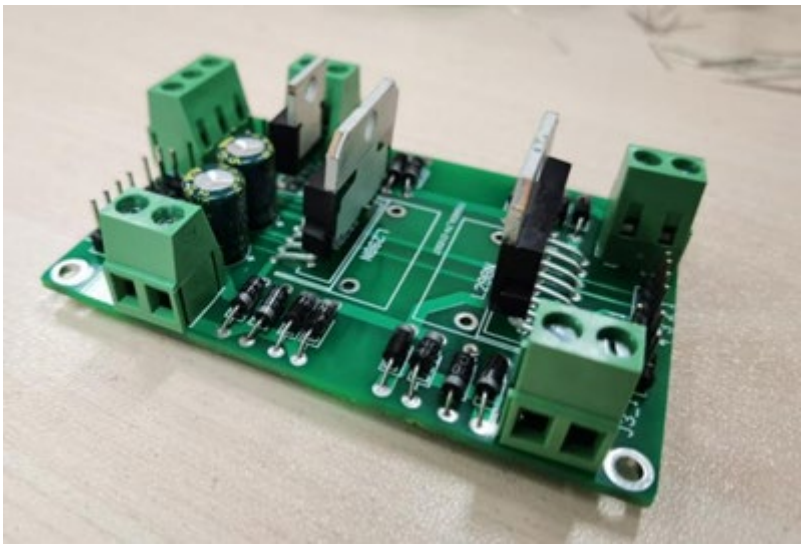


Figure 8 Driver module board

The Vss is connected to the voltage supply Vcc 12 volts and the Vs is connected to the digital signal 5 volts. There are 2 groups of OU pins, OU1 to OU4 are used to input digital signal series which control the direction of current output to the two motors (M1 M2), the other group is used to control the other motors (M3, M4). Besides, the ENA and ENB pins are used to control the speed of wheels by adjusting the PWM of input. The Figure 6 illustrates how the driver module is connected to other parts.

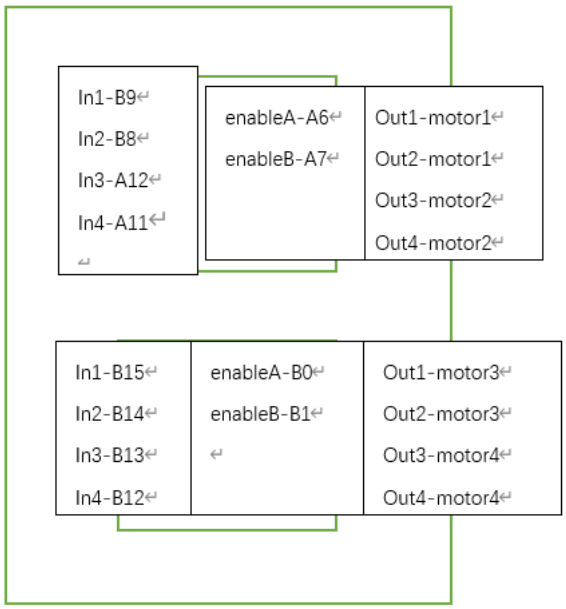


Figure 9 The wiring diagram of driver module

Next, we considered linking JY901 to the design.

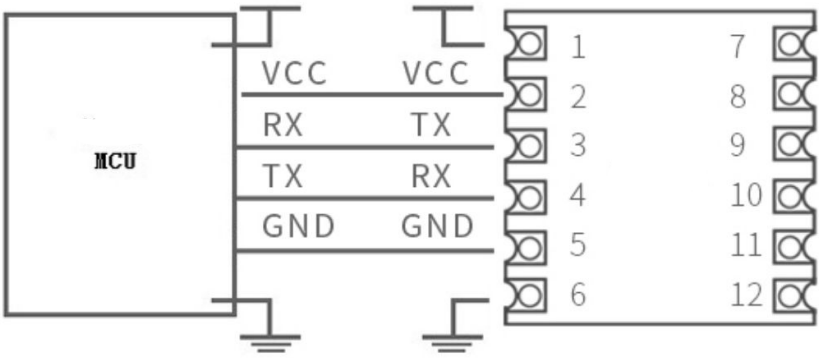


Figure 10 JY901

Then, we connected the motion processing module.

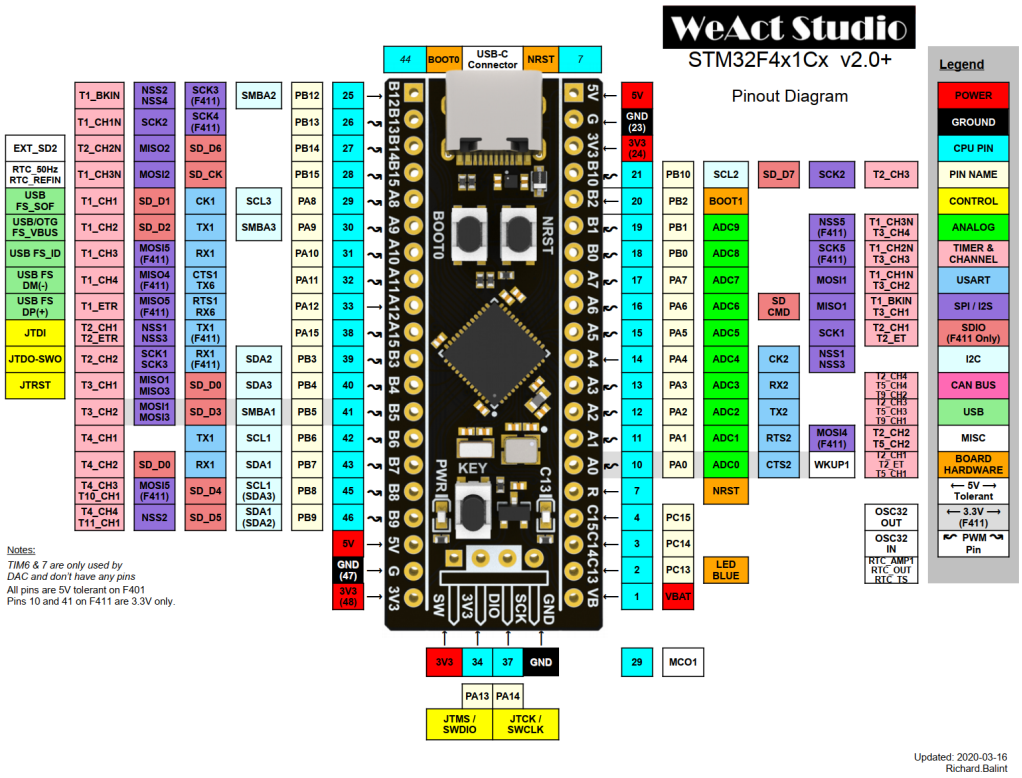


Figure 11 I/O of MCU

After finishing the above work, the rudiment of the car was formed.

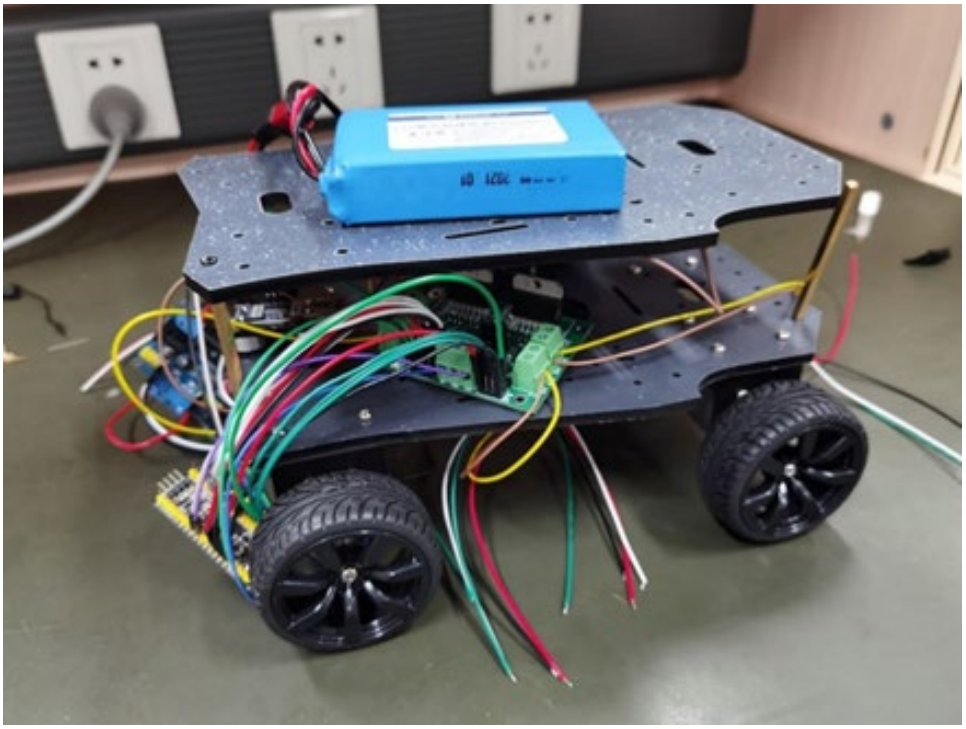


Figure 12 The rudiment of car

3.2 Car Group: L298N Driver Module Design

Section Author **Weizhe Zhao**
(UESTC ID: 2018190606004, UofG ID: 2429361Z)

3.2.1 Introduction to L298

As the power of a microcontroller is too small to drive motors, we design a driver module with chip L298, a dual full-bridge driver which can provide 2 DC motors with directional and speed control. Pin connections and functions of L298 are demonstrated in Figure 13 and Table 2.

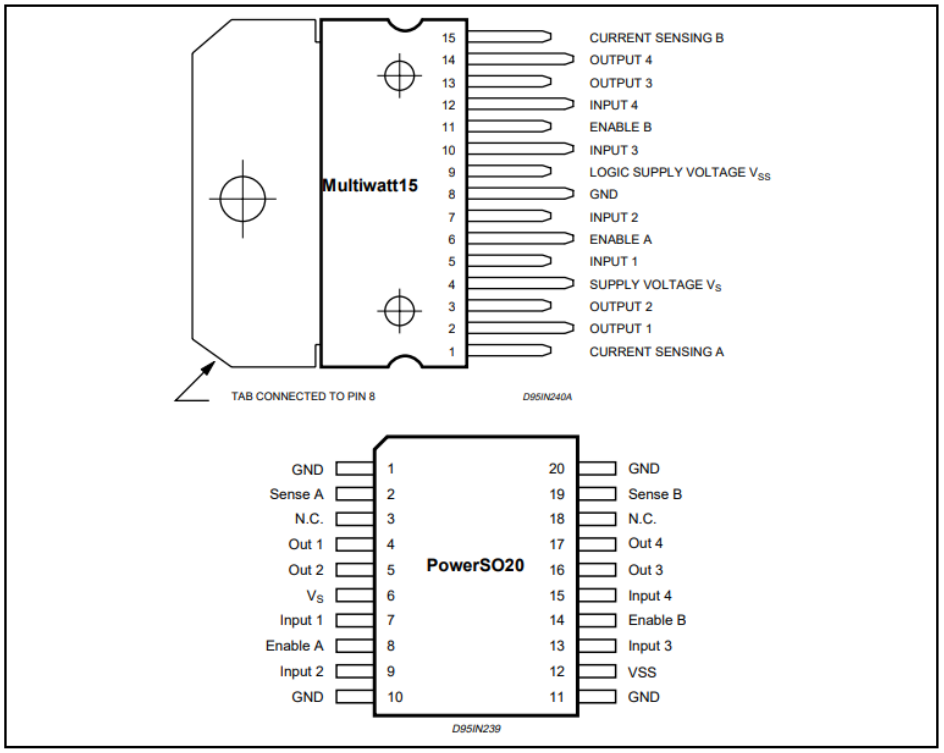


Figure 13 Pin Connections of L298

Pin name	Description
VSS	12V input from DC power source
VS	Supplies power for the switching logic circuitry inside L298N IC
GND	Ground pin
IN1 & IN2	Motor A input pins. Used to control the spinning direction of Motor A
IN3 & IN4	Motor B input pins. Used to control the spinning direction of Motor B
ENA	Enables PWM signal for Motor A

ENB	Enables PWM signal for Motor B
OUT1 & OUT2	Output pins of Motor A
OUT3 & OUT4	Output pins of Motor B

Table 2 Pin Functions of L298

3.2.2 Driver Module PCB Design

In our 4 wheel-drive smart car, two L298 chips are required, so I design the driver module accordingly. The driver module consists of two L298 chips, one LM7805 Voltage Regulator, some capacitors and a bunch of diodes (Figure 14).

IC LM7805 is used to restrict the power input of L298 logic circuitry to exact +5 V. If there were no such voltage regulator attached to the power supply, the logical output would be unstable, thus affecting driver performance.

When a motor is powered then disconnected, there would be a voltage inverse to the power due to self-induction of coils inside the motor, which can also generate an inverse current flowing through the circuit. The inverse current is large enough to destroy the microprocessor. Considering this risk, I employ some diodes at driver module inputs to insulate STM32 from the induction current. A couple of capacitors are also applied to absorb the current, thus preventing it from destroying internal circuit of the driver module.

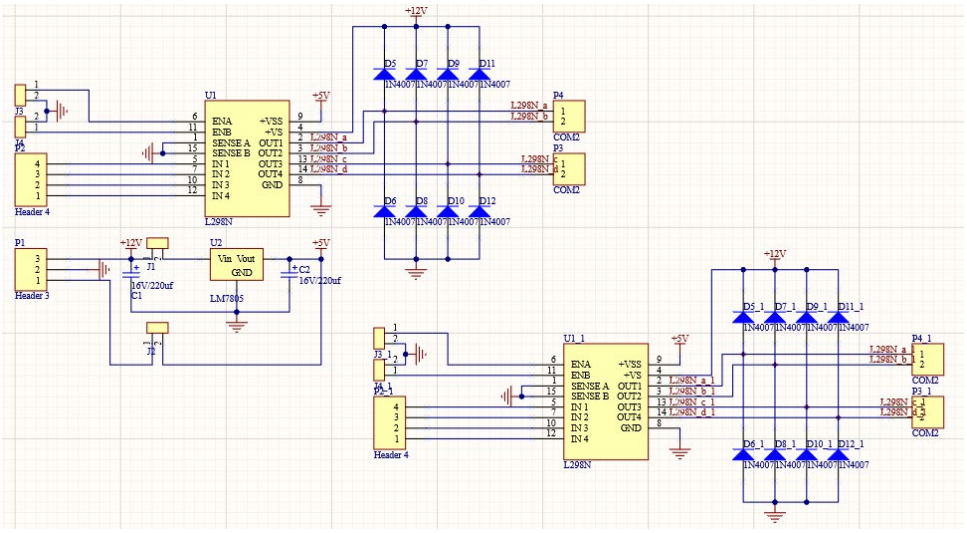


Figure 14 Driver Module Schematic Plot

Using Altium Designer, we designed and drew PCB layout based on the schematic plot. The packages of the devices are downloaded from official website and then connected. By making power and ground traces wider than average, I prevent burning wires as such traces have more current flowing through them. The final PCB design is shown in Figure 15.

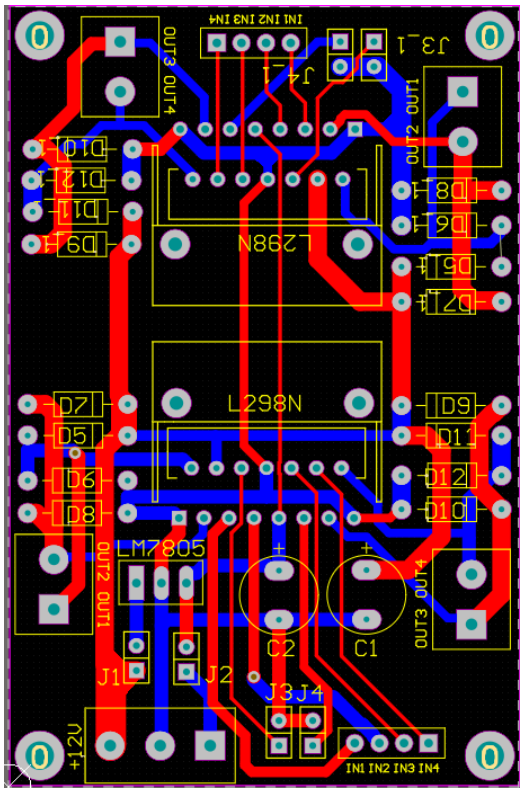


Figure 15 Driver Module PCB layout

3.2.3 Software Implementation of Driver Module

The implementation of driver module can be divided into two parts: logical inputs to control movement patterns of wheels, and pulse width modulation (PWM) to regulate the speed.

In terms of logical inputs, each motor is related to three signal outputs from the driver, defining how the motor is set to act. The logical relationship for one couple of motors (one L298 chip) is demonstrated in Table 3, where “0” or “1” for EN pin stands for low or high.

EN A/B	IN1/IN3	IN2/IN4	Motor A/B
0	X	X	Stop
1	1	0	Clockwise
1	0	1	Anticlockwise
1	1	1	Brake
1	0	0	Stop

Table 3 Logical Patterns of Driver Module

Meanwhile, PWM is given by corresponding “EN” pins. Through field tests, we figure out the speed of the wheel is basically linearly proportional to the pulse width percent is PWM.

Below is a code sample for the software implementation to control one of the wheels:

```
A1 = Pin('B3',Pin.OUT_PP)
A2 = Pin('B5',Pin.OUT_PP)
PWMA = Pin('B8') # PB8 has TIM4, CH3
tim = Timer(4, freq=1000)
ch1 = tim.channel(3, Timer.PWM, pin=PWMA)
ch1.pulse_width_percent(abs(self.NrA1-self.NrA2))
```

Code 1 STM32 Pin Configuration for Driver

Two pins of STM32 are configured as output mode to control the corresponding "IN" pins of driver module. Another pin is set as PWM output to modulate "EN", whose timer is enabled.

3.3 Car Group: Gyroscope

Section Author	**Ziyang Long** **(UESTC ID:2018190502030, UofG ID:2429503L)** **Yuchen Yao** **(UESTC ID:2018190602001, UofG ID:2429207Y)**

3.3.1 The Choice between MPU-6050 and JY901S

There are two candidates in our choice of Gyroscope, namely MPU-6050 in Figure 16 and JY901S in Figure 17.

Figure 16 MPU-6050

The MPU-6050 devices combine a 3-axis gyroscope and a 3-axis accelerometer on the same silicon die, together with an onboard Digital Motion Processor, which processes complex 6-axis motion fusion algorithms. For us, we mainly use three angular velocities 'Gyro' around three axis 'x, y, z', where the unit is °/s. After obtaining the angular velocity, a formula is necessary to change the angular velocity into angle, and the easiest way is:

$$angle = angular\ velocity \times time$$

The process of obtaining angle can be regarded as integral calculation when adding the angular velocity per unit time together. The code below presents the main logic of calculation angle.

```
sum=0
from time import sleep
while 1:
    sum=sum+MPU.read.Gyro_z()*0.002
    if(sum>=90 or sum<=-90):
        print('Sum is ',sum)
        sum=0
        sleep(0.5)
    sleep(0.001)
```

Code 2 integral of angular velocity

Although MPU6050 performs well (the excellent ability to capture the change of position) when rapidly rotating 90 degrees, it might lead to large error if the slight deflection occurs in unit time. On the condition that the small deflection occurs, the angular velocity will be unexpected small, and due to the limitation of MPU-6050's sampling rate, (maximum value is 8KHz), some unwanted omission may happen during the accumulation of the angular velocity per unit time

```
def reset(self):
        self._write_byte(MPU_PWR_MGMT1_REG, 0x00) # Configure the
power management register openMPU6050
        self._write_byte(MPU_GYRO_CFG_REG,  config_gyro_range<<3) #
gyro sensor,?2000dps
        self._write_byte(MPU_ACCEL_CFG_REG, config_accel_range<<3)#
acceleration sensor ,?2g
        self._write_byte(MPU_SAMPLE_RATE_REG,0x01)#The     sampling
frequency >512
        self._write_byte(MPU_CFG_REG,0x00)#Set the digital low pass
filter to the first mode and the output frequency is 8KHz
        self._write_byte(MPU_INT_EN_REG,0X00) #Close all interrupts
        self._write_byte(MPU_USER_CTRL_REG,0X00) #I2C main mode off
        self._write_byte(MPU_FIFO_EN_REG,0X00) #close FIFO
        self._write_byte(MPU_INTBP_CFG_REG,0X80) #INT Pin low level
valid

        buf = self._read_byte(MPU_DEVICE_ID_REG)
        if buf != self._address:
            print("MPU6050 not found!")
        else:
            pass
```

Code 3 initialization in MPU6050

Considering the defects of MPU6050 mentioned before, we decided to give up using MPU-6050.

✓ Angle Range:±180°
✓ Stability:0.05°/s
✓ Protocol: UART

Figure 17 JY901

3.3.2 Two Communication Protocols: I2C and UART

MPU-6050 access other devices through I2C bus, which is a synchronous serial communication protocol, so data is transferred bit by bit along a single wire. I2C in Figure 18 only uses two wires to transmit data between devices, including SDA (Serial Data)-the line for the master and slave to send and receive data and SCL (Serial Clock) – the line that carries the clock signal.

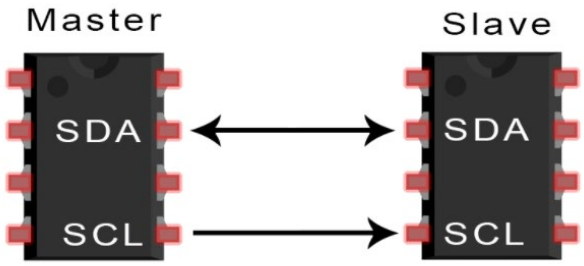


Figure 18 I2C Principle

With I2C, data is transferred in a message (Figure 19). The message is decomposed into data frames. Each message has an address frame that contains the binary address of the slave station and one or more data frames that contain the data being sent. The message also includes start and stop conditions, read/write bits, and ACK/NACK bits between each data frame.

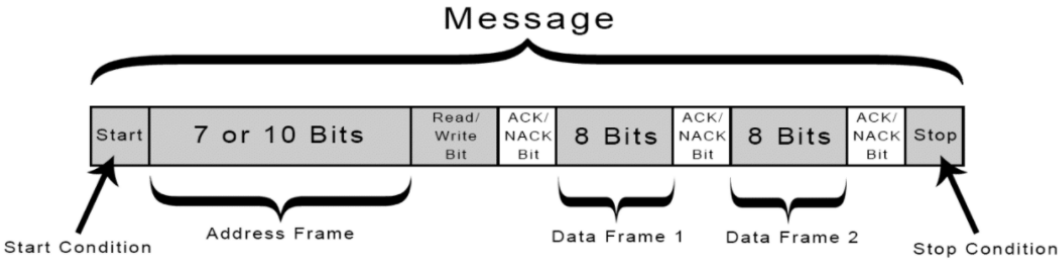


Figure 19 I2C message

Start condition: SDA line from high voltage level to SCL line from high to low voltage level.

Stop condition: SDA line is switched from low voltage level to SCL line from low to high voltage level.

Address frame: a unique 7-bit or 10-bit sequence for each slave device that identifies each slave when the master mother tongue wants to talk to it.

Read/write bits: Specifies whether the master sends data to or requests data from slave devices (low voltage levels).

ACK/NACK bit: Each frame in the message is followed by an acknowledgement/no acknowledgement bit. If the address frame or data frame is successfully received, it is returned from the receiving device to the sender's ACK bits

We refer to the relative materials and find that there are two different methods to define I2C protocol in Micro Python language. ‘pyb’ and ’machine’ are two library functions with different grammar in define and use I2C protocol. We can write code either ‘from pyb import I2C’ or ‘from machine import I2C’’. The figures below illustrate the detailed codes for the two methods respectively.

For ‘pyb’:

```
i2c=I2C(1,I2C.MASTER,baudrate=400000)
men_write()
mem_read()
```

For ‘machine’:

```
i2c=I2C(scl='PB6',sda='PB7',freq=400000)
readfrom_mem()
writeto()
```

3.3.3 Data Processing

To exploit the data generated by JY901S, we need to obtain them via UART communication and then process them to be their usable form.

The initialization of UART execution in `main` is listed as follows:

```
UART_Gyroscope = UART(2)
UART_Gyroscope.init(38400, bits=8, parity=None, stop=1,
timeout_char=100)
```

```
global z_angle
count=10000
UART_Gyroscope.irq(trigger = UART.IRQ_RXIDLE, handler =
UART_Gyroscope_ISR)
global original_angle
```

Code 4 UART initialization with Gyroscope

The global variables are defined for future applications.

Data processing is finished in the interrupt service routine (ISR), where the function `DueData` is called.

```
def UART_Gyroscope_ISR(t):
    global turn_angle
    global count
    global signal
    global command
    global original_angle
    global s_original_angle
    global angle
    global z_angle
    global y_angle
    msg_Gyroscope=UART_Gyroscope.read(UART_Gyroscope.any())
    angle = jy901.DueData(msg_Gyroscope)
    z_angle=angle[0]
    y_angle=angle[1]

    if type(z_angle)==float:
      if count==0:
          original_angle=z_angle
          s_original_angle=z_angle
      count=count+1
    return
```

Code 5 Gyroscope ISR

Here the condition `if type(z_angle)==float:` is invoked in case a null value is read. Again, global variables are defined for future application because an ISR can neither receive parameters nor return any.

The module that processes data is listed as follows:

```
from pyb import UART
from time import sleep
```

```
ACCData=[0.0]*8
GYROData=[0.0]*8
AngleData=[0.0]*8
FrameState = 0
Bytenum = 0
CheckSum = 0

a = [0.0]*3
w = [0.0]*3
Angle = [0.0]*3


def DueData(inputdata):
    global  FrameState
    global  Bytenum
    global  CheckSum
    global  a
    global  w
    global  Angle
    for data in inputdata:
        if FrameState==0:
            if data==0x55 and Bytenum==0: #Start reading at 0x55 and
increase bytenum
                CheckSum=data
                Bytenum=1
                continue
            elif data==0x51 and Bytenum==1:
                CheckSum+=data
                FrameState=1
                Bytenum=2
            elif data==0x52 and Bytenum==1:
                CheckSum+=data
                FrameState=2
                Bytenum=2
            elif data==0x53 and Bytenum==1:
                CheckSum+=data
                FrameState=3
                Bytenum=2
        elif FrameState==1: # acc

            if Bytenum<10:           # read 8 bits
                ACCData[Bytenum-2]=data
```

```
                CheckSum+=data
                Bytenum+=1
            else:
                if data == (CheckSum&0xff):
                    a = get_acc(ACCData)
                CheckSum=0
                Bytenum=0
                FrameState=0
        elif FrameState==2: # gyro

            if Bytenum<10:
                GYRODData[Bytenum-2]=data
                CheckSum+=data
                Bytenum+=1
            else:
                if data == (CheckSum&0xff):
                    w = get_gyro(GYRODData)
                CheckSum=0
                Bytenum=0
                FrameState=0
        elif FrameState==3: # angle

            if Bytenum<10:
                AngleData[Bytenum-2]=data
                CheckSum+=data
                Bytenum+=1
            else:
                if data == (CheckSum&0xff):
                    Angle = get_angle(AngleData)
                CheckSum=0
                Bytenum=0
                FrameState=0
    return Angle


def get_acc(datahex):
    axl = datahex[0]
    axh = datahex[1]
    ayl = datahex[2]
    ayh = datahex[3]
    azl = datahex[4]
    azh = datahex[5]
```

```
    k_acc = 16.0

    acc_x = (axh << 8 | axl) / 32768.0 * k_acc
    acc_y = (ayh << 8 | ayl) / 32768.0 * k_acc
    acc_z = (azh << 8 | azl) / 32768.0 * k_acc
    if acc_x >= k_acc:
        acc_x -= 2 * k_acc
    if acc_y >= k_acc:
        acc_y -= 2 * k_acc
    if acc_z >= k_acc:
        acc_z-= 2 * k_acc

    return acc_x,acc_y,acc_z


def get_gyro(datahex):
    wxl = datahex[0]
    wxh = datahex[1]
    wyl = datahex[2]
    wyh = datahex[3]
    wzl = datahex[4]
    wzh = datahex[5]
    k_gyro = 2000.0

    gyro_x = (wxh << 8 | wxl) / 32768.0 * k_gyro
    gyro_y = (wyh << 8 | wyl) / 32768.0 * k_gyro
    gyro_z = (wzh << 8 | wzl) / 32768.0 * k_gyro
    if gyro_x >= k_gyro:
        gyro_x -= 2 * k_gyro
    if gyro_y >= k_gyro:
        gyro_y -= 2 * k_gyro
    if gyro_z >=k_gyro:
        gyro_z-= 2 * k_gyro
    return gyro_x,gyro_y,gyro_z


def get_angle(datahex):
    rxl = datahex[0]
    rxh = datahex[1]
    ryl = datahex[2]
    ryh = datahex[3]
```

```
    rzl = datahex[4]
    rzh = datahex[5]
    k_angle = 180.0

    angle_x = (rxh << 8 | rxl) / 32768.0 * k_angle
    angle_y = (ryh << 8 | ryl) / 32768.0 * k_angle
    angle_z = (rzh << 8 | rzl) / 32768.0 * k_angle
    if angle_x >= k_angle:
        angle_x -= 2 * k_angle
    if angle_y >= k_angle:
        angle_y -= 2 * k_angle
    if angle_z >=k_angle:
        angle_z-= 2 * k_angle
    return angle_z,angle_y
```

Code 6 Module jy901

The data is collected as byte variables `inputdata`, which is received as an input of function `DueData`. In `DueData`, the header of the input data is removed, and the remaining data is classified according to their starting bits, which represent their address in JY901S. Once it is determined whether it represents acceleration, gyroscope reading or angle, the data is passed to corresponding specified functions. In our project, y angle and z angle are required, so we extract them only.

3.3.4 Comparison between 6-axis Scheme and 9-axis Scheme

We need the gyroscope to decide whether the car has completed its turn and reached its target angle. At first, we chose for our jy901S the 9-axis gyro sensor consisting of 3 accelerometer axes, 3 gyroscope axes, and 3 magnetometer axes additional to its 6-axis counterpart. The magnetometer measures the magnetism of the earth, which automatically determines the zero position for z-angle. Accordingly, its 180 degree and -180 degree axes, which appear to be the same one, are also decided. This brings difficulty to our operation because we cannot set 0 by ourselves, and we might encounter the problem that the car should cross the -180/180 line, which gives rise to a jump in angle value.

We solved the problem by setting a global variable *count* that counts the times of interrupt. Whenever we need to set a starting position for a turning, where the gyroscope is involved, we set *count* to 0, and record the current value read from the gyroscope. The difference between current axis and the original one is then calculated each time a new current position is read. The problem brought by the -180/180 intersection was solved through classification of scenarios. When the original angle is between -90° and -180° and the target angle is between 90° and 180°, we have:

$$delta = original\ angle - target\ angle + 360°,$$

and when the original angle is between 90° and 180° and the target angle is between -90° and -180°, we have:

$$delta = original\ angle - target\ angle - 360°.$$

In this way, our delta angle is guaranteed to be in the range of [0,360°].

However, another problem is encountered when we began to test our algorithm on the car. The car is discovered to turn to the position that is quite different from our desired one. It is then found out that the angle read by the gyroscope is far from accurate. For instance, it read 270 degrees when we turn it 180 degrees up. Still worse, the situation was not improved much after calibration.

It is then discovered that when we switched to 6-axis scheme, that is, abandoning the magnetometer axes, the behavior of the gyroscope turned out surprisingly good. The reading was accurate, and the zero position can be set. However, as the zero-position problem has been solved in our algorithm, we decided not to change the it, which can also be used in 6-axis scheme.

3.3.5 Applications

3.3.5.1 Application1: turning on the spot

An essential application involving JY901S is to assist the car to complete accurate turnings of arbitrarily given angles. To achieve this goal, we read the z angle at the very moment of instruction and record it as `original_angle`. Our target angle is calculated as

$$target\_angle = original\_angle + turn\_angle,$$

where $turn\_angle$ comes from instruction from OpenMV. We then update `z_angle` each time a new one is read in ISR, and this is where the global variable `z_angle` comes into use. The difference `delta` is also updated each time `z_angle` changes. Note that in JY901S, `delta`>0 implies a counterclockwise turning and `delta`>0 implies a clockwise one. When the absolute value of `delta` is close enough to zero, it is regarded that the turning task is accomplished. A function `turn` is defined for the car to turn, making two wheels on one side to turn forward and two wheels on the other side to turn backward. The code for function `turn` is listed below:

```
def turn(delta):
    if delta>0:                     #clockwise
        if delta>20:
            Car.run_(0,40,40,0,40,0,0,40)
            time.sleep(0.4)
```

```
        if delta<20:
            Car.run_(0,30,30,0,30,0,0,30)
            time.sleep(0.2)
            Car.run_(0,0,0,0,0,0,0,0)
            time.sleep(0.1)
    if delta<0:                      #anticlockwise
        if delta<-20:
            Car.run_(40,0,0,40,0,40,40,0)
            time.sleep(0.4)
        if delta>-20:
            Car.run_(30,0,0,30,0,30,30,0)
            time.sleep(0.1)
            Car.run_(0,0,0,0,0,0,0,0)
            time.sleep(0.1)
```

Code 7 Turning

When the absolute value of `delta` is below 10°, the pwm input for the wheels is set to be lower than normal. This is to ensure fine adjustments to be made so that the car would stop at a precise angle.

3.3.5.2 Application2: Send down-bridge signal to OpenMV

When the car goes down from the bridge, OpenMV should be informed to get back to the state of tracing. This requires a signal from stm32, which is generated when JY901S senses a large y angle. The code for this coincides with the first application in terms of UART and ISR. The different part is listed as below:

```
global y_angle
#       print("y_angle:",y_angle)
        if(y_angle>10):
            enableCamera=1
            print("sent enable camera.")
            UART_OpenMV.write(str(enableCamera))
```

Code 8 Down-bridge signal

3.3.5.3 Application3: dynamic self-adaption direction maintenance

The third application of gyroscope is dynamic self-adaption direction maintenance in order to make car go straight. The key element of this task is to ensure the running route be an approximately straight line within tolerate drift angle as 0.1°. It is critical in both patio 1 and patio 2.

The first method we tried was to use the speed feedback of the wheel to make the right and left

sides of the wheel the same speed by setting the PID with reasonable parameters. In other words, it is to walk in a straight line. Based on this, we have done work on the encoder, converting the number of pulses received by the encoder into speed, and adding the PID algorithm to achieve the same speed on both sides.

```
err = target - now #now:'count'
pwm = pwm + self.kp*(err - last_err) + self.ki*err + self.kd*(err -
last_err)
if (pwm >= self.pwm_range):
    pwm = self.pwm_range
if (pwm <= -self.pwm_range):
    pwm = -self.pwm_range
last_err = err
return pwm
```

Code 9 PID algorithm

However, the fact is not what we imagined. Even if we can make the rotation speed on both sides of the wheel exactly and quickly, due to the unevenness of the ground, the accidental idling will cause the deviation of the car. Especially on the cobblestone pavement of the second patio.

Figure 20 Cobblestone pavement

Gyroscope was then implemented to accurately perceive the angle of the current car and adjusted the output values of the PWM motors on both sides by analyzing the angle offset. The simplified version of the logic is as follows: if the angle is greater than 0.1, the car is tilted to the left, then we will increase the PWM output of the left wheel; if the angle is less than 0.1, the car is tilted to the right, then we will increase the PWM output of the right wheel.

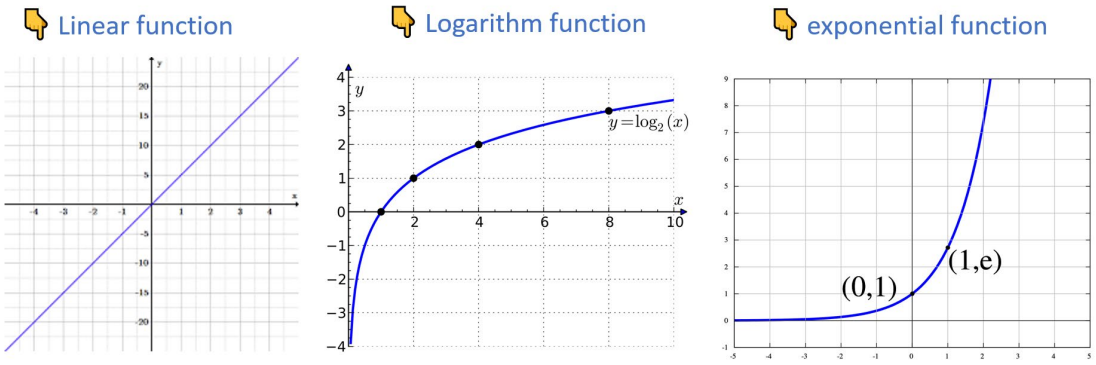


Figure 21 Three functions

The final solution: Since the change of PWM should be related to the change of angle, by comparing three functional relationships, including linear function, in function and exponential function, we find that the exponential function changes little when the angle is small, and the changes are obviously when the angle is large, this feature satisfies our idea of fine-tuning in small angles and drastically adjusting in large angles.

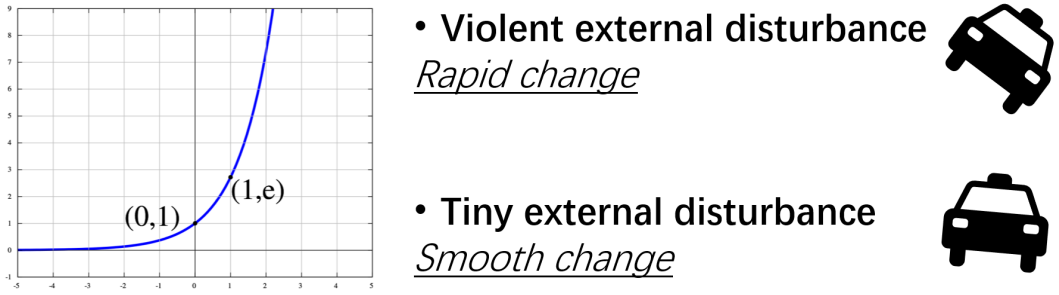


Figure 22 The merit of exponential funtion

In addition, we found that adding negative feedback (that is, the current PWM value is the last PWM value) and limiting the maximum PWM difference between the two wheels of the trolley will make the trolley adjust more quickly and the offset when the trolley goes straight will be smaller. The car has outstanding performance on the cobblestone road.

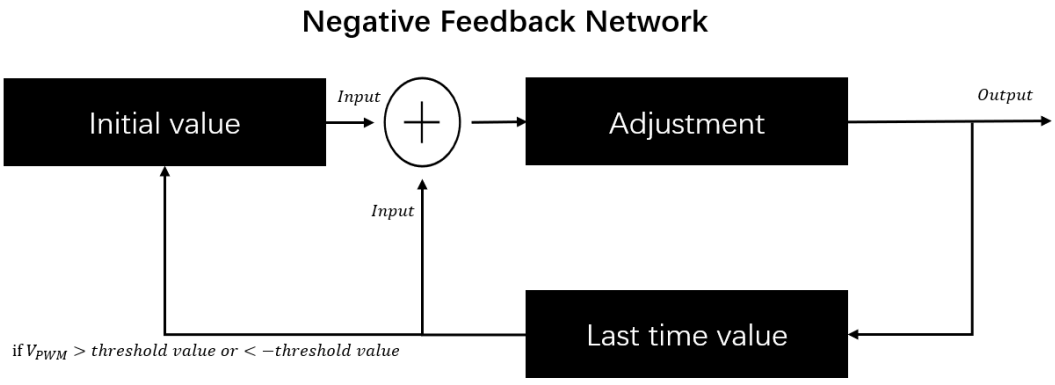


Figure 23 negative feedback network

```
if(sum<-yuzhi):
       self.NrA1=self.NrA1+increment
       self.NrD1=self.NrD1+increment
       self.NlC1=self.NlC1-increment
       self.NlB1=self.NlB1-increment
       if((self.NrA1-self.NlB1)>rang):
           self.NlB1=base-(rang/2)
           self.NlC1=base-(rang/2)
           self.NrA1=base+(rang/2)
           self.NrD1=base+(rang/2)
if(sum>yuzhi):
       self.NlB1=self.NlB1+increment
       self.NlC1=self.NlC1+increment
       self.NrA1=self.NrA1-increment
       self.NrD1=self.NrD1-increment
       if((self.NlB1-self.NrA1)>rang):
           self.NlB1=base+(rang/2)
           self.NlC1=base+(rang/2)
           self.NrA1=base-(rang/2)
           self.NrD1=base-(rang/2)
if(sum<=yuzhi and sum>=-yuzhi):
   pass
```

Code 10

3.4 Car Group: Velocity Feedback

Section Author	**Xinyue Li** **(UESTC ID:2018190505006, UofG ID:2429596L)**
Technically assisted by	Chenyang Fan (UESTC ID:2018190505036, UofG ID:2429626F) Shubo Yang (UESTC ID:2018190607005, UofG ID:2429400Y)

In order to enable the car to walk in a straight line with a small offset, whether on a cobblestone road or a flat road, we chose a Hall encoder to obtain the speed of the four wheels by reading the number of pulses received by the encoder.

Figure 24 Hall Encoder

The red wire is the power cable；the green wire is the encoder phase A output; the white wire is the encoder phase B output；The black wire is the encoder ground wire.

The encoder has AB phase output, so it can not only measure the speed, but also distinguish the direction of rotation. According to the wiring instructions in the above figure, we can see that we only need to supply 5V to the encoder power supply, and the square wave signal can be output through the AB phase when the motor is rotating. The encoder comes with a pull-up resistor, so there is no need for an external pull-up, and it can be directly connected to the microcontroller IO for reading.

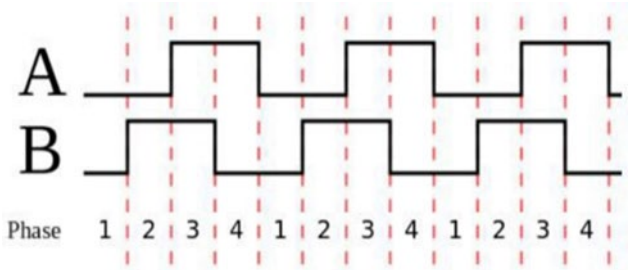


Figure 25 Phase A and Phase B

The quadruple frequency technique is applied to the encoder by measuring the rising and falling edges of the A-phase and B-phase encoders. In this way, 12 times (3 cycles of 1234) can be counted in the same time. The number of lines per circle of our encoder is 390, so in the case of quadruple frequency and AB phase, there will be 1560 pulses per circle.

As regards velocity measurement methods, there are two suitable methods we can choose: M method and T method. After Comparing and analyzing the difference between M velocity measuring method and T method, we found that there maybe an error of 2 pulses as the principle of M method is converting the number of pulses per unit time into frequency. And the principle of T method is measuring the time between two pulses , and then obtain the frequency ,which could not obtain accurate velocity while measuring high speed. After comparing with the actual speed, we decided to use M method since it is relatively more accurate.

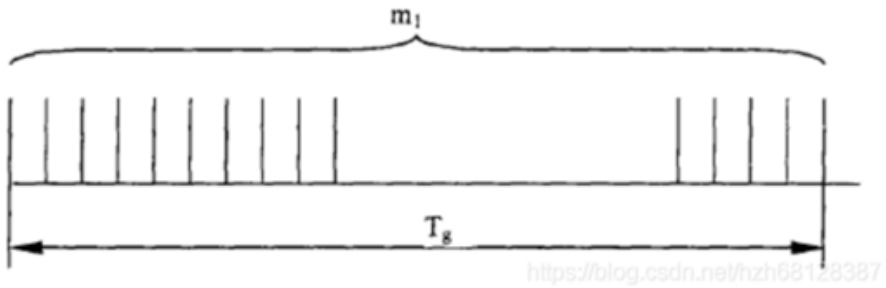


Figure 26 M velocity measurement method

Figure 27 T velocity measurement method

The single-chip microcomputer with encoder interface, STM32, can directly use the hardware to count and collect encoder data.

MG540 motor&encoder	
Encoder A phase	X1(CH1)
Encoder B phase	X2(CH2)
Encoder 5V	V+(Vin)
Encoder GND	GND
The motor 12V	12V power supply
The motor GND	Power supply GND

Table 4 The wiring diagram of motor and encoder

We use external interrupts to collect the number of pulses, and use formulas to calculate the

speed of each wheel.

As for how to capture the pulses through the code, we used the external interrupt method.The external interrupt is an internal mechanism for the MCU to process external events in real time. When the rising edge and the falling edge appears, the interrupt system of the single-chip microcomputer will force the CPU to suspend the program being executed, and then proceed to the processing of the interrupt event: count=count+1; after the interrupt processing is completed. It returns to the interrupted program and continues execution.

The code is shown below in the figures.

```
from pyb import Pin, Timer
#import pyb
import time
import cmath
import pyb
count=0
#class Encoder:
#     def __init__(self, encoder_freq):
#         self.pi = cmath.pi  # the value ofpi
#         self.encoder_precision = 4 * 390 * 30
#         self.encoder_freq = encoder_freq
#     def enc_to_speed(self):
# while 1:
# time.sleep(0.1)
# speed = (count/encoder_precision) * (2 * self.pi * 0.0335) * 1000
# return speed
# count=0
#     def target_enc_process(self, speed):
#          enc = (speed *self.encoder_precision) / ((2 * self.pi *
0.0335) * 1000)
#         return enc
#def Encoder(count, extint1,extint2):
#    extint1.enable()
#    extint2.enable()
#    time.sleep(0.1)
#    extint1.disable()
#    extint2.disable()
#return count
#def callback(line):
#    global count
#    count=count+1
def callback(line):
```

```
   global count
   count=count+1
# wheel 1
    A8 = Pin('A8')
    B9 = Pin('B9')
    extint1    =    pyb.ExtInt(A8,    pyb.ExtInt.IRQ_RISING_FALLING,
pyb.Pin.PULL_DOWN, callback)
    extint2    =    pyb.ExtInt(B9,    pyb.ExtInt.IRQ_RISING_FALLING,
pyb.Pin.PULL_DOWN, callback)
    count=0
# wheel 2
    B12 = Pin('B12')
    B10 = Pin('B10')
    extint3    =    pyb.ExtInt(B12,    pyb.ExtInt.IRQ_RISING_FALLING,
pyb.Pin.PULL_DOWN, callback)
    extint4    =    pyb.ExtInt(B10,    pyb.ExtInt.IRQ_RISING_FALLING,
pyb.Pin.PULL_DOWN, callback)
    count=0
# wheel 3
    A4 = Pin('A4')
    A5 = Pin('A5')
    extint5    =    pyb.ExtInt(A4,    pyb.ExtInt.IRQ_RISING_FALLING,
pyb.Pin.PULL_DOWN, callback)
    extint6    =    pyb.ExtInt(A5,    pyb.ExtInt.IRQ_RISING_FALLING,
pyb.Pin.PULL_DOWN, callback)
    count=0
# wheel 4
    A1 = Pin('A1')
    B7 = Pin('B7')
    extint7 = pyb.ExtInt(A1, pyb.ExtInt.IRQ_RISING_FALLING, pyb.Pin.
PULL_DOWN, callback)
    extint8 = pyb.ExtInt(B7, pyb.ExtInt.IRQ_RISING_FALLING, pyb.Pin.
PULL_DOWN, callback)
    count=0
def Encoder(n,f):
    global count
    if n==1:
        extint1.enable()
        extint2.enable()
        time.sleep(1/f)
        speed = (count / 1560) * (2 * 3.14159 * 0.0335) * f
        extint1.disable()
```

```
        extint2.disable()
    if n==2:
        extint3.enable()
        extint4.enable()
        time.sleep(1/f)
        speed = (count / 1560) * (2 * 3.14159 * 0.0335) * f
        extint3.disable()
        extint4.disable()
    if n==3:
        extint5.enable()
        extint6.enable()
        time.sleep(1/f)
        speed = (count / 1560) * (2 * 3.14159 * 0.0335) * f
        extint5.disable()
        extint6.disable()
# pass
    else:
        extint7.enable()
        extint8.enable()
        time.sleep(1/f)
        speed = (count / 1560) * (2 * 3.14159 * 0.0335) * f
        extint7.disable()
        extint8.disable()
# pass
        count = 0
return speed
```

Code 11 Velocity feedback

3.5 Car Group: Robotic Arm

Section Author **Xinyue Li**
(UESTC ID:2018190505006, UofG ID:2429596L)

3.5.1 Identify the Requirement and Problem Formulation

Initially, it is significant to identify the requirement of the robotic arm. The robot is required to release the fish food at the release point. The figure below shows the size of the railing at releasing point.

The programming requirement is that the robotic arm receives instructions from openmv,

when OpenMV transmits the releasing instruction, the robotic arm release the fish food to the target position.

It is noteworthy that, due to the sophisticated design of our car, the size of our robotic arm should be carefully controlled. Also, we are required to avoid influencing the view of camera which is displayed at the front of the car.

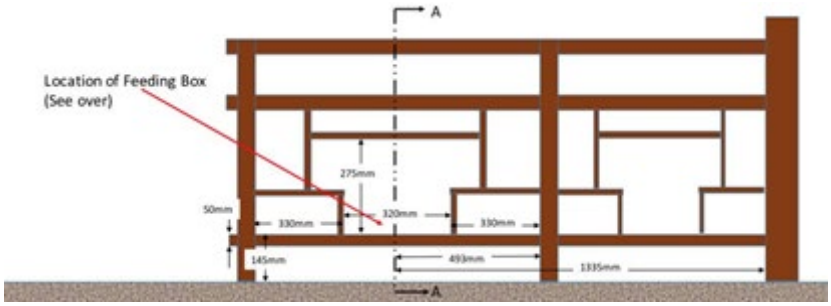


Figure 28 Size of the railing at the releasing point

3.5.2 Ideas of Robotic Arm:

At the inception, we adopted two methods.

Method 1:

We combined a Conveyor belt and a robotic claw together, after receiving instruction of releasing, the conveyor belt transmits the claw to the release point, then the claw is programmed to release the food.

This method effectively solves the problem of influencing the view of camera, however, it transmits stiffly and cannot adjust the angle flexibly.

Figure 29 Conveyor

Figure 30 Claw

Method 2:

We designed a two degrees of freedom robotic arm with a box sticked to the claw, which can move more flexibly.

Figure 31 Robotic arm

3.5.3 The Final Design of Robotic Arm

At last, we adopted the two degrees of freedom robotic arm with a box sicked to the claw. The two main joints are controlled by 2 steering gear engines. When receiving the instructions from OpenMV, the robotic arm will perform the task by casting the fish food to the releasing point and return to the original state.

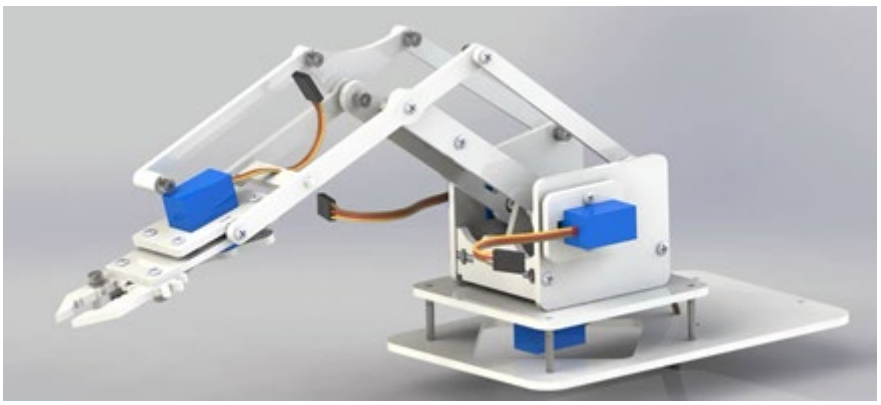


Figure 32 structure of robotic arm

3.5.4 Components and Basic Structure

We construct the arm with the components, our arm mainly contains 3 main parts, which including the claw, left arm and right arm and the foundation. The function of the claw is to hold the fish food, which can open up to 5cm, the main role of right and left arm is to lift up the claw with fish food.

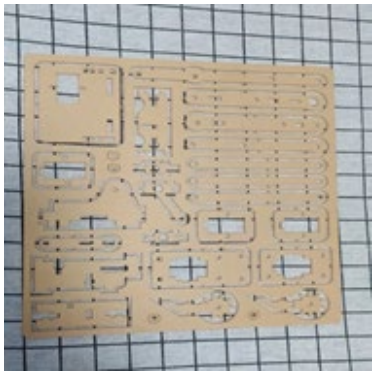


Figure 33 Components of robotic arm

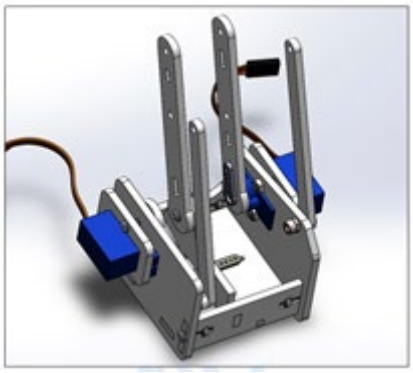


Figure 34 Left and right arm of the robotic arm

3.5.5 Size of the Robotic Arm

After adjustment and measurement, we obtained the largest arm span is up to 23 cm, and the arm can dive down as deep as 7 cm, the highest lifting distance is 15cm.

Figure 35 Size of robotic arm

3.5.6 Working Principle of Servo

The two joints of the arm are controlled by two servos (MG90S) respectively, where the rotation angle of servos can be adjusted by PWM value. The MG90S is powered by 5V. Servo mainly relies on pulses for positioning. Basically, it can be understood that when the servo motor receives 1 pulse, it will rotate the angle corresponding to that to achieve displacement. In this way, the rotation of the motor can be precisely controlled to achieve precise positioning.

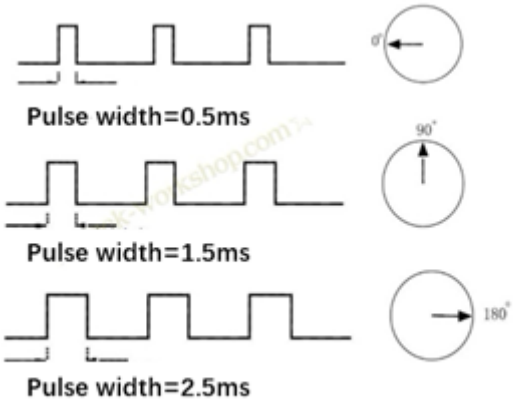


Figure 36 Working principle of servos

3.5.7 Code

```
# main.py -- put your code here!
from pyb import UART, Pin,Timer
import time
from time import sleep

print("hello world")

A0 = Pin('PA0') #control the left steering gear engine
tim = Timer(2, freq=50)
ch = tim.channel(1, Timer.PWM, pin=A0)

B3 = Pin('PB3')# control the right steering gear engine
tim1 = Timer(2, freq=50)
ch1 = tim.channel(2, Timer.PWM, pin=B3)

ch.pulse_width_percent(6.3)
ch1.pulse_width_percent(8.5)
```

```
def releasing():
    i=6.3 #left
    m=8.5 #right
    flag=1
    while flag:
        i+=0.1
        m+=0.1
        ch.pulse_width_percent(i)
        ch1.pulse_width_percent(m)
        if m>=12:
            m=12
        if i>=11.8 and m>=12:
            #print(i)
            #print(m)
            i=11.8
            m=12
            ch.pulse_width_percent(i)
            ch1.pulse_width_percent(m)
            time.sleep(2)
            while i>=6.3 or m>=8.5:
                if i>=6.3:
                    i=i-0.1
                if m>=8.5:
                    m=m-0.1
                print(i)
                print(m)
                time.sleep(0.3)
                ch.pulse_width_percent(i)
                ch1.pulse_width_percent(m)
                if i<6.3 and m<8.5:
                    flag=0
```

Code 12 Robotic Arm

3.6 Car group: Debugging (The Breakdown of STM32 Boards)

Section Author	**Ziyang Long** **(UESTC ID:2018190502030, UofG ID:2429503L)**
Technically Assisted by	Yuchen Yao (UESTC ID:2018190602001, UofG ID:2429207Y)

Weizhe Zhao
(UESTC ID:2018190606004, UofG ID: 2429361Z)

Preface:

As one invisible task, debugging was often neglected to demonstrate, however, bugs probably are the most troublesome thing in both software and hardware, especially in hardware. Most of the problems are proved to be simple afterwards, it took a lot of time to investigate.

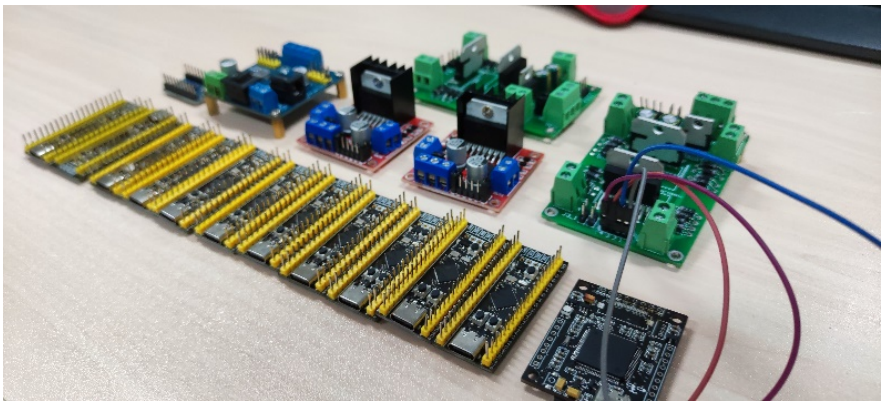


Figure 37 Boards' graveyard

3.6.1 Wrong Design in L298N PCB

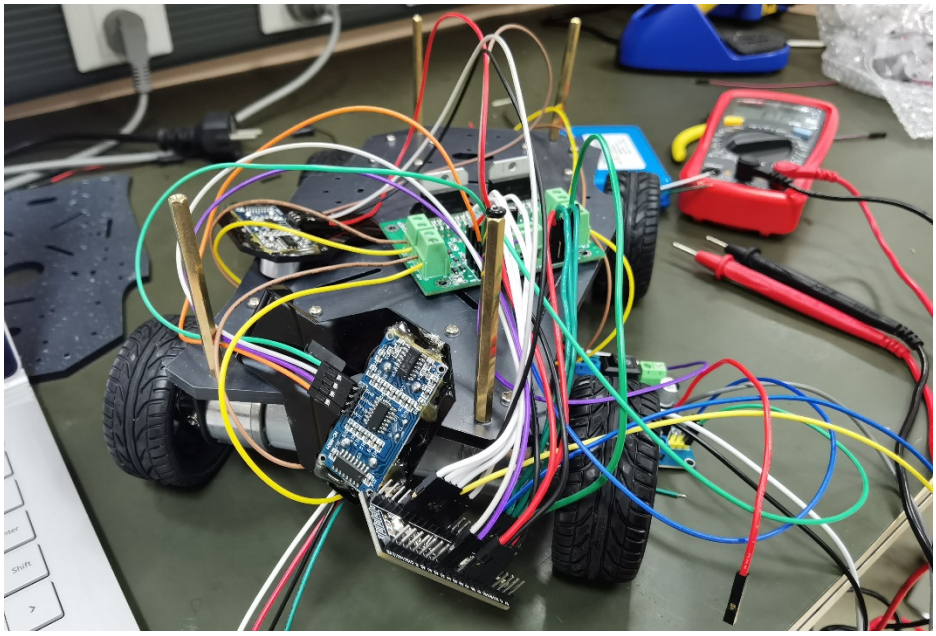


Figure 38 The first breakdown STM32 board

The first two STM32' broken was found when the power LED of STM32 board die out and cannot read the data from STM32 board when we use type C line to connect it with laptop. By checking other part of module, we focus on the problem in L298N. We design the initial L298N

PCB based on a reference file found on the internet. With only a sketchy glance of its design, we solder L298N boards, and we merely checked its power supply.

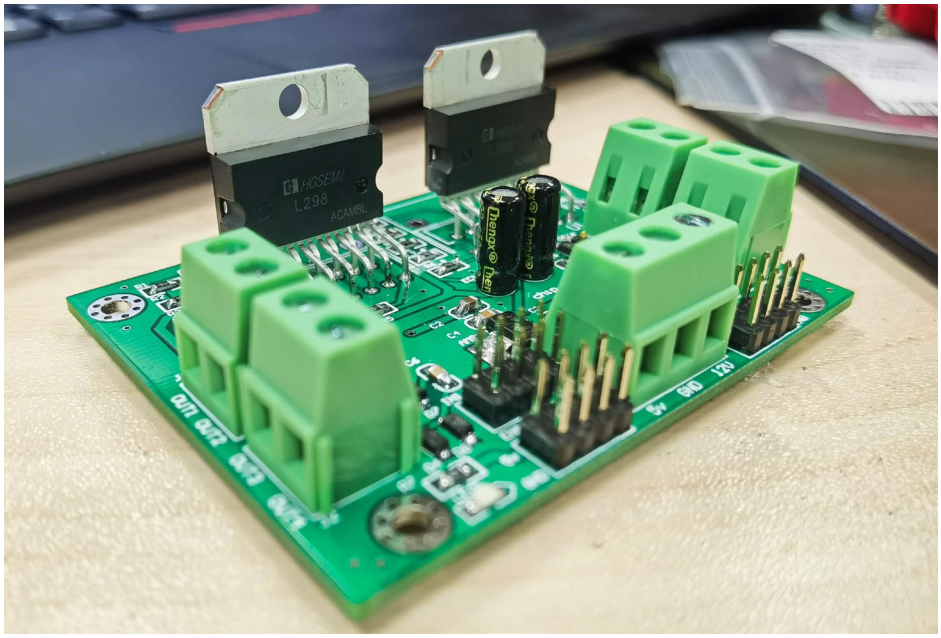


Figure 39 initial wrong design L298N module

By using Altium designer to scrutinize every detail of that PCB design in Figure 40, we found that the input pins were wrongly connect to the 5V power supply line, which is too obscure to notice. This mistake causes the reverse breakdown of STM32 board.

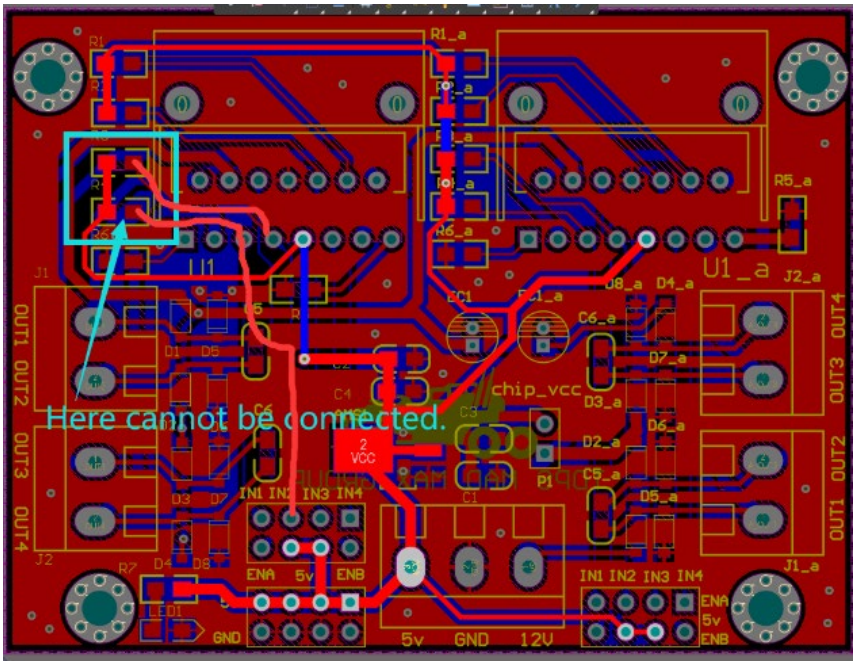


Figure 40 short circuit in PCB design

3.6.2 The Drawback of Directly Connecting L298N Module

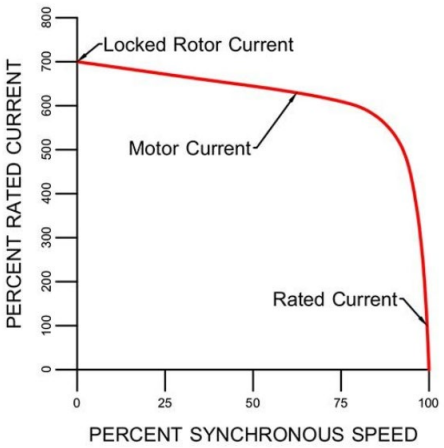


Figure 41 locked rotor current

Without any protection circuit or isolation methods, we found the STM32 will be broken by directly connecting to L298N module after using for a long time. Overcurrent and overvoltage to the Pin port are the main reason. When car's rotor is kept stationary or in other words rotor is not spinning or rotating, it will generate locked rotor current, which basically drawn by the motor at its rated voltage. The maximum current for all pins of the STM32 is 150mA[6], once the locked rotor current exceeds this value for a long period, the Pin port of STM32 will break. The rotor's frequently back and forth switch will let the voltage applied at its terminal be rated voltage of motor. This voltage sometimes high enough to breakdown the diode after the Pin port or the causing the damage of STM32.

3.6.3 Human Error-wrong Wire Connection

The 3.3V Pin port and GND Pin port is very close in STM32. When using multimeter to check the output voltage, it is common to make them short out. The 5V and 3.3V' misuse may breakdown the regulator inside the STM32 causing the damage due to negligence of team member.

3.6.4 Solution: Adding Optical Coupler Isolation between STM32 and L298N

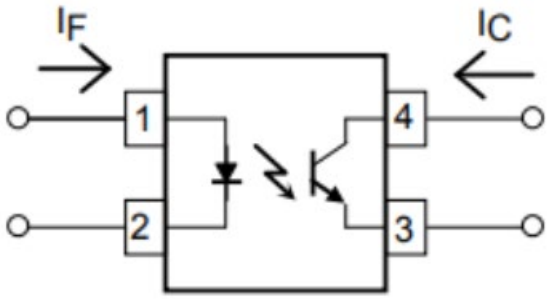


Figure 42 Circuit inside optocoupler

The structure of the optocoupler (Figure 42) is equivalent to the light-emitting diode and photosensitive triode packaged together. The working principle is the process of electricity - light - electricity. The working current drives the light-emitting diode to emit light of a certain wavelength, which is received by the photosensitive triode to produce a certain photocurrent and output after amplification. The optocoupler isolation circuit realize target that no direct electrical connection between the isolated two parts of the circuit, especially between the low voltage control circuit-STM32 and the external high voltage circuit-L298N.

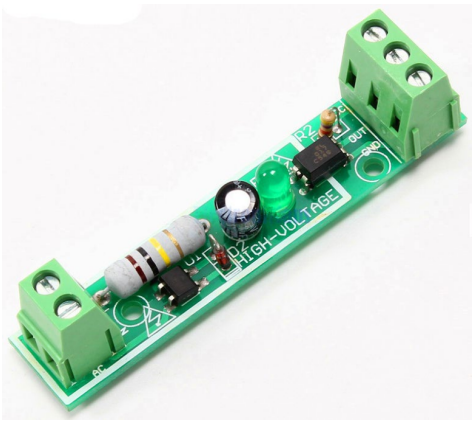


Figure 43 Optocoupler module

Snubber circuit is another potential method to avoid overvoltage, overcurrent and overheat. The inductor's storage and release of energy will remarkably decrease $\frac{di}{dt}$ and $\frac{dv}{dt}$. The switch on and switch off's voltage/current trajectory will be shaped by snubber circuit.

3.7 Image Group: Tracing

3.7.1 Image Processing

Section Author	**Shubo Yang** **(UESTC ID:2018190607005 UofG ID:2429400Y)** **Qinlin Han** **(UESTC ID: 2018190505005 UofG ID:2429595H)**
Technically Assisted by	Ziyi Xie (UESTC ID:2018190502024, UofG ID:2429497X)

Better performance of signal processing helps to improve the performance of tracing.

3.7.1.1 Edge Detection

Edge detection can be considered as recognizing the roughness difference between the road and the off-road path. Since the stones on the road are much rougher than the side path, the edges of the stones can be utilized for further decision making. Several edge detection methods are illustrated in this section. Simple edge detection is what we adopted finally. The disadvantages of other detection methods including Canny edge detection, morph edge detection, and Binarization are also presented.

A. Simple edge detection

Simple edge detection is a fast algorithm functioning on grayscale images. It is a high-pass filter that finds edges using function on OpenMV “image.EDGE_SIMPLE”. A binarization process is embedded in this function, with threshold adjustable. Thus, the threshold should be altered according to different ambient light, and made robust to different environmental conditions. The code for simple edge detection is as in Code 13, where the vector “img_threshold” contains the lower and upper bound of the binarization threshold. For optimal edge detection performance, the threshold should be experimented according to different environmental conditions. Besides, simple edge detection can work on images of different resolution, which provides us the freedom to choose.

```
# Parameter setup
img_threshold=[130,200]
sensor.set_pixformat(sensor.GRAYSCALE) # or sensor.GRAYSCALE
sensor.set_framesize(sensor.QVGA)
sensor.skip_frames(20)
sensor.set_auto_gain(False)  # must turn this off to prevent image washout
sensor.set_auto_whitebal(False)   # must turn this off to prevent image
```

```
washout

# Simple edge detection
img = sensor.snapshot()
img.find_edges(image.EDGE_SIMPLE,            threshold=(img_threshold[0],
img_threshold[1]))
```

Code 13 Simple edge detection.

The resulting image processed by simple edge detection is shown in Figure 44. As in the figure, the edges of the rough stones can be clearly detected as white points, while the off-road path has much less edges, and is generally black. The line dividing the white and black points is the side of the road.

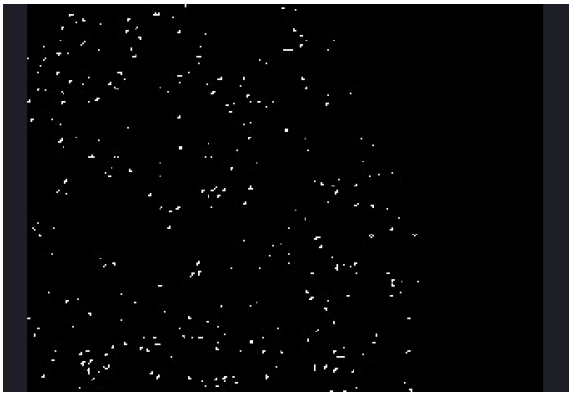


Figure 44 Image processed by simple edge detection.

Simple edge detection outperforms other methods. This is because when other methods are trying to filter out the noise, they filter out the features and edges contained in the noise. Simple edge detection acts as a high-pass filter, and it maintains the edge characteristics in the image and the noise, and thus is adopted.

B. Binarization

Binarization is an alternative method of image processing. According to different color thresholds, the road is expected to be binarized into white while the side path into black. In this way, the road can be identified. “img.binary()” function can binarize the images with its threshold carefully set. In experimental trails, we use “img.binary()”and adjust the thresholds. However, because the rough pebbles on the road, the black and white pixels are crossed and the edges are hard to distinguish from the binarized images. Additionally, we find that the performance is affected by the light and shades, so this solution is not robust against environmental changes. As in Figure 45, though this is a rather good one among all binarized images, the color threshold still cannot distinguish off-road interference, that is the white points on the right. This image also proves the difficulty of adjusting the proper threshold.

Figure 45 Figure after binarization.

C. Canny edge detection

Canny edge detection algorithm is a mature one containing noise reduction. The steps are preprocessing, calculating gradients, non-maximum suppression and threshold with hysteresis. For preprocessing, the image is smoothed using Gaussian filter. For calculating gradients, gradient magnitudes and directions are calculated at every single point in the image by using the finite difference of the first-order partial derivative. The magnitude of gradient is: $m = \sqrt{{G_x}^2 + {G_y}^2}$, and the direction of gradient is: $\theta = \tan^{-1}(\frac{G_y}{G_x})$. Non-maximum suppression of gradient amplitudes simply means suppressing a pixel if the pixel is not a maximum. At each point, the center pixel m of the neighborhood is compared with the two pixels along the gradient line. If the gradient value of m is not greater than that of two adjacent pixels along the gradient line, the value of m is 0: $N[i,j] = NMS(M[i,j], \xi[i,j])$. Eventually, threshold with hysteresis double threshold algorithm is used to detect and connect edges.

D. Morph edge detection

The morph edge detection is convolving the image with a kernel. It is also applied on grayscale images. The brightness of each pixel can be adjusted in the function “img.morph(kernel, kernel_size)”.

Nevertheless, the canny and morph operator functions both reduce the noise, during which process, the features are also removed and the performance of edge detection is affected.

3.7.1.2 Filters

Filters are critical methods in image processing, which removes noise and makes features explicit for further decision making. In this section, diverse filters are described, and each of them has respective effects.

However, as in section 3.7.1.1, the filters are not adopted because they reduce the

characteristics contained in the images, which makes it more difficult to identify the edges. Additionally, after experimentation, the filters do not provide great improved performance. On the contrary, the filters prolong the execution time of image processing, which is detrimental for decision making, since the decisions must be simultaneous with the car motions. Thus, simple is perfect. This means that when filtering has not much performance advancement, no filters is the optimal choice, keeping all the features.

A. Median filter

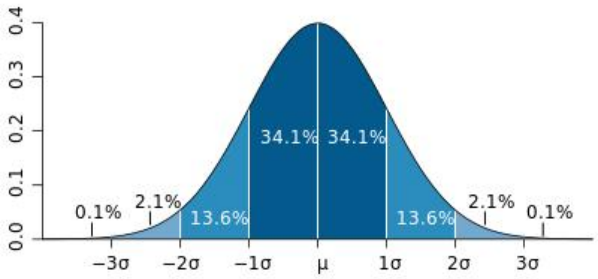
Each pixel in the image and its neighbors would be considered in turn to determine whether it is able to represents its surroundings. All the pixel values of the surrounding neighborhood are sorted in numerical order, and then the middle pixel value is used to replace the considered pixel.

B. Mean filter

The main method of mean filtering is domain average, whose basic principle is to replace each pixel value in the original image with the average value. For the pixel (x, y) to be processed, a template composed of several neighboring pixels is selected to calculate the mean value of all pixels in the template, and then the mean value is given to the pixel (x, y). The gray level of the processed image at the point: $g(x,y) = \sum f(x,y)/m$, where m is the total number of pixels including the current pixel in the template.

C. Gaussian blur

Gaussian blur is to use normal distribution to calculate the transformation of each pixel in the image. In two-dimensional space, the normal distribution equation is: $G(u,v) = \frac{1}{2\pi\sigma^2} e^{-(u^2+v^2)/2\sigma^2}$. The convolution matrix composed of non-zero pixels is transformed with the original image. The value of each pixel is the weighted average of the neighboring pixel values. The value of the original pixel has the largest Gaussian distribution value, so that it has the largest weight (the wight of normal distribution).



As the distance between the adjacent pixels and the original pixels becomes longer and longer, the weight becomes smaller and smaller. In this way, the edge effect can be preserved more effectively than other equalization fuzzy filters.

D. Erode and Dilate

The erode and dilate processor have similar function: noise elimination and separating image elements into segments and connecting the adjacent elements in the image. From a mathematical point of view, dilation or erosion is convolution of image and element. Dilation is the operation of seeking the local maximum value. Convolution with element is the maximum value of the pixels in the area covered by this element, and the maximum value is assigned to the pixels specified by the reference point, so as to increase the highlight area. Erosion is the opposite operation, and it could acquire the local minimum value. When the results of the template and the corresponding elements in the input image are both not 0, the result is 0. This operation will delete some pixels of the object boundary and reduce the highlight area.

3.7.1.3 Segmentation

Segmentation is to divide the image into small parts, and each part can be processed individually. This is because the diverse environmental factor makes the image owning different thresholds at different segments. The code for segmentation is shown in Code 14. The number of regions of interest (ROI) regions can be set by variables “xnum” and “ynum”. Then, the 320×240 image is divided into “ROInum” small segments. “xaxis” and “yaxis” contain the indexes for ROI x-axis and y-axis. In “divide_process”, the segments can be processed individually.

```
def divide_process(ROI,value):
    for x in range(ROI[0],ROI[1]):
        for y in range(ROI[2],ROI[3]):
            img=img.set_pixel(x,y,value)

# Binary threshold
thresholds = [(60, 255)]

width=320
height=240

# how many ROI regions on x&y-axis
xnum=2
ynum=2
ROInum=xnum*ynum

# calculate the starting points of ROIx, ROIy
xaxis=[0]*(xnum+1)
yaxis=[0]*(ynum+1)
for a in range(xnum+1):
```

```
    xaxis[a]=a*width/xnum
    yaxis[a]=a*height/ynum

ROIx=[0]*ROInum
ROIy=[0]*ROInum
for a in range(xnum):
    for b in range(ynum):
        ROIx[b+ynum*a]=xaxis[a]
for a in range(ynum):
    for b in range(xnum):
        ROIy[b + xnum * a] = yaxis[b]
```

Code 14 Segmentation.

However, despite this method may offer performance improvement, the processing procedures are tedious and redundant, which increases the program execution time dramatically. The parameters of number of ROI should also be carefully set. Therefore, due to the abovementioned drawbacks, this method is not adopted.

3.7.2 Information Utilization

Section Author	**Ziyi Xie** **(UESTC ID:2018190502024, UofG ID:2429497X)**
Technically Assisted by	Shubo Yang (UESTC ID:2018190607005, UofG ID:2429400Y)

This part mainly discusses how to use the image information to determine the moving direction. There are several methods for completing this task. In the following paragraph, two main methodologies will be illustrated.

3.7.2.1 Method A: Deviation Angle

One method is to calculate the angle between the road's middle line and car's current moving direction. By returning the deviation value, car would turn a certain angle to keep moving along the middle of the line.

The method above needs to solve a critical problem: how to get the middle line of the road. Hough transform, as a traditional method used in image processing, could get clearly straight lines. The effect is shown in Figure 46.

Figure 46 Hough Transform

Another way to get the middle line of the road is by doing mathematical calculation. After binarization, I get Figure 47 which only consists white points and black points(White points represent the road). I use 1 to represent white points and 0 for black points and do counting to get the position of the road middle line. Then using the arctan function to get the deviation angle.

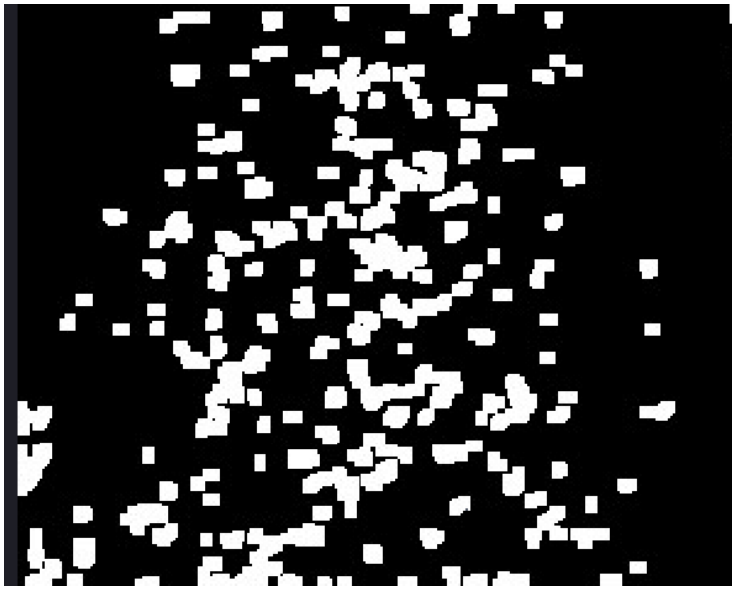


Figure 47 Binarization

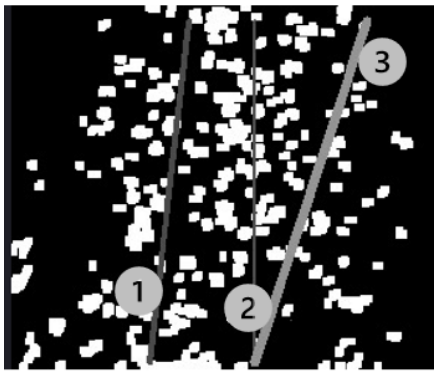
Codes are shown below:

```
while(True):
    for i in range(0,10):
        clock.tick()
        img = sensor.snapshot()
        img.find_edges(image.EDGE_CANNY,            threshold=(120,
250),scale=3)
        widthNum1=[0]*280  #imgWidth
        widthNum2=[0]*280  #imgWidth
        widthTotal1=0
```

```
        widthTotal2=
        # calculate the theta
        x0=120
        y0=180; y1=235; x2=160; x3=160; y2=235; y3=180
        x1=180
        ## for road angle
        x_virtual=int(x3+(x0-x1))
        height_road=(y0-y1)
        width_road=x3-x_virtual
        theta_road=width_road/height_road
        theta_road=math.degrees(math.atan(theta_road))
        road_vec[i]=theta_road
    road_sum=median(road_vec)
    road_sum=sum(road_vec)/len(road_vec)
```

Code 15: Drawing the road middle line and returning deviation angle

Result and Analysis:



✓ 1: Road middle line
✓ 2: Car current direction
✓ 3: Next step direction

Figure 48 Result of calculating angle

The result is visualized in Figure 48, car will move along the line 3 direction. This method of returning angle is simple and straightforward. The returned variable is easy to understand and could be directly used by the car motion group. However, it is not stable enough. The noise points will greatly influence the position of road middle line, resulting in dramatically changing angles. Also, if the car is already off the road, there are few white points in the picture to get the middle line and determine the next step direction.

3.7.2.1 Method B: Horizontal Deviation Distance

The second method of determining the turning direction is to calculate the horizontal deviation

value. I choose 320*240 raw image and setting up coordinate system in Figure 49. It is obvious that the middle line's x-coordinate is 160. Because white points represent road, if they appear on the left side of x=160, middle line should move to the left, which means x_initial middle should decrease. Otherwise, x_initial middle increases. Therefore, we could get the horizontal distance between the car and the road middle line and using pid algorithm to change the left side speed and right side speed separately.

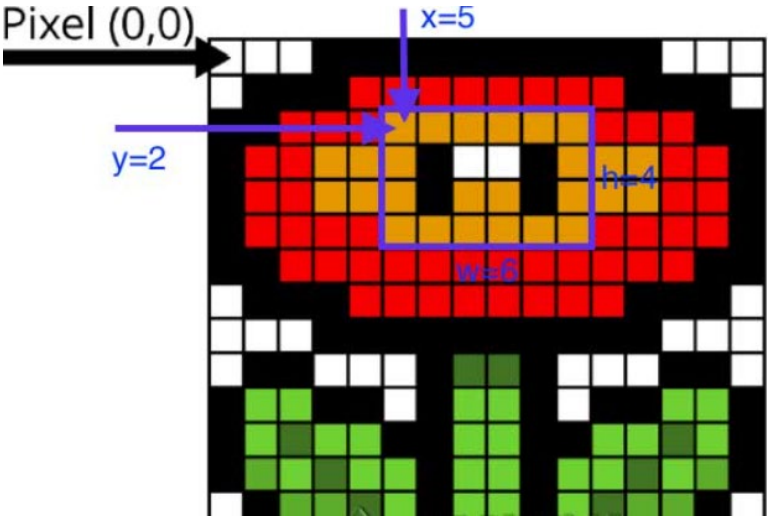


Figure 49 Setting up the coordinate system

```
for i in range(1,320,2):
        for j in range(1,240,2):
            if img.get_pixel(i,j)==255:
                x+=i-160
                count+=1
print("Dot Count=",count)
x_av=x/count
x_error=x_av
x_output=x_pid.get_pid(x_error,0.8*0.5)
img.draw_rectangle((160+int(x_error),0,10,243))
print('x_output:',x_output)
print('x_error:',x_error)
left_pwm=int(25-x_output)
right_pwm=int(25+x_output)
```

Code 16 Horizontal deviation distance

Result and Analysis:

Compared with returning the deviation angle, this method sends the horizontal deviation distance as the PID input variable. It could directly control the PWM output of the car's four wheel and enable the car moving more smoothly. But the noise white points(non-track areas) have an influence on the accuracy of algorithm as well.

3.7.3 Robustness Improvement of Tracing

Section Author	**Ziyi Xie** **(UESTC ID:2018190502024, UofG ID:2429497X)**
Technically Assisted by	Shubo Yang (UESTC ID:2018190607005, UofG ID:2429400Y)

Roading tracing depends on both the image processing and image information selection and utilization. Due to the small RAM of OpenMV and limited resolution of camera, there could be little improvement in pure image processing. Therefore, how to select the useful image information to do analyze is quite important. Otherwise, the car will be misled and making a wrong decision. In the following section, I will display two kinds of problems and interference we met in testing and demonstrate the corresponding solutions.

3.7.3.1 Problem 1: Far Distance Vision

Noise points appear in the edge of the picture, which worsen the effect of calculating the road middle line.

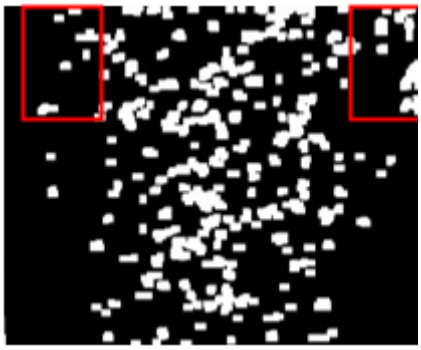


Figure 50 Far distance noise points

Cause:

As the angle between our camera and the ground, the upper side and lower side of the picture needs different focal lengths. Using unique focal length results in noise points in the red region shown above.

Solution:

Two methods are combined to solve this problem. The first way is to choose part of the picture——Region of Interest (ROI) for image processing. In this way, we only need to focus on the threshold for edge detection in ROI instead of the whole picture. Therefore, the threshold could be restricted to a small range to improve the accuracy of edge detection and the effect of road tracing.

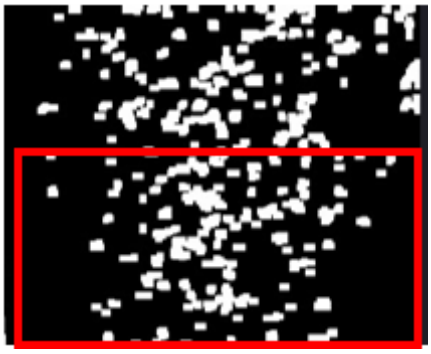


Figure 51 ROI method

```
for i in range(1,320,2):
       for j in range(70,230,2):
           if img.get_pixel(i,j)==255:
               x+=i-160
               count+=1
print("Dot Count=",count)
```

Code 17 ROI method

The second way is to do roughness statics and set a separate threshold. As the road is made up of small stone, which makes it look relatively rougher than the non-track areas which are made up of whole marble. Therefore, the road region obviously has a higher density of white points while the noise area of non-track has a lower density. By setting a threshold, the isolated noise points could be ignored. Results shown in the red region below:

Figure 52 Roughness Statics

```
# Binary threshold
thresholds = [(60, 255)]

width=320
height=240

# how many ROI regions on x&y-axis
xnum=2
ynum=2
ROInum=xnum*ynum

# calculate the starting points of ROIx,ROIy
```

```
xaxis=[0]*(xnum+1)
yaxis=[0]*(ynum+1)
for a in range(xnum+1):
    xaxis[a]=a*width/xnum
    yaxis[a]=a*height/ynum

ROIx=[0]*ROInum
ROIy=[0]*ROInum
for a in range(xnum):
    for b in range(ynum):
        ROIx[b+ynum*a]=xaxis[a]
for a in range(ynum):
    for b in range(xnum):
        ROIy[b + xnum * a] = yaxis[b]


while(True):
    img= sensor.snapshot()

    # find edges
    img.find_edges(image.EDGE_CANNY)

    # process every ROI region
    for i in range(ROInum):

statistics=img.get_statistics(roi=(ROIx[i],ROIy[i],width/xnum,height/ynu
m))
        ROI=[ROIx[i],ROIy[i],ROIx[i]+width/xnum,ROIy[i]+height/ynum]

        # just to see the ROI rectangle?
        img.draw_rectangle(ROI)
        time.sleep(0.5)

        # set to 0/255
        if statistics.median()==255:  # statistics.mean()
            divide_process(ROI,255)
        else:
            divide_process(ROI,0)
```

Code 18 Roughness statics

3.7.3.2 Problem 2: Off the Road

The car will off the road especially in the corner.

Cause:

The previous algorithm for tracing is based on counting the white points to get the road middle line. However, if there are few white points, which means the car is already on the edge of the road, the middle line is hard to be determined.

Solution: Back up mechanism

If the car sees few points, it will back up and return to the road slowly. During this process, it would judge whether the information is enough or not to make decision. The code is shown below:

```
    left_pwm=0
    right_pwm=0
    x=0
    count=0
    for i in range(1,320,2):
        for j in range(70,238,2):
            if img.get_pixel(i,j)==255:
                x+=i-160
                count+=1
    print("Dot Count=",count)

#Back up mechanism
    if count<130:
        left_pwm=-15
        right_pwm=-15
    else:
        x_av=x/count
        x_error=x_av
        x_output=x_pid.get_pid(x_error,0.8*0.5)
        img.draw_rectangle((160+int(x_error),0,10,243))
        print('x_output:',x_output)
        print('x_error:',x_error)
        left_pwm=int(25-x_output)
        right_pwm=int(25+x_output)
```

Code 19 Back up mechanism

3.8 Image Group: Color Recognition

Section Author	**Jingluan Yang (UESTC ID:2018190505002 UofG ID:2429592Y)**
Technically Assisted by	Qinlin Han (UESTC ID:2018190505005, UofG ID:2429595H)

3.8.1 Initial Design

L*a*b* Color Values:

L*a*b* is a color model published by the International Lighting Commission (CIE) in 1976. This mode, which relies neither on light nor pigments, includes color patterns for all colors visible to the human eye.

L*a*b* mode consists of three channels, but not R, G, and B channels. One of its channels is brightness -- L. The other two are colour channels, represented by A and B. Channel A includes colors from dark green (low brightness value) to gray (medium brightness value) to bright pink (high brightness value), and B channel from dark blue (low brightness value) to gray (medium brightness value) to yellow (high brightness value). As a result, this colour blends to produce bright colour.

Colour Histogram:

The values in the colour histogram are statistical, which describes the quantitative characteristics of colour in the image and can reflect the statistical distribution and basic tone of the image colour. The colour histogram contains only the frequency of a colour value in the image and loses the spatial location information in which the pigment is located.

Colour-Recognition:

OpenMV could capture the color thresholds for whatever was in the center of the image by the function img.get_histogram( ) which acquires the colour histogram of the image with Lab mode. Next, the function histogram.get_percentile(percentile) could calculate and select the cumulative distribution function of the colour histogram in a certain range. And then, the range of colour threshold could be calculated by the simple equation. Finally, through the function img.find_blobs([threshold]) could find the colour block which has the similar colour threshold.

```
import sensor, image, time
sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QVGA)
sensor.skip_frames(time = 2000)
```

```
sensor.set_auto_gain(False) # must be turned off for color tracking
sensor.set_auto_whitebal(False) # must be turned off for color tracking
clock = time.clock()

# Capture the color thresholds for whatever was in the center of the
image.
r = [(320//2)-(50//2), (240//2)-(50//2), 50, 50] # 50x50 center of QVGA.

for i in range(60):
    img = sensor.snapshot()
    img.draw_rectangle(r)

threshold = [0, 0, 0, 0, 0, 0] # Middle L, A, B values.
for i in range(60):
    img = sensor.snapshot()
    hist = img.get_histogram(roi=r)
    lo = hist.get_percentile(0.3) # Get the CDF of the histogram at the
1% range (ADJUST AS NECESSARY)!
    hi = hist.get_percentile(0.99) # Get the CDF of the histogram at the
99% range (ADJUST AS NECESSARY)!
    # Average in percentile values.
    threshold[0] = (threshold[0] + lo.l_value()) // 2
    threshold[1] = (threshold[1] + hi.l_value()) // 2
    threshold[2] = (threshold[2] + lo.a_value()) // 2
    threshold[3] = (threshold[3] + hi.a_value()) // 2
    threshold[4] = (threshold[4] + lo.b_value()) // 2
    threshold[5] = (threshold[5] + hi.b_value()) // 2
    for blob in img.find_blobs([threshold], pixels_threshold=100,
area_threshold=100, merge=True, margin=10):
        img.draw_rectangle(blob.rect())
        img.draw_cross(blob.cx(), blob.cy())
        img.draw_rectangle(r)

while(True):
    clock.tick()
    img = sensor.snapshot()
    for blob in img.find_blobs([threshold], pixels_threshold=100,
area_threshold=100, merge=True, margin=10):
        img.draw_rectangle(blob.rect())
        img.draw_cross(blob.cx(), blob.cy())
    print(clock.fps())
```

Code 20 Initial design of color recognition

Results:

However, in the observing of the later experiment, because the actual environment is complex, and the threshold range obtained by the img.get_histogram( ) is wide, and the error is large.

3.8.2 Final Design

In order to improve the accuracy of colour recognition, pre-setting the color threshold, and matching the target color with color threshold library is more appropriate. The color threshold will vary depending on the weather and time, so multiple sets of threshold data are used to select the final threshold according to the actual situation.
Sets of threshold data:

```
#sunny day
#thresholds=[(23, 32, -3, 15, -58, -22),     #pink
           #(42, 60, -27, 1, 17, 127),      #yellow
           #(49, 0, -128, 127, -128, 127)]  #blue
#cloudy
#thresholds=[(50, 100, 19, 127, -128, 127),  #pink
           #(54, 91, -38, -9, -128, 127),   #yellow
           #(31, 60, -20, 79, -69, -8)]     #blue
#final test
thresholds=[(60, 77, 20, 82, -73, 52),      #pink
           (58, 100, -73, -19, -49, 68),    #yellow
           (20, 68, -13, 17, -71, -21)]     #blue
```

Code 21 Thresholds for color block

To improve the accuracy of color recognition, the camera selects 40 images and identifies the colors that appear in the images, with the color being identified the most times as the last color to be determined.

Code:

```
for i in range(40):
    clock.tick()
    img = sensor.snapshot()
    for blob in img.find_blobs(thresholds, pixels_threshold=200,
area_threshold=200):
        img.draw_rectangle(blob.rect())
        img.draw_cross(blob.cx(), blob.cy())
        #print(blob.code(),type(blob.code()))
```

```
        if blob.code()==1:
            m=1
        elif blob.code()==2:
            m=2
        else:
            m=3
        n=n+m
        l=l+1
route=int(n/l+0.5)
print(n,l,route)
```

Code 22 Recognize the target color

Figure 53 Target color

When the color is recognized, the car is driven to the corresponding color block position. The first step is to position the color blocks by the function of blob.cx() and blob.cy() to get the coordinate of the center of the color block. In order to ensure the processing speed of the picture, the picture format uses QVGA, so the image is composed by 320 x 240 pixels. The center point of the car is at the (blob.cx(), blob.cy()) position, and the center of the color block is at the (160, 240) coordinates. By calculating the deflection angle between the two-point connection and the center of the image, the car's forward route can be obtained.

```
    for blob in img.find_blobs([thresholds1], pixels_threshold=20,
area_threshold=20, merge=True,margin=10):
        img.draw_rectangle(blob.rect())
        img.draw_cross(blob.cx(), blob.cy())
        img.draw_line(160,240,blob.cx(),blob.cy(),color=180, thickness=2)
        Intermediate_Variable = (160-blob[5])*(160-blob[5]) + (240-
blob[6])*(240-blob[6])
        Distance= math.sqrt(Intermediate_Variable)
        theta_err  =  math.asin((160-blob[5])/Distance)*(180/math.pi)     #
deflection angle
```

Code 23 Deflection angle

Figure 54 Route of color recognition

Color block in the picture pixel size can reflect the distance of the car from the color block, when there are 2000 pixels in the image, the car is seen as near to the color block, the use of the blob[4] function could acquire the size of the color block pixels in the image. This principle will also be applied to color beacon in patio2.

3.9 Image Group: Ultrasonic Sensor

Section Author **Shubo Yang**
(UESTC ID:2018190607005 UofG ID:2429400Y)

Ultrasonic sensors are essential for detecting obstacles and barriers for further commands making. They act like another pair of eyes besides the OpenMV camera. The distance feedback is used in determine further instructions, such as tracing along handrails, detecting signs, and detecting beacons. In this section, the mechanics of ultrasonic sensor HC-SR04 module are illustrated in details.

The ultrasonic sensor HC-SR04 module has four pins, that are, VCC, GND, ECHO, and TRIG, which is shown in Figure 54. The VCC pin is connected to 3.3V on OpenMV, while GND is bound to connect with Ground. TRIG is connected to an output pin, and ECHO is for an input pin.

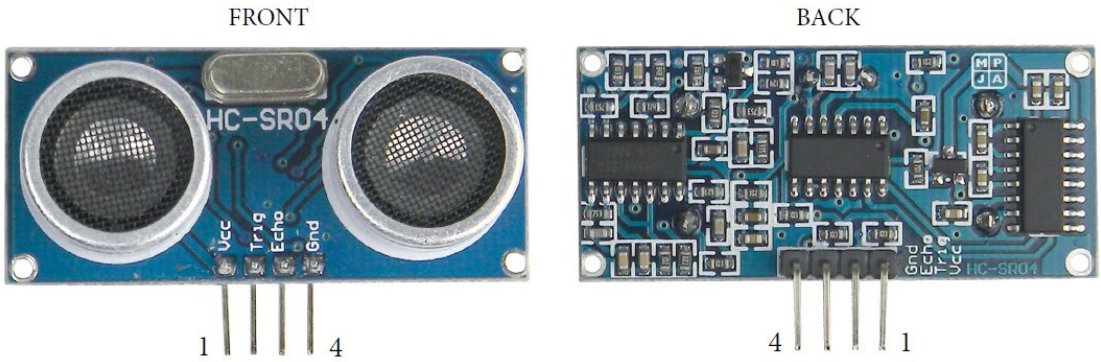


Figure 55 Ultrasonic sensor HC-SR04 with its pins.

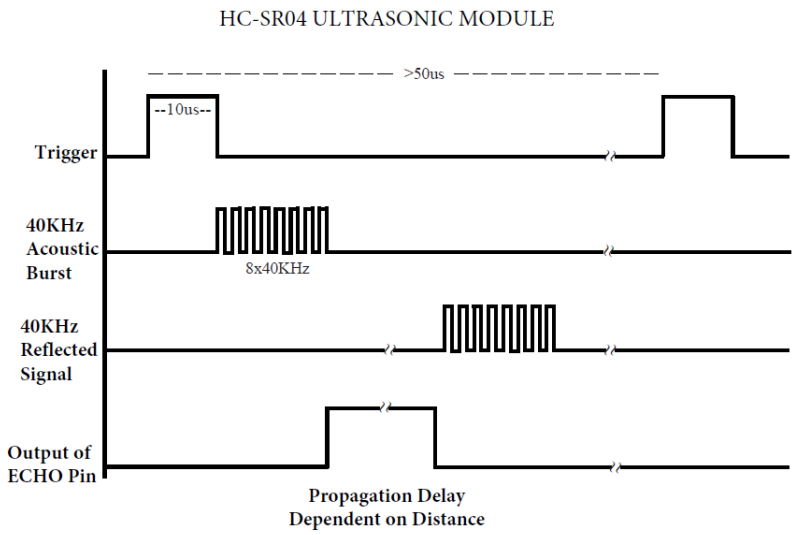


Figure 56 Mechanics of the ultrasonic sensor.

As in Figure 55, the mechanics of the ultrasonic sensor is presented. Initially, the Trig pin triggers a minimum 10us High level pulse. The sensor transmits an ultrasonic signal and receives the echo. The Echo pin is connected with an input pin on OpenMV, which should be set to wait the High level of the Echo pin. 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.

$$Distance = (ECHO\ high\ level\ time \times ultrasonic\ velocity)\ /2\ ,$$

where ultrasonic velocity means the speed of sound in air, that is, 340 m/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.

Additionally, the program is wrapped into a function called “wave_distance_process” in our program, which aims at conveniently calling the function. Consequently, whenever the function is called and the pins for the ECHO and TRIG are defined in the main program, the function “wave_distance_process” can be adopted to calculate the distance to the objects. The integrated code is in Code 13. The TRIG pin is first set to “0” for 5 us, then “1” for 15 us, and finally “0”, to trigger the sensor. Function “`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. Consequently, the distance is calculated.

```
def wave_distance_process(wave_echo_pin,wave_trig_pin):
    # Wave Start
    wave_trig_pin.value(0)
    utime.sleep_us(5)
    wave_trig_pin.value(1)
```

```
    utime.sleep_us(15)
    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
```

Code 24 Code for ultrasonic sensor.

In this way, the integrated function provides a simple, convenient, and explicit interface for the OpenMV to correctly measure the distances in any direction, as long as the pins for the sensors are properly defined.

3.10 Image Group: Clock Module

Section Author	**Shubo Yang** **(UESTC ID:2018190607005 UofG ID:2429400Y)**
Technically Assisted by	Zhangchen Xu (UESTC ID:2018190606034, UofG ID:2429391X)

The real-time clock (RTC) module DS3231 is adopted to return the value of current time for further HC-12 transmission. DS3231 is a low power, full binary-coded decimal (BCD) clock/calendar, with alarm output. Here, only the real-time function is utilized. The integrated function of DS3231 should return the accurate current time, including the date and time. The returned time would be used in the main program for HC-12 transmission.

As shown in Figure 56, the DS3231 module has six pins, but only four pins are considered for IIC communication with OpenMV, that are, SCL, SDA, VCC, and GND. SCL is the clock transmission line, and SDA is the data transmission line.

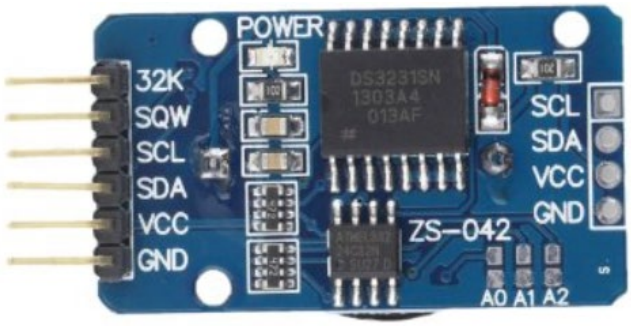


Figure 57 DS3231 module.

First, four pins should be connected with OpenMV I2C pins, and I2C port 4 is used. The current time should be set at first. A class named “DS3231” is created for further reference and function calling. The code is in Code 16. The variable “DS3231_time” stores a matrix that contains the date and current time.

```
DS3231_time=ds.DateTime()
year=DS3231_time[0]
month=DS3231_time[1]
day=DS3231_time[2]
weekday=DS3231_time[3]
hour=DS3231_time[4]
minute=DS3231_time[5]
```

Code 25 DS3231 code.

In this way, we can derive the specific time and write it on the UART of HC-12. Consequently, HC-12 can transmit the time.

3.11 Image Group: HC-12 Wireless Communication

Section Author **Zhangchen Xu**
(UESTC ID: 2018190606034, UofG ID: 2429391)

3.11.1 Module Overview

HC-12 communication module (see in Figure 58) is a half-duplex 20 dBm communication module with 100 channels in the 433.4-473.0 MHz range that is capable of transmitting up to 1 km. The maximum operating current is 100mA. The module has 4 pins. Apart from VCC and GND for power supply, the other two pins (TX and RX) are UART ports for communication between microcontroller and the module. When wiring, the TX of the HC-12 module should be connected to the RX of the microcontroller, and the RX should be connected to the TX, while ensuring that the two modules share a common ground (Figure 59).

Figure 58 HC-12 Module

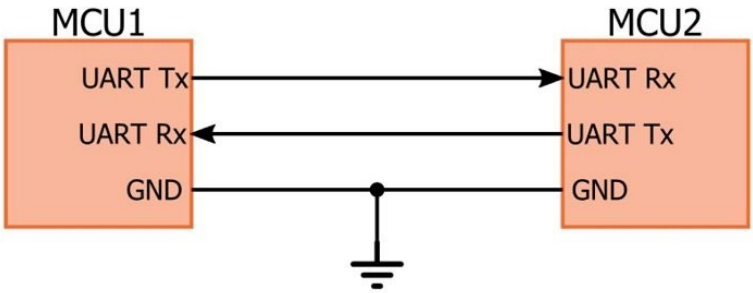


Figure 59 Wiring of HC-12

3.11.2 Send Message from Microcontroller

Due to the powerful functions from the microPython, after wiring, we can easily send message to the computer through three lines of codes as shown in Code 26. The first two lines are for instantiate UART port and initialize parameters. Note that the default setting of HC-12 module is 9600 baud rate with no parity and 1 stop bit. We should initialize the UART in accordance with these parameters. Then we can directly send message using .write command.

```
# HC-12 UART Settings
UART_hc12=pyb.UART(1, 9600, timeout_char=1000)
UART_hc12.init(9600, bits=8, parity=None, stop=1, timeout_char=1000)

UART_hc12.write("hello world")
```

Code 26 Send Message from HC-12

3.11.2 Receive Message

Compared to send messages, it's much more complicated when receiving messages. There are solutions such as repeat reading messages using UART .read() command in while loop,

however, it may waste a lot of time. Also, the efficiency is low as there is no message in most of the time. As a result, we use interrupt as the solution, which is more convenient and can save computing resources.

The codes for receiving messages are shown in Code 27. At first, we need to define an IRQ with a trigger (i.e., the RX port is idle). After that, a handler is defined where the reading and decoding takes place. Once the program enters the interrupt, it will read the message, decode and print the message. Here, we use a global variable hc12_indicator, aiming to provide an acknowledgement of the message received for the main program and is used in Patio 2.

```
# HC-12 Communication Interrupt
def UART_HC12_ISR(t):
    global hc12_indicator
    msg_hc12=UART_hc12.readline()
    if msg_hc12 != None:
        msg_hc12=msg_hc12.decode('utf-8')
        print("Receive HC-12 Message:"+str(msg_hc12))
        hc12_indicator=hc12_indicator+1
    return

## HC-12 UART Interrupt
UART_hc12.irq(trigger = pyb.UART.IRQ_RXIDLE, handler = UART_HC12_ISR)
```

Code 27 HC-12 Receive Messages

3.12 Image Group: Beacon Design

3.12.1 Patio 1 Apriltag Beacon

Section Author	**Ziyi Xie** **(UESTC ID:2018190502024, UofG ID:2429497X)**

Purpose:

In order to go straight through the bridge for patio1. We need to know when to sending turning instructions and the accurate turning angle. Apriltag could satisfy our requirements as it can tell the 3d dimensional coordinates to adjust the car position.

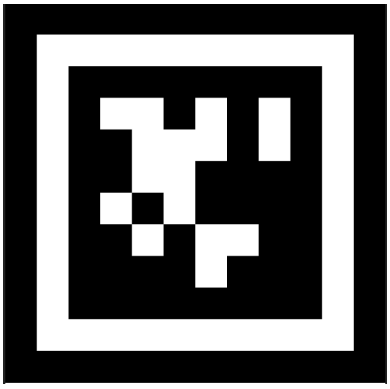


Figure 60 Apriltag Beacon

Procedure:

1、Measure the constant parameter for the real distance and the image pixel:
2、Print the certain id Apriltag and set the beacon.
3、Get the distance in the z direction which is the horizontal distance between the beacon and the car.
4、Get the rotation angle of the beacon seen from the camera. Using this angle to adjust the car and then turning a accurate angle based on the gyroscope. After that, the car could go straight through the bridge.

```
# AprilTags Example
#
# This example shows the power of the OpenMV Cam to detect April Tags
# on the OpenMV Cam M7. The M4 versions cannot detect April Tags.

import sensor, image, time, math

sensor.reset()
sensor.set_pixformat(sensor.GRAYSCALE)
sensor.set_framesize(sensor.QQVGA) # we run out of memory if the resolution
is much bigger...
sensor.skip_frames(15)
sensor.set_auto_gain(False)   # must turn this off to prevent image
washout...
sensor.set_auto_whitebal(False)  # must turn this off to prevent image
washout...
clock = time.clock()

f_x = (2.8 / 3.984) * 320
f_y = (2.8 / 2.952) * 240
c_x = 320 * 0.5 # (image.w * 0.5)
c_y = 240 * 0.5 # (image.h * 0.5)
```

```
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): # 默认为 TAG36H11
        img.draw_rectangle(tag.rect(), color = (255, 0, 0))
        img.draw_cross(tag.cx(), tag.cy(), color = (0, 255, 0))
        rotation_simple=degrees(tag.rotation())
        print_args = (tag.x_translation(), tag.y_translation(), tag.z_translation(), \
            degrees(tag.x_rotation()), degrees(tag.y_rotation()), degrees(tag.z_rotation()))
        #rotation is 0-360 , from left（1°） to right（360）
        #print("Tx: %f, Ty %f, Tz %f, Rx %f, Ry %f, Rz %f" %


        print_args)
        print(tag.id(),rotation_simple,tag.z_translation())

    print(clock.fps())
```

Code 28 Apriltag Example

3.12.2 Patio 2 Color Beacon

In patio 2, the beacon is placed at the position of the feeding port, which is used to make sure that the car could stop at the specific position and complete the task 2, feeding. Because the car moves to this position with the help of ultrasonic sensor, the car can not only use ultrasonic to identify beacon when selecting beacon. Thus, the camera and the ultrasonic are expected to jointly detect the beacon. Only when the distance measured by the ultrasonic is less than 40 cm, and the yellow beacon is recognized at the same time, the car stops and starts feeding operation.

Figure 61 Beacon for Patio 2

3.13 Inter-board Communication

Section Author **Zhangchen Xu**
(UESTC ID: 2018190606034, UofG ID: 2429391)

Inter-board Communication is vital in our system as it's composed of two boards. The STM32F411 board is responsible for controlling the movement while OpenMV is responsible for giving orders. In order for the car to move according to OpenMV's commands, a highly reliable inter-board communication protocol needs to be constructed.

3.13.1 The Protocol

We choose **UART** as the communication protocol. Compared to other serial communication protocols such as SPI or IIC, it has the advantages of few line requirements (only three lines, TX, RX and ground are required, no clock is needed), and as mentioned in previous sections, microPython's UART functions are simple and reliable. However, UART has the limitations such as slow speed and small data frame, but it's not important in our project as the command from OpenMV to STM boards are short (less than 7-bit).

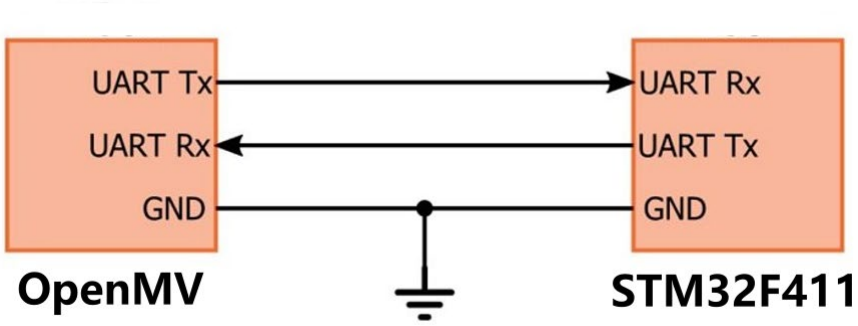


Figure 62 Inter-board Communication Wiring

Code 29 shows the codes for inter-board communication (OpenMV side). The design is similar with that of HC-12 communication with direct writing and interrupt reading. In order to achieve high transmission speed, we set the inter-board communication baud rate to 115200.

```
# Interboard Communication Interrupt
def UART_STM_ISR(t):
    msg_stm=UART_stm.readline() # To avoid reflection
    msg_stm=UART_stm.readline()
    if msg_stm != None:
        msg_stm=msg_stm.decode('utf-8')
        print("STM32 Message: "+str(msg_stm))
    return
```

```
## Inter-Board Communication UART Settings
UART_stm=pyb.UART(3)
UART_stm.init(115200, bits=8, parity=1, stop=1, timeout_char=100)

## Inter-Board Comm UART Interrupt
UART_stm.irq(trigger = pyb.UART.IRQ_RXIDLE, handler = UART_STM_ISR)
```

Code 29 Codes for Inter-board Communication

3.13.2 Coding Scheme

Due to the different requirements of the tasks, OpenMV may give the STM32 different kinds of instructions (e.g., turn 90 degree, go straight, stop). We developed a coding scheme with maximum 7 bit which is readable by both OpenMV and STM32. It contains 6 different modes for different kinds of tasks, and bit 1 is the Mode Bit. As shown in Figure 63, after receiving the command, STM32 can easily distinguish which operation it need to do by reading the mode bit.

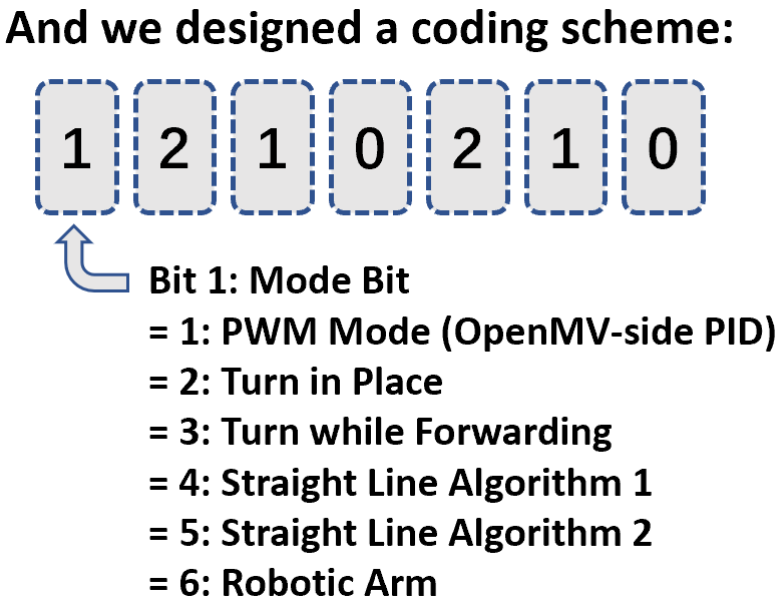


Figure 63 Coding Scheme

Table 5 shows detailed encoding specification for different modes. Let's take Mode 1, PWM Mode as an example. In this mode, OpenMV will send a 7-bit message to the STM32, where Bit 2-4 contains the speed and direction of left wheels, while Bit 5-7 contains information of right wheels.

```
Bit 1: Mode Bit
        Mode = 0: STOP   0
        Mode = 1: PWM Mode  e.g., 1210210 PWM10, Go forward
```

```
                Continuous Control by PID in OpenMV
                Bit 2-4: Left PWM
                        Bit 2 = 1: Negative PWM (backward)
                        Bit 2 = 2: Positive PWM (forward)
                        Bit 3-4: Magnitude(00-99)
                Bit 5-7: Right PWM
                        Bit 5 = 1: Negative PWM (backward)
                        Bit 5 = 2: Positive PWM (forward)
                        Bit 6-7: Magnitude(00-99)
        Mode = 2: Revolving on the spot  e.g., 21090 Turn left for 90
degree
                Bit 2-5: Angle Value
                        Bit 2 = 1: Turn Left
                        Bit 2 = 2: Turn Right
                        Bit 3-5: 0-360
        Mode = 3: Turn while Moving  e.g., 31020 turn left for 20 degree
while moving
                speed is defined by the car
                Bit 2-5: Angle Value
                        Bit 2 = 1: Turn Left
                        Bit 2 = 2: Turn Right
                        Bit 3-5: 0-360
        Mode = 4: Straight Line Algorithm 1  e.g. 4
        Mode = 5: Straight Line Algorithm 2  e.g. 5
      Mode = 6: Robotic Arm  e.g. 6
```

Table 5 Encoding Specification

4 System Integration, Results and Discussion

4.1 Command Execution System (Car Group)

4.1.1 System Integration

4.1.1.1 Hardware

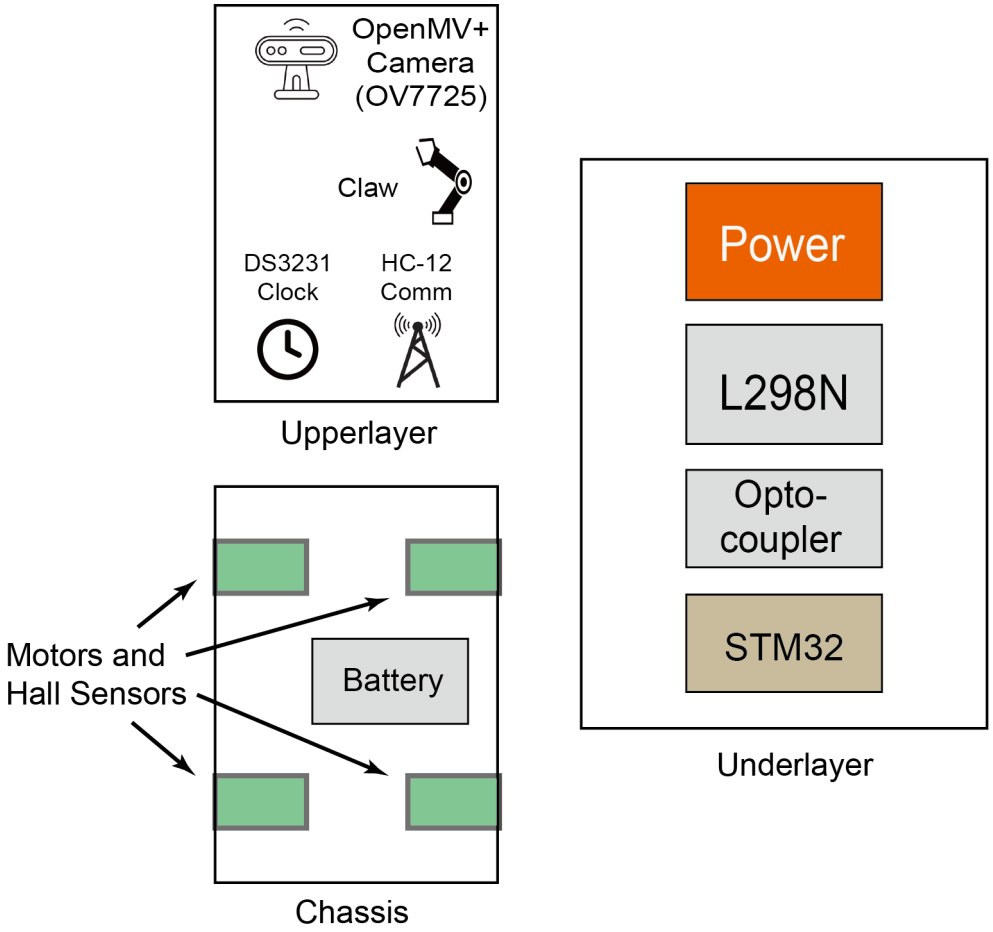


Figure 64 The Display of Modules in Three Layers

The main hardware components are illustrated in the above graph (Figure 64), including motor, hall element, L298N module, STM32 MCU, battery, HC-12 and claw. Note that one invisible module optical coupler module is between STM32 boards and L298N boards, it isolates the control circuit and drive circuit in electricity improving the safety and stability of MCU, which avoid the potential overvoltage or overcurrent in PWM value switching stage. The addition of the optical coupler module is because during our experimentation process, the high current overload easily breaks down STM32 boards. After several debugging steps and trials of changing boards and hardware, we find that the module is the optimal choice to entirely

eliminate the risk of STM malfunction.

4.1.1.2 Software: Mode Realization

The `main` function in stm32 is designed to contain several modes, each representing a different type of movement, for the convenience of external instructions. The modes are integrated in the same while loop according to their mode codes, which is illustrated in the figure below:

Bit 1: Mode Bit
- Mode = 0: STOP **0**
- Mode = 1: PWM Mode **1**210210 PWM10, Go forward
  - Continuous Control by PID in OpenMV
  - Bit 2-4: Left PWM
    - Bit 2 = 1: Negative PWM (backward)
    - Bit 2 = 2: Positive PWM (forward)
    - Bit 3-4: Magnitude(00-99)
  - Bit 5-7: Right PWM
    - Bit 5 = 1: Negative PWM (backward)
    - Bit 5 = 2: Positive PWM (forward)
    - Bit 6-7: Magnitude(00-99)
- Mode = 2: Revolving on the spot **2**1090 Turn left for 90 degree
  - Bit 2-5: Angle Value
    - Bit 2 = 1: Turn Left
    - Bit 2 = 2: Turn Right
    - Bit 3-5: 0-360
- Mode = 3: Turn while Moving **3**1020 turn left for 20 degree while moving
  - speed is defined by the car
  - Bit 2-5: Angle Value
    - Bit 2 = 1: Turn Left
    - Bit 2 = 2: Turn Right
    - Bit 3-5: 0-360
- Mode = 4: Straight Line Algorithm 1 **4**
- Mode = 5: Straight Line Algorithm 2 **5**
- Mode = 6: Robotic Arm **6**

Figure 65 Mode illustration

The decision which mode to enter is made by OpenMV, whose messages specifies the mode at the first bit. Since the messages, named `sum`, are in the syntax of string, they are easily truncated, whereby instructions are extracted. For example, the command bit is obtained as:

```
command_bit=int(sum[0:1])
```

Code 30 command_bit

a. **Mode 0: Stop**

Mode 0 brakes the car, giving all pins on L298N high input. This enables the car to make an abrupt stop.

```
i.  # mode0: stop
    1. if command_bit==0:
    2. Car.run_(0,0,0,0,0,0,0,0)
```

Code 31 Mode 0

b. **Mode 1: Receiving pwm control from OpenMV**

In this mode, the car receives all pwm controls from OpenMV, including pwm values and the angular velocity directions of each wheel. Basically, two wheels on the same side (i.e. left or right) are always given the same instructions for simplicity. Seven bits are needed in this mode, the first for mode switch and the others for car motion. Bits 2 to 4 give instructions to left wheels. A 2 on bit 2 indicates the left wheels turn forward and a 1 on this bit indicates backward movement. Bit 3 to 4 gives a pwm as a decimal figure with two digits, ranging from 0 to 99. The same rule is obeyed in bit 5 to 7 for right wheels.

```
# mode1: receiving PWM from OpenMV
    elif command_bit==1:
        if len(sum)!=7: #if wrong pattern received, move forward
            sum="1220220"
        else:
            left=int(sum[1:4])
            if left>200:
                NlB1=left-200
                NlB2=0
                NlC1=left-200
                NlC2=0
            else:
                NlB2=left-100
                NlB1=0
                NlC2=left-100
                NlC1=0
            right=int(sum[4:7])
            if right>200:
                NrA1=right-200
                NrA2=0
                NrD1=right-200
                NrD2=0
            else:
                NrD2=right-100
                NrD1=0
```

```
                    NrA2=right-100
                    NrA1=0
            Car.run_(NrA1,NrA2,NlB1,NlB2,NlC1,NlC2,NrD1,NrD2)
```

Code 32 Mode 1

c. **Mode 2: Turn on the spot**

In mode 2, the car complete turning instructions of arbitrary angle. This requires a command message of five bits, in which bit 2 implies whether the turning is clockwise or counterclockwise, and bit 3 to 5 gives the degree of turning. On entering mode 2, we set the global variable count to be zero, indicating an original angle is read and recorded from JY901S. While the turning is not completed, the program would stay in the while loop and new commands about modes are ignored, if and, during this period.

```
    # mode2:turning withour moving along
        elif command_bit==2:
            if len(sum)!=5:
                sum="21000" #if wrong pattern, stay still
            else:
                if int(sum[1:2])==1: #counterclockwise
                    turn_angle=int(sum[2:5])    # >0: anticlockwise,
<0: clockwise
                if int(sum[1:2])==2: #clockwise
                    turn_angle=-int(sum[2:5])
                count=0
                time.sleep(0.5)
                turn_complete=0
                target_angle=original_angle+turn_angle
                if target_angle<=-180:
                     target_angle=target_angle+360
                if target_angle>180:
                     target_angle=target_angle-360
                while(turn_complete==0):
                  if    z_angle<=-90    and    z_angle>=-180    and
target_angle<=180 and target_angle>=90:
                     delta=z_angle-target_angle+360
                  elif    z_angle>=90    and    z_angle<=180    and
target_angle>=-180 and target_angle<=-90:
                     delta=z_angle-target_angle-360
                  else:
                     delta=z_angle-target_angle
                     if z_angle-target_angle>=180:
                         delta=-(360-delta)
                     if z_angle-target_angle<=-180:
```

```
                    delta=-(360+delta)
                # print(delta,z_angle)
                if abs(delta)>=1:
                    print(delta,z_angle)
                    turn(delta)
                if abs(delta)<1:
                    # print(delta)
                    turn_complete=1
                    # Car.run_(0,0,0,0,0,0,0,0)
                sum="0"
```

Code 33 Mode 2

d. **Mode 3: Turning while moving**

Instead of making an accurate turning in place, in this mode, the car may move on and meanwhile adjust its direction according to the delta angle returned from OpenMV, characterizing the difference between its current orientation and the target one. The message for this mode consists of five bits, with one command bit, one bit indicating the direction of angular velocity of turning and three bits telling the absolute value of the delta angle.

```
# mode3: move while turning
    elif command_bit==3:
        if len(sum)!=5:
            sum="31000" #if wrong pattern, move forward
        else:
            if int(sum[1:2])==1:
                str_angle=int(sum[2:5])
            elif int(sum[1:2])==2:
                str_angle=-int(sum[2:5])
        Car.straight(40,40,40,40,str_angle)
```

Code 34 Mode 3

The essence of this mode is the function it calls named straight. In this function, we first decide whether the angle is valid. If it is too large, we ignore the instruction because in this mode the car is supposed to move in a relatively straight manner. If the returned angle lies between an acceptable interval, we adjust the speed of wheels to cater for the degree expected. The adjustment is based on the angle received. If the absolute value of the angle is below ten degrees, we make fine adjustments. Otherwise, we adjust one side of wheels to turn backwards and the other two to turn forward. For example, if we want the car to turn left, we instruct the right wheels to turn forward and the left ones to turn backward. In this way, our car can adjust its direction according to the instructions from OpenMV.

```
def straight(self, pwm_A, pwm_B, pwm_C, pwm_D, sum):
      self.NrA1 = pwm_A #NrA1=NrD1(same speed for wheels on the
```

```
same side)
          self.NlB1 = pwm_B
          self.NlC1 = pwm_C
          self.NrD1 = pwm_D
          step = 10
          wt=0.001
          v_base=40
          if(sum<70 and sum>-70):
             # increment=(sum/abs(sum))*math.exp(sum/2.2)
             increment=sum/2
             # increment=1

             if (sum<=10 and sum>=-10):
                 self.NrD1=v_base+increment
                 self.NrA1=v_base+increment
                 self.NlB1=v_base-increment
                 self.NlC1=v_base-increment
                 self.NrA2=0
                 self.NlB2=0
                 self.NlC2=0
                 self.NrD2=0
             if sum>10:
                 self.NrA1=v_base+increment
                 self.NrD1=v_base+increment
                 self.NlC2=v_base+increment
                 self.NlB2=v_base+increment
                 self.NrA2=0
                 self.NrD2=0
                 self.NlC1=0
                 self.NlB1=0
             if sum<-10:
                 self.NrA2=v_base-1.5*increment
                 self.NrD2=v_base-1.5*increment
                 self.NlC1=v_base-1.5*increment
                 self.NlB1=v_base-1.5*increment
                 self.NrA1=0
                 self.NrD1=0
                 self.NlC2=0
                 self.NlB2=0
          if(sum>70 or sum<-70):
             self.NrA2=0
             self.NlB2=0
```

```
            self.NlC2=0
            self.NrD2=0
          Car_motion.run_(self,   self.NrA1,self.NrA2,   self.NlB1,
self.NlB2, self.NlC1, self.NlC2, self.NrD1, self.NrD2)
          if abs(sum)>10:
            sleep(0.2)
          else:
            sleep(0.3)
          print(sum,'FL=',self.NlB1-self.NlB2,'FR=',self.NrA1-
self.NrA2,'BL=',self.NlC1-self.NlC2,'BR=',self.NrD1-self.NrD2)
          Car_motion.run_(self, 0, 0, 0, 0, 0, 0, 0, 0)
       sleep(0.5)
```

Code 35 Function straight

e. **Mode 4: Moving in a straight line on pebble ground**

In some parts in patio 2, the car is desired to move in a straight line. However, it is difficult for the car to keep direction on pebble ground, where its wheels may rotate without touching the ground, making it futile for control over wheel speeds. Thus, we designed a mode especially for this scenario, in which JY901S is invoked. To enter this mode, the message from OpenMV requires a mere command bit.

```
    # mode4: move along a stright line (patio 2)
        elif command_bit==4 and isInitialize==0:
            turn_angle=0
            count=0 #set another original angle
            time.sleep(0.5)
            a=[31,30,30,31]
            target_angle=s_original_angle+turn_angle
            isInitialize=1
            if target_angle<=-180:
                target_angle=target_angle+360
            if target_angle>180:
                target_angle=target_angle-360

        elif command_bit==4 and isInitialize==1:
            if     z_angle<=-90     and     z_angle>=-180     and
target_angle<=180 and target_angle>=90:
                delta=z_angle-target_angle+360
            elif z_angle>=90 and z_angle<=180 and target_angle>=-
180 and target_angle<=-90:
                delta=z_angle-target_angle-360
            else:
                delta=z_angle-target_angle
```

```
                if z_angle-target_angle>=180:
                    delta=-(360-delta)
                if z_angle-target_angle<=-180:
                    delta=-(360+delta)
             # print(delta,z_angle)

          a=Car.stra2(a[0],a[1],a[2],a[3],delta)
```

Code 36 Mode 4

The function `stra2` is called here to fulfill this task. This function adjusts the pwm value for wheels based on its last values. The code for it is listed below:

```
    def stra2(self, pwm_A, pwm_B, pwm_C, pwm_D, sum):
          self.NrA1 = pwm_A #NrA1=NrD1(same speed for wheels on the
same side)
          self.NlB1 = pwm_B
          self.NlC1 = pwm_C
          self.NrD1 = pwm_D
          self.NrA2=0
          self.NlB2=0
          self.NlC2=0
          self.NrD2=0
          yuzhi=0.1
          increment=abs(sum)
          rang=40#maximum differnece between speeds on two sides
          base=30
          if(sum<-yuzhi):
                self.NrA1=self.NrA1+increment
                self.NrD1=self.NrD1+increment
                self.NlC1=self.NlC1-increment
                self.NlB1=self.NlB1-increment
                if((self.NrA1-self.NlB1)>rang):
                   self.NlB1=base-(rang/2)
                   self.NlC1=base-(rang/2)
                   self.NrA1=base+(rang/2)
                   self.NrD1=base+(rang/2)
          if(sum>yuzhi):
                self.NlB1=self.NlB1+increment
                self.NlC1=self.NlC1+increment
                self.NrA1=self.NrA1-increment
                self.NrD1=self.NrD1-increment
                if((self.NlB1-self.NrA1)>rang):
                   self.NlB1=base+(rang/2)
```

```
                    self.NlC1=base+(rang/2)
                    self.NrA1=base-(rang/2)
                    self.NrD1=base-(rang/2)
          if(sum<=yuzhi and sum>=-yuzhi):
             pass
          Car_motion.run_(self,   self.NrA1,self.NrA2,   self.NlB1,
self.NlB2, self.NlC1, self.NlC2, self.NrD1, self.NrD2)
          print(sum,'FL=',self.NlB1-self.NlB2,'FR=',self.NrA1-
self.NrA2,'BL=',self.NlC1-self.NlC2,'BR=',self.NrD1-self.NrD2)
          sleep(0.1)
      return (self.NrA1, self.NlB1, self.NlC1, self.NrD1)
```

Code 37 Function stra2

f. Mode 5: Moving in a straight line across the bridge

When crossing the bridge, the car needs to follow a straight trace as well. However, our function `stra2` is designed to be sensitive to small degree changes, and the car might behave like swinging from side to side at times, making it not suitable for the bridge of a limited width. Mode 5 is designed for this task and performs smooth motion.

```
   # mode5: move along a straight line (bridge)
       elif command_bit==5 and isInitialize==0:
           turn_angle=0
           count=0 #set another original angle
           time.sleep(0.5)
           target_angle=s_original_angle+turn_angle
           if target_angle<=-180:
               target_angle=target_angle+360
           if target_angle>180:
               target_angle=target_angle-360
           isInitialize=1
           print(isInitialize)

       elif command_bit==5 and isInitialize==1:
           if     z_angle<=-90     and     z_angle>=-180     and
target_angle<=180 and target_angle>=90:
               delta=z_angle-target_angle+360
           elif z_angle>=90 and z_angle<=180 and target_angle>=-
180 and target_angle<=-90:
               delta=z_angle-target_angle-360
           else:
               delta=z_angle-target_angle
               if z_angle-target_angle>=180:
                   delta=-(360-delta)
```

```
            if z_angle-target_angle<=-180:
                delta=-(360+delta)
        a=Car.stra3(delta)
        global y_angle
#        print("y_angle:",y_angle)
        if(y_angle>10):
            enableCamera=1
            print("sent enable camera.")
        UART_OpenMV.write(str(enableCamera))
```

Code 38 Mode 5

One of the main differences between mode 5 and mode 4 is that in mode 5, stm32 returns a message to turn on the camera and informs OpenMV that it is going down the bridge. In addition, another function, namely `stra3` is called in this mode. This function is not iterative, ensuring a steady performance of the car.

```
    def stra3(self,delta):
        self.NrA1 = 36 #NrA1=NrD1(same speed for wheels on the
same side)
        self.NlB1 = 35
        self.NlC1 = 35
        self.NrD1 = 36
        step = 10
        wt=0.001
        v_base=35
        sum=delta
        if (sum==0):
           pass
        if(sum>0.1 or sum<-0.1):
           # increment=(sum/abs(sum))*math.exp(sum/2.2)
           increment=40*sum
           self.NrD1=v_base-increment
           self.NrA1=v_base-increment
           self.NlB1=v_base+increment
           self.NlC1=v_base+increment
           # self.NrA2=0
           # self.NlB2=0
           # self.NlC2=0
           # self.NrD2=0
        else:
           pass
        self.NrA2=0
        self.NlB2=0
```

```
          self.NlC2=0
          self.NrD2=0
          if(self.NrA1>50):
             self.NrA1 = 50#NrA1=NrD1(same speed for wheels on
the same side)
             self.NrD1 = 50
          if(self.NrA1<20):
             self.NrA1 = 20#NrA1=NrD1(same speed for wheels on
the same side)
             self.NrD1 = 20
          if(self.NlB1>50):
             self.NlB1 = 50#NrA1=NrD1(same speed for wheels on
the same side)
             self.NlC1 = 50
          if(self.NlB1<20):
             self.NlB1 = 20#NrA1=NrD1(same speed for wheels on
the same side)
             self.NlC1 = 20
          Car_motion.run_(self, self.NrA1,self.NrA2, self.NlB1,
self.NlB2, self.NlC1, self.NlC2, self.NrD1, self.NrD2)
          # sleep(0.2)
          print(sum,'FL=',self.NlB1-self.NlB2,'FR=',self.NrA1-
self.NrA2,'BL=',self.NlC1-self.NlC2,'BR=',self.NrD1-self.NrD2)
```

Code 39 Function stra3

g. **Mode 6: Operating the robotic arm**

If OpenMV returns a single '6', the car enters the mode of robotic arm operation. This is illustrated in detail in the parts of robotic arm. The code for mode 6 is simple, since most work is finished in the function `robotic_arm.releasing()`. After the execution of this function, we automatically set the command bit to 0 so that the car would stop for a while and wait for new instructions.

```
    elif command_bit==6:
           robotic_arm.releasing()
        sum="0"
```

Code 40 Mode 6

4.1.2 Result and Discussion

4.1.2.1 Patio1

Tracing Response

How to make car go straight is an issue that have troubled us for a long time. The key factor of the task is to ensure the running route be an approximately straight line within tolerate drift angle as 0.1. It is critical in both patio1 and patio 2.

The first solution we tried was to use the velocity feedback of the wheels to enable the right and left sides of the wheel the same speed by setting the PID with reasonable parameters. In other words, it is to walk in a straight line. Based on this, we converted the number of pulses received by the encoder into velocity as well as added the PID algorithm to achieve the same speed on both sides.

However, the fact is not what we imagined. Even if we can make the rotation speed on both sides of the wheel exactly and quickly, due to the unevenness of the ground, the accidental idling will cause the deviation of the car. Especially on the cobblestone pavement of the second patio.

Figure 66 Cobblestone pavement

The second attempt was to use a gyroscope to accurately perceive the angle of the current car, and adjusted the output values of the PWM motors on both sides by analyzing the angle offset. The simplified version of the logic is as follows: if the angle is greater than 0.1, the car is tilted to the left, then we will increase the PWM output of the left wheel; if the angle is less than 0.1, the car is tilted to the right, then we will increase the PWM output of the right wheel. The first gyroscope we tried was MPU6050, (the code above, the part with the problem). However, due to the long time required to obtain the speed of this chip, we could not get the offset of the small angle in time, so we finally gave up this scheme.

The final solution: Since the change of PWM should be related to the change of angle, by comparing three functional relationships, including linear function, ln function and exponential function, we found that the exponential function changes little when the angle is small, and the changes are obviously when the angle is large, This feature satisfies our idea of fine-tuning in small angles and drastically adjusting in large angles. In addition, we have tried two different logic methods. We found that adding negative feedback (that is, the current PWM value is the

last PWM value) and limiting the maximum PWM difference between the two wheels of the trolley will make the car adjust more quickly and the offset will be smaller when the car goes straight. The car has outstanding performance on the cobblestone road.

```
import pyb
import MPU6050
 '''
    input parameters: current enc
   output parameters: current pwm value
'''
 class PID:
    def __init__(self, pwm_range, kp_A, ki_A, kd_A, kp_B, ki_B,
kd_B, kp_C, ki_C, kd_C, kp_D, ki_D, kd_D):
        #pwm_range: maximum pwm value
        #A:B8-B7, front-right
        #B:B6-B5, front-left
        #C:B1-A6, rear-left
        #D:B0-A7, rear-right
        pwm_A = 0 #pwm_A = Nr1-Nr2
        pwm_B = 0
        pwm_C = 0
        pwm_D = 0
        err = 0
       err_A = 0
        err_B = 0
        err_C = 0
        err_D = 0
        last_err_A = 0
        last_err_B = 0
        last_err_C = 0
        last_err_D = 0
        self.pwm_range = pwm_range
        self.kp_A = kp_A
        self.ki_A = ki_A
        self.kd_A = kd_A
        self.kp_B = kp_B
        self.ki_B = ki_B
        self.kd_B = kd_B
        self.kp_C = kp_C
        self.ki_C = ki_C
        self.kd_C = kd_C
        self.kp_D = kp_D
```

```
        self.ki_D = ki_D
        self.kd_D = kd_D
        self.pwm_A = 0
        self.pwm_B = 0
        self.pwm_C = 0
        self.pwm_D = 0
        self.err = err
        self.err_A = err_A
        self.err_B = err_B
        self.err_C = err_C
        self.err_D = err_D
        self.last_err_A = last_err_A
        self.last_err_B = last_err_B
        self.last_err_C = last_err_C
        self.last_err_D = last_err_D

    # increment pid
     def incremental_pid(self, now, target):
         pwm += Kp[e(k) - e(k-1)] + Ki*e(k) + Kd[e(k) - 2e(k-1) +
e(k-2)]

        '''
         err = target - now #now:'count'
        pwm = pwm + self.kp*(err - last_err) + self.ki*err +
self.kd*(err - last_err)
         if (pwm >= self.pwm_range): # limit the amplitude
             pwm = self.pwm_range
         if (pwm <= -self.pwm_range):
             pwm = -self.pwm_range
         last_err = err
         return pwm

    def pid_A(self, now, target):
         self.err_A = target - now
         self.pwm_A = self.pwm_A + self.kp_A * ( self.err_A -
self.last_err_A ) + self.ki_A*self.err_A + self.kd_A*(self.err_A -
self.last_err_A)
         if (self.pwm_A >= self.pwm_range):
             self.pwm_A = self.pwm_range
         if (self.pwm_A <= -self.pwm_range):
             self.pwm_A = -self.pwm_range
         self.last_err_A = self.err_A
```

```
        # print("pwm_A", self.pwm_A)
        return self.pwm_A

    def pid_B(self, now, target):
        self.err_B = target - now
        self.pwm_B   =   self.pwm_B   +   self.kp_B*(self.err_B   -
self.last_err_B) + self.ki_B*self.err_B + self.kd_B*(self.err_B -
self.last_err_B)
        if (self.pwm_B >= self.pwm_range):
            self.pwm_B = self.pwm_range
        if (self.pwm_B <= -self.pwm_range):
            self.pwm_B = -self.pwm_range
        self.last_err_B = self.err_B
        return self.pwm_B

    def pid_C(self, now, target):
        self.err_C = target - now
        self.pwm_C   =   self.pwm_C   +   self.kp_C*(self.err_C   -
self.last_err_C) + self.ki_C*self.err_C + self.kd_C*(self.err_C -
self.last_err_C)
        if (self.pwm_C >= self.pwm_range):
            self.pwm_C = self.pwm_range
        if (self.pwm_C <= -self.pwm_range):
            self.pwm_C = -self.pwm_range
        self.last_err_C = self.err_C
        return self.pwm_C

    def pid_D(self, now, target):
        self.err_D = target - now
        self.pwm_D   =   self.pwm_D   +   self.kp_D*(self.err_D   -
self.last_err_D) + self.ki_D*self.err_D + self.kd_D*(self.err_D -
self.last_err_D)
        if (self.pwm_D >= self.pwm_range):
            self.pwm_D = self.pwm_range
        if (self.pwm_D <= -self.pwm_range):
            self.pwm_D = -self.pwm_range
        self.last_err_D = self.err_D
        return self.pwm_D
```

code 41 car dynamic self-adaption direction maintenance

In order to ensure that the car can patrol the route steadily, we have tried many methods. At first, we decided to let OpenMV transmit the angle between the central axis and the car to STM32 through inter-board communication (the distance between the trolley and the center

line is not transmitted), but the effect is not good: After a good number of trials, it is easy for the car to deviate from the route it needs to follow.

Next, we tried two straight-line algorithms, the exponential function algorithm with negative feedback (mode 4) and the linear function algorithm (mode 5). When we call mode4, the probability of error is small when the car is patrolling the line, but the car twists and turns when it moves forward, which affects the speed of the car and wastes a lot of time. When we call mode 5, the car can complete a short distance perfectly, but when it needs to travel a long distance, it will deviate a lot from the central axis.

After many trials by our team members, we have come up with a relatively rigorous solution: command OpenMV to transmit an 8-bit command, Bit1 is 1 means that we will use PWM Mode (continuous control by PID in OpenMV) Bit2-7 directly transfers the PWM value information to the four wheels, which has achieved good results. Since the exact PWM value makes our control of the wheels more precise, avoiding the situation that the car greatly deviates from the route.

Figure 67 the car is patrolling the line while turning

In-Position Turning

In Patio 1, there are two points where each a 90 degree in-position turn is required. One is at the beacon before the bridge, the other one is right after crossing the bridge.

At first point, once the beacon is detected, Open MV sends signal “22090” to STM32, ordering Mode 2 (in-position turning) and a clockwise 90 degree turn. Receiving the order, STM32 first gets an original angle from gyroscope and sets a target angle accordingly. In the turning process, gyroscope keeps reading current angle for STM32 to make comparison with the target one. We set the tolerance to be ±1 degree, i.e., the turning process is finished once |current - target|≤ 1. The second turn is carried out in a similar way.

In field tests, we find that the car often turns over 90 degree and figure out the reason: the area around turning points is smooth enough for the car to make turns rapidly, so even though the

gyroscope is real-time and accurate, it still turns over easily due to inertia. To fix such fault, we set the car to turn at a lower speed when the turning is about to be complete (|current - target|≤ 20).

```
def turn(delta):
    if delta>0:                         #clockwise
        if delta>20:
            Car.run_(0,50,50,0,50,0,0,50)
            time.sleep(0.4)
        if delta<20:
            Car.run_(0,30,30,0,30,0,0,30)
            time.sleep(0.2)
            Car.run_(0,0,0,0,0,0,0,0)
            time.sleep(0.1)
    if delta<0:                         #anticlockwise
        if delta<-20:
            Car.run_(50,0,0,50,0,50,50,0)
            time.sleep(0.4)
        if delta>-20:
            Car.run_(30,0,0,30,0,30,30,0)
            time.sleep(0.1)
            Car.run_(0,0,0,0,0,0,0,0)
            time.sleep(0.1)
```

Code 42 Improved Turning Program

After adopting this solution, the in-position turning orders are always executed perfectly in Patio 1.

Crossing Bridge

In field tests, we find that there is no problem with climbing up the bridge or running down the slope, since the wheels of our car have good track adhesion and the 12-Volt motors are powerful enough. Therefore, we just focus on how to move straight when crossing the bridge.

As there is no line for Open MV to carry out tracing, the car can only depend on the gyroscope to determine direction, i.e., Mode 4 and Mode 5 are to choose from.

Although Mode 4 stands for being sensitive to deviation, it also means the car would response in a dramatic way, adding risks for the slopes. As a result, we choose Mode 5. In Mode 5, the car adjusts direction in a milder way than in Mode 4. Despite possible additive deviation, Mode 5 shows great performance in a short distance, therefore is suitable to be applied in bridge crossing.

Above all, field tests have witnessed the correctness of our choice.

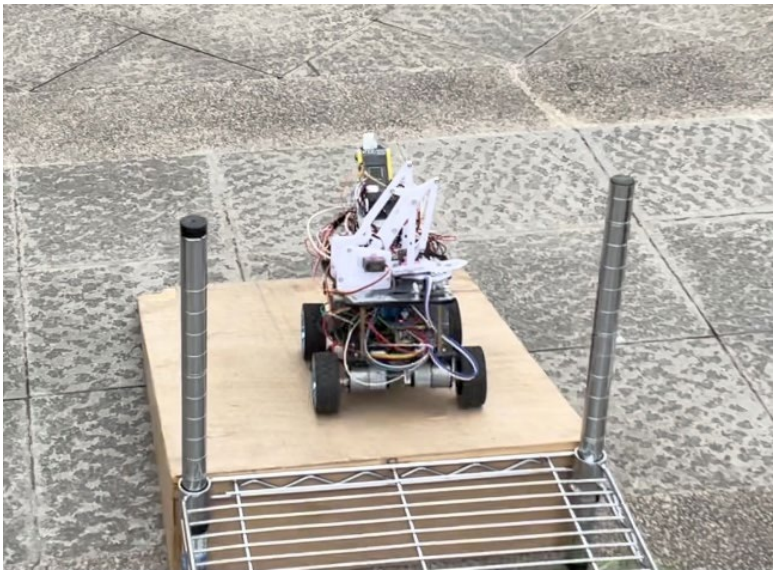


Figure 68 Crossing Bridge

4.1.2.2 Patio2

A: Self-adaptive direction control mode

In task 2, dynamic self-adaptive direction control mode is widely implemented in both the smooth tile surface and the rough cobblestone surface (Figure 69). It successfully makes the car go straight in two surfaces with quite different friction coefficients. We overcome the difficulties that the tire is easy to slip on tile surface and the wheel is likely to stuck in cobblestone.

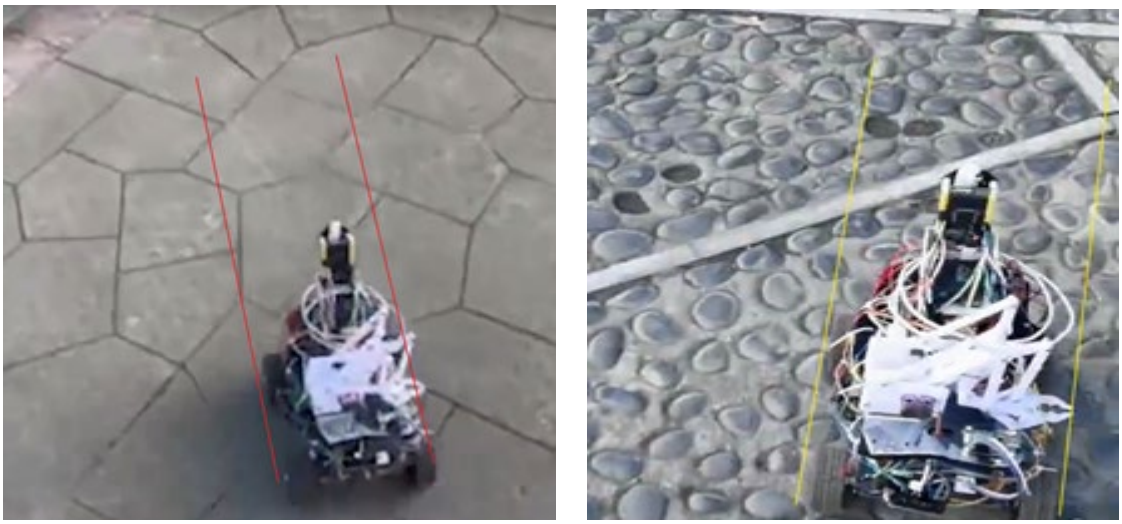


Figure 69 Running along a straight line in tile surface and cobblestone surface

Compared with fixed PWM value adjustment per second, the dynamic self-adaptive direction control mode use both exponential functions related to deviation angle

$$\text{Adjustment}=|\text{deviation angle}|\times e^{|\text{deviation angle}|/2.2}$$

and negative feedback to get better performance in going straight. The adjustment is relevant to the magnitude of disturbance, this algorithm first quantizes the influence of the uneven surface of the ground and the stall of the wheel into the deviation angle mapping to yaw axis (rotate on car's up vector)

Based on self-adaptive, the angle adjustment is divided into three stages due to the on-spot test experience, stable zone, tremor zone, and fluctuation zone. We set the different magnifications for adjustments in three zones. Moreover, to avoid the circumstance that the wheel stuck on the pebble, we amplify the PWM output value to the motor in pebble road. On the contrary, the car runs slowly in the tile surface. The car can automatically distinguish the different roads by judging the angle fluctuation in pitch axis (rotate on the cross product of the other camera's up and direction vectors).

B: Turing:

Turning on the spot plays an indispensable role in performing the tasks 2, which has been repeatedly implemented and has great influence on the positioning of the car. It is essential for car to turn on the spot at a target angle precisely. To be specific, turning at an angle precisely is the foundation for the car to track the desired route, as the car may fail to approach the desired position without a correct initial direction. We have designed algorithm to cope with the turning problem.

Our algorithm enables the car to turn at a precise angle with the tolerance of 1 degree and can revise the turning angle of the car automatically with the use of gyroscope.

We defined delta as the difference angle between the target angle received from openmv and the current angle provided by the gyroscope, where both the target angle and current angle are ranging from -180 degree to 180 degree, which determines the turning direction and the turning angle. To be specific, if delta is greater than 0, the car rotates clockwise and rotates anticlockwise when delta less than 0. However, we faced with a challenge due to the range of delta, for instance, the problem occurs when target angle=179°, while current angle=-179°, the actual result we want is to turn clockwise for 2°, the result will be turning anticlockwise for 358°. In order to solve this problem, we adjust the range of delta by dividing the turning in to several conditions, for instance, when:

$$90° \leq \text{target ange} \leq 180° \quad \text{and} \quad -180° \leq \text{current angle} \leq -90°$$
$$\text{delta} = \text{current angle} - \text{target angle} + 360°$$

With the algorithm, delta is adjusted into the range of -180° to 180°. And the car will repeatedly revise the angle until the delta angle is less than 1 degree. Moreover, the car will turn at a higher speed when delta angle is greater than 20°.

```
def turn(delta):
    if delta>0:                         #clockwise
```

```
        if delta>20:
            Car.run_(0,50,50,0,50,0,0,50)
            time.sleep(0.4)
        if delta<20:
            Car.run_(0,30,30,0,30,0,0,30)
            time.sleep(0.2)
            Car.run_(0,0,0,0,0,0,0,0)
            time.sleep(0.1)
    if delta<0:                      #anticlockwise
        if delta<-20:
            Car.run_(50,0,0,50,0,50,50,0)
            time.sleep(0.4)
        if delta>-20:
            Car.run_(30,0,0,30,0,30,30,0)
            time.sleep(0.1)
            Car.run_(0,0,0,0,0,0,0,0)
            time.sleep(0.1)
            if int(sum[1:2])==1: #counterclockwise
                turn_angle=int(sum[2:5])    # >0: anticlockwise,
<0: clockwise
            if int(sum[1:2])==2: #clockwise
                turn_angle=-int(sum[2:5])
            # global z_angle
            # UART_Gyroscope.irq(trigger = UART.IRQ_RXIDLE, handler
= UART_Gyroscope_ISR)
            # global original_angle
            count=0
            time.sleep(0.5)
            turn_complete=0
            target_angle=original_angle+turn_angle
            if target_angle<=-180:
                 target_angle=target_angle+360
            if target_angle>180:
                 target_angle=target_angle-360
            while(turn_complete==0):
              if z_angle<=-90 and z_angle>=-180 and
target_angle<=180 and target_angle>=90:
                 delta=z_angle-target_angle+360
              elif z_angle>=90 and z_angle<=180 and target_angle>=-
180 and target_angle<=-90:
                 delta=z_angle-target_angle-360
              else:
```

```
                delta=z_angle-target_angle
                if z_angle-target_angle>=180:
                    delta=-(360-delta)
                if z_angle-target_angle<=-180:                    #
取小于 180 的夹角
                    delta=-(360+delta)
            # print(delta,z_angle)
            if abs(delta)>=1:
                print(delta,z_angle)
                turn(delta)
            if abs(delta)<1:
                # print(delta)
                turn_complete=1
                # Car.run_(0,0,0,0,0,0,0,0)
                sum="0"
                 # UART_Gyroscope.deinit
```

C: Robotic Arm:

The robotic arm is necessary in task 2 to comply the task of releasing fish food. A two-degree-of-freedom robotic has been installed, which successfully performed the task in the last examination. The arm is designed to perform an action of throwing which can cast the fish food to the target position and then return to the original state in a relatively more sluggish speed. Besides, the size of the robotic arm is of great importance, the deepest diving down distance is up to 7cm, the largest arm span is up to 23cm, and the highest lifting distance is up to 15cm.

Figure 70 Size of robotic arm

The two joints of the arm are controlled by two servos (MG90S) respectively, where the rotation angle of servos can be adjusted by PWM value.

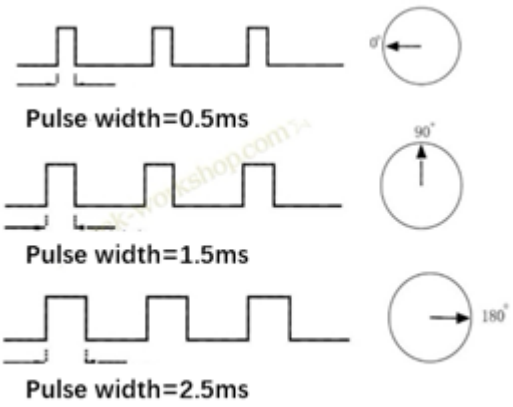


Figure 71 Working principle of servos

It is noteworthy that the PWM range is 0.25 to 12.5 under the condition that the period of the servos is 20ms. Based on measurement and trials, we set the PWM values of the two servos (which are in responsible for the control of right and left arms) to about 12. Moreover, the robotic arm is controlled by the instructions from OpenMV.

```
# main.py -- put your code here!
from pyb import UART, Pin,Timer
import time
from time import sleep

print("hello world")

A0 = Pin('PA0') #control the left steering gear engine
tim = Timer(2, freq=50)
ch = tim.channel(1, Timer.PWM, pin=A0)

B3 = Pin('PB3')# control the right steering gear engine
tim1 = Timer(2, freq=50)
ch1 = tim.channel(2, Timer.PWM, pin=B3)

ch.pulse_width_percent(6.3)
ch1.pulse_width_percent(8.5)

def releasing():
    i=6.3 #左边
    m=8.5 #右边
    flag=1
    while flag:
        i+=0.1
```

```
        m+=0.1
        ch.pulse_width_percent(i)
        ch1.pulse_width_percent(m)
        if m>=12:
            m=12
        if i>=11.8 and m>=12:
            #print(i)
            #print(m)
            i=11.8
            m=12
            ch.pulse_width_percent(i)
            ch1.pulse_width_percent(m)
            time.sleep(2)
            while i>=6.3 or m>=8.5:
                if i>=6.3:
                    i=i-0.1
                if m>=8.5:
                    m=m-0.1
                print(i)
                print(m)
                time.sleep(0.3)
                ch.pulse_width_percent(i)
                ch1.pulse_width_percent(m)
                if i<6.3 and m<8.5:
                    flag=0
```

4.2 Decision-Making System (Image Group)

4.2.1 System Integration

4.2.1.1 Patio 1

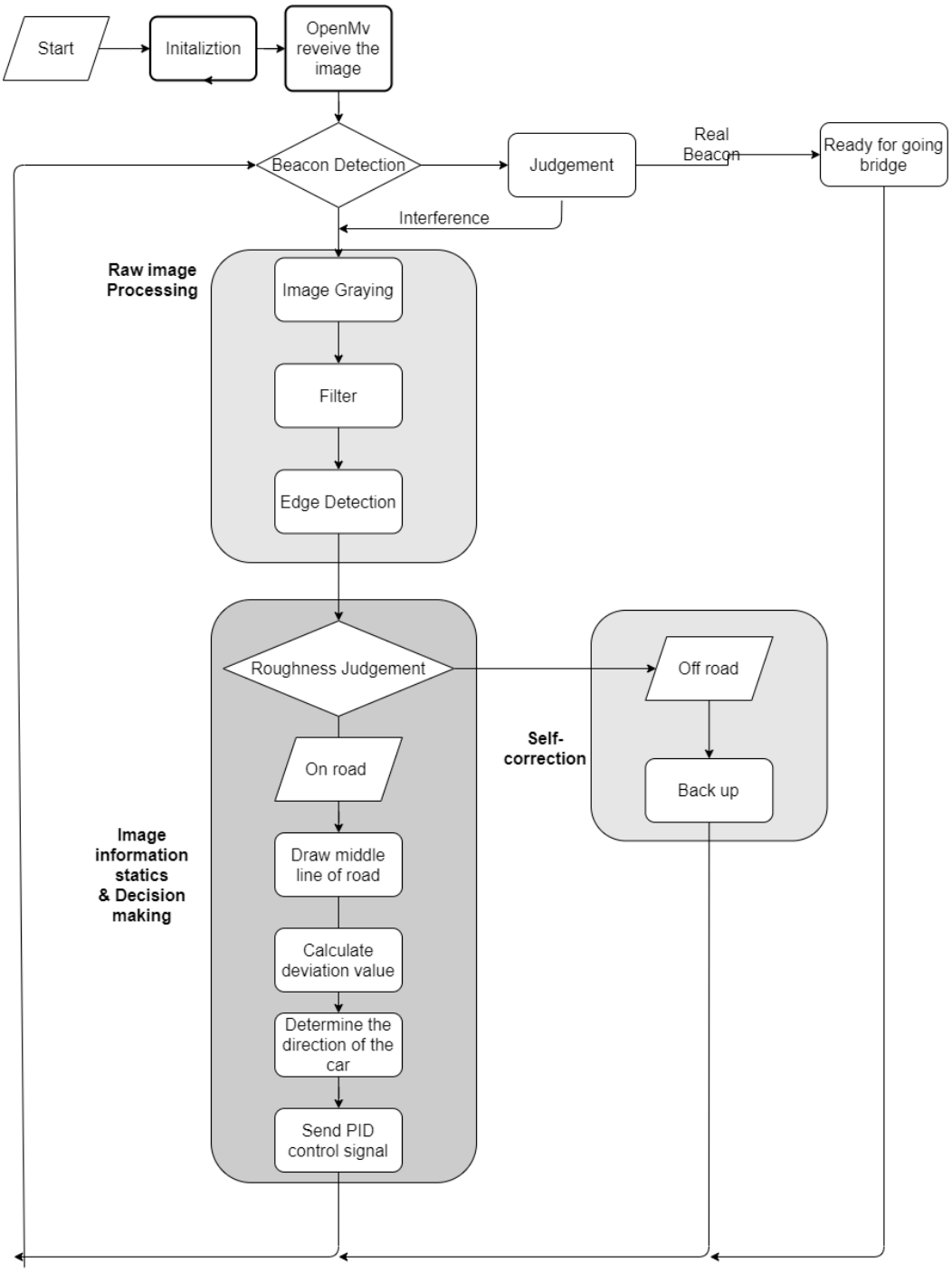


Figure 72 Patio system integration flow chart.

Figure 72 presents the system integration flow chart for Patio 1. The detailed information for specific algorithm realization and decision making is illustrated in section 4.2.2, and 3.6-3.12.

Starting from the top left of Figure 72, the OpenMV camera is initialized and begin to capture images. Then, the beacon detection process is entered. Whenever the beacon is detected, the decision-making process for beacon, i.e., crossing the bridge, is entered, and images capturing cycle is jumped out.

For processing images from OpenMV, the raw images are converted into grayscale, filtered and processed by edge detection. The edge detection provides details of path roughness in the image. Then, the decisions are made following several steps, including drawing middle line of road, calculating deviation value. Consequently, the direction for the robotic car is determined, and the corresponding PWM signals are derived by PID algorithm. The motion of the car can be controlled by the transmission of PWM signals.

It is worth noting that there is a self-correction mechanism that is capable of interrupting the direction judgement. If OpenMV sees an off-road image, the backup mechanism begins. This allows us to self-correct our erroneous direction commands promptly.

4.2.1.2 Patio 2

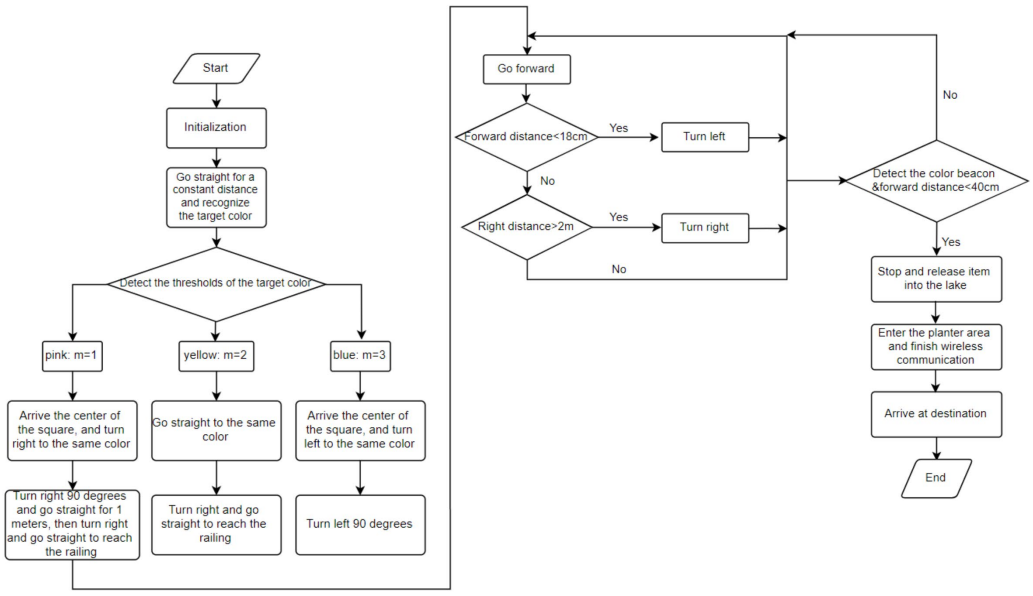


Figure 73 System Integration Flow Chart of Patio 2

The picture is the flow chart of patio 2 main program. The detailed process and improvement are introduced in3.5, 3.8, 3.9, 3,11 and 4.2.3

Patio 2 starts from color recognition and uses the threshold in the collected threshold library to determine the image. When the car recognizes the color of the block is pink, yellow or blue, the m is assigned to 1, 2 or 3 respectively. According to the value of m, the car could determine the direction of the car and drive to the position where the color block is.

After the color recognition, the car judges the current position according to the value of m and drives to the railings along the lake and uses the ultrasonic to measure the distance between the car and the right railings and the front railings to determine whether the car should turn.

When the car finds the beacon, it will stop running and use the mechanical arm to complete feeding action. After feeding, the car will travel to the back of plant area to complete the wireless communication task by HC-12.

4.2.2 Result and Discussion of Patio 1

4.2.2.1 Beacon chosen

Initially, we decided to use the Aptiltag beacon as it could return many parameters like distance and rotation angle in 3D dimension. The more information we get, the more accurate control signals we can send to the car. However, due to the limited RAM of OpenMV, the algorithm for recognizing Apriltag could only use QQVGA resolution. This resolution will greatly influence our effect of edge detection and cause the car to off the road especially in the corner. Balancing the road tracing and the beacon chosen, we decided to give up Apriltag and use ultrasonic sensor to measure distance only. In the testing, we only need to ensure our car to be in the middle line of the road before crossing the bridge. The effect is good actually.

4.2.2.2 Crossing the Bridge and Gate

After tracing, the bridge and gate should also be crossed, and instructions are required within this path. As in Figure 74, three general commands are labelled in the route after tracing.

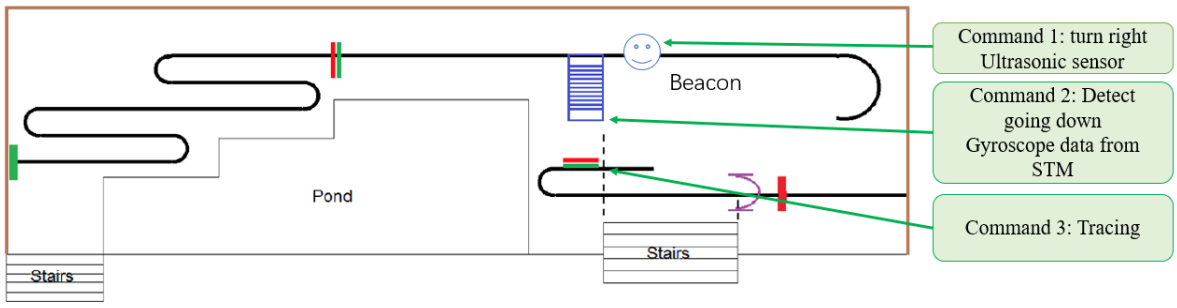


Figure 74 Commands after tracing.

Command 1 is for turning right for the bridge. A beacon, or object, is used to tell our robotic car where to turn right. The detection is assisted with one front ultrasonic sensor on our car. Once the ultrasonic sensor detects a distance within 35 cm, the OpenMV will transmit a “stop” instruction to the car. The detailed code is in Code 43.

```
if (25<forward_distance<35 and isPassBridge==0):

    # repeat
    stm_info="0"  # stop the car
    UART_stm.write(stm_info)
    print(stm_info)
    time.sleep(1)

forward_distance2=wave_distance_process(wave_echo_pin_2,wave_trig_pin_2)
    time.sleep(0.5)

forward_distance3=wave_distance_process(wave_echo_pin_2,wave_trig_pin_2)
    if (25<forward_distance2<35 and 25<forward_distance3<35):
        isBeacon=1

if isBeacon==1:
    stm_info="0"  # stop the car
    UART_stm.write(stm_info)
    print(stm_info)
    time.sleep(0.5)
```

Code 43 Command for turning right to cross the bridge.

The “if” condition judgement in the first line is to detect when the distance is less than 35 cm. Highlight that here we adopt distance detection repetition. This is because there exist reflectors on the road, which will falsely make the distance measurement less than 35 cm. Three distance detections are executed in total. After the first detection, the robotic car will stop and wait for another two detections. Only after all three distances are less than 35 cm, the car will turn right and cross the bridge, while crossing the bridge is achieved by the gyroscope.

In order to know when to begin tracing again, we should identify the time that the car goes down the bridge. This is achieved by collecting data from STM gyroscope. The gyroscope is aware of the angle changes, so when the vertical angle is changing, a message is sent back to OpenMV. Consequently, OpenMV knows the bridge is passed and executes tracing program for command 3. Note that “isbeacon” variable in the code is designed for counting the times of crossing the bridge, which means that the car crosses only once. After “isbeacon” is set to “1”, the crossing bridge program will not be executed again.

Command 3 is tracing along the road until the car passes the gate. Tracing along the road can guarantee the car not to deviate from the road and cross the gate successfully.

4.2.2.3 Algorithm Robustness

A highlight for our algorithm in Patio 1 is the robustness. Four aspects of robustness are considered, and this ensures our performance against environmental factors and various errors produced.

The backup mechanism is our first consideration of robustness. Since the instruction time may not follow up the speed of motions and OpenMV may not be stable to give correct instructions every time, the robotic car may deviate from the path. Once off the path, the normal OpenMV system cannot automatically correct the decision since it does not know which direction the path locates. Under this condition, we set a backup mechanism to let the car move backwards when OpenMV cannot detect the path. After backing up, the camera may see a more stable picture and make rational decisions.

The second mechanism is proposed to prevent ultrasonic erroneous detections. As mentioned in 4.2.2.2, three times of distance detections are collected continuously from OpenMV for robustness. The erroneous near distances can be filtered out and the decisions will not be interfered.

The third aspect aims at tackling the unstable control for the robotic motion. Since the road detected varies greatly, then the control commands will vary discontinuously, which makes it difficult for the robotic car to react. Thus, Proportional Integral Differential (PID) algorithm is applied to smoothly adjust the command for the car. PID algorithm gives continuous instructions that are convenient and fast to adjust. Additionally, PWM output, instead of angle output, is given directly from OpenMV, which is more direct. If angles are passed, the OpenMV first calculates the angle, and the car in turn adjusts its PWM for the wheels to achieve that angle. The PWM output transmission rules out the tedious steps and quantization errors encountered into multiple conversions.

The fourth aspect accounted of is the robustness for the decision making based on processed images. Due to the environmental factors and condition variations, the images vary drastically. There are substantial interference and noise, which cannot be totally eliminated and contain abundant features. In terms of robustness, region of interest (ROI) is adopted. The pixels in ROI are processed, without considering the pixels out of ROI. In this way, the OpenMV only make decisions based on the reliable pixels within ROI. The instructions are in turn more robust and reliable.

Another highlight for adjusting parameters and debugging for our system is the Liquid Crystal Display (LCD) on our OpenMV. The robotic car must be run off-line, and we cannot hold a laptop connected to see what happens in OpenMV every time. Therefore, a LCD is set on

OpenMV to show the processed images as a powerful tool, providing information for debugging.

4.2.3 Result and Discussion of Patio 2

4.2.3.1 Color Recognition

The color threshold will vary depending on the weather and time, so multiple sets of threshold data are used to select the final threshold according to the actual situation. Secondly, in the actual test, because the camera's field of view angle is limited, the car should have an initial angle of rotation based on the recognized target color, ensuring that the second color block to be reached determines the car's forward path within the vehicle's field of view. The last problem is that when a small car on the track completes a turn on a tile, the slip phenomenon is more serious, and the instruction cannot be completed accurately. Thus the car will adjust the direction in the center of the square.

To avoid the above problems, the last color-recognized path of the car is shown in the following image:

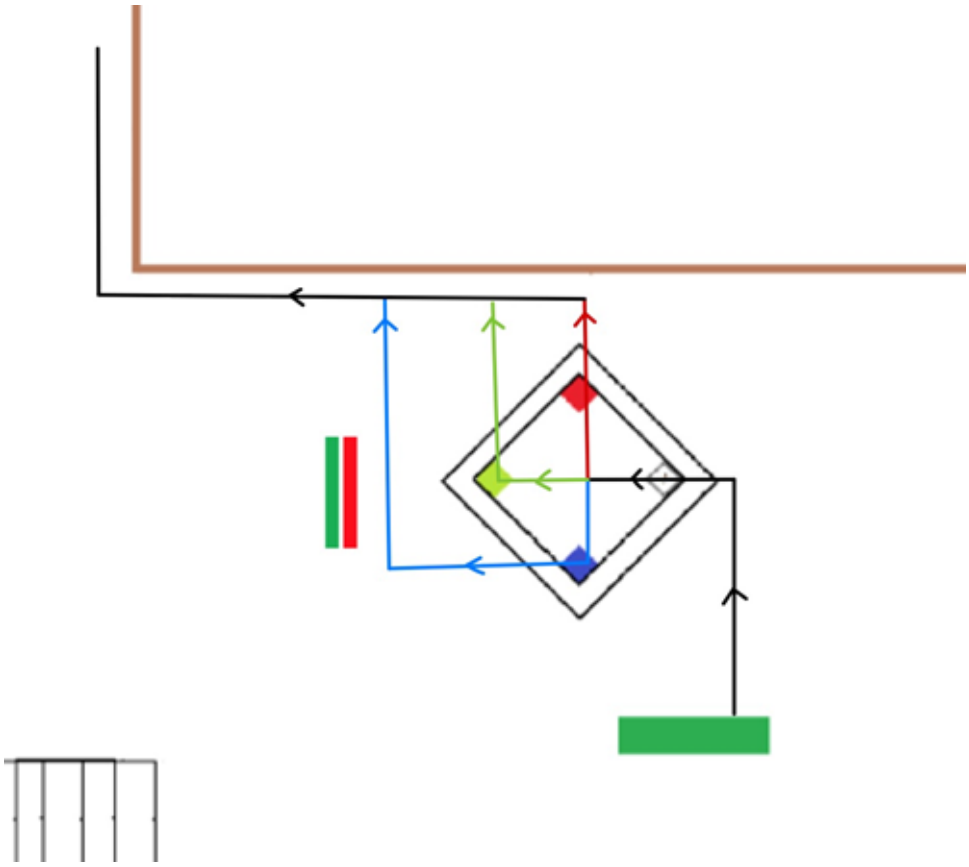


Figure 75 Route of color recognition

4.2.3.2 Programme of Go Straight

In the past programme, the car called an ultrasonic ranging every 30cm forward to measure the distance between the car and the railings in front of it. While in the actual test, when the car sent forward instructions, the gyroscope will be reset, so the car cannot run straight.

```
forward_distance=wave_distance_process(wave_echo_pin_2,wave_trig_pin_2)
print(forward_distance)
while(forward_distance>25):
        forward_distance=wave_distance_process(wave_echo_pin_2,wave_trig_
        pin_2)
        print(forward_distance)
        go_forward(100)
        turn_left()
```

Code 44 Initial design for arriving railings

In the final design, the straight-line instruction is called only once, so the gyroscope is set only once in the first forward instruction, so the car could keep moving in the direction in which the instruction is sent out.

```
go_straight()
while True:
        forward_distance=wave_distance_process(wave_echo_pin_2,wave_trig_
        pin_2)
        print(forward_distance)
        if forward_distance<40:
            stop()
            time.sleep(0.5)
            forward_distance=wave_distance_process(wave_echo_pin_2,wave_t
            rig_pin_2)
            print(forward_distance)
            if forward_distance<50:
                time.sleep(1)
                turn_left_93()
                break
            else:
                go_straight()
```

Code 45 Final design for arriving railings

4.2.3.3 Ultrasonic Wave Detecting Railing

The complete route along the railing is shown below:

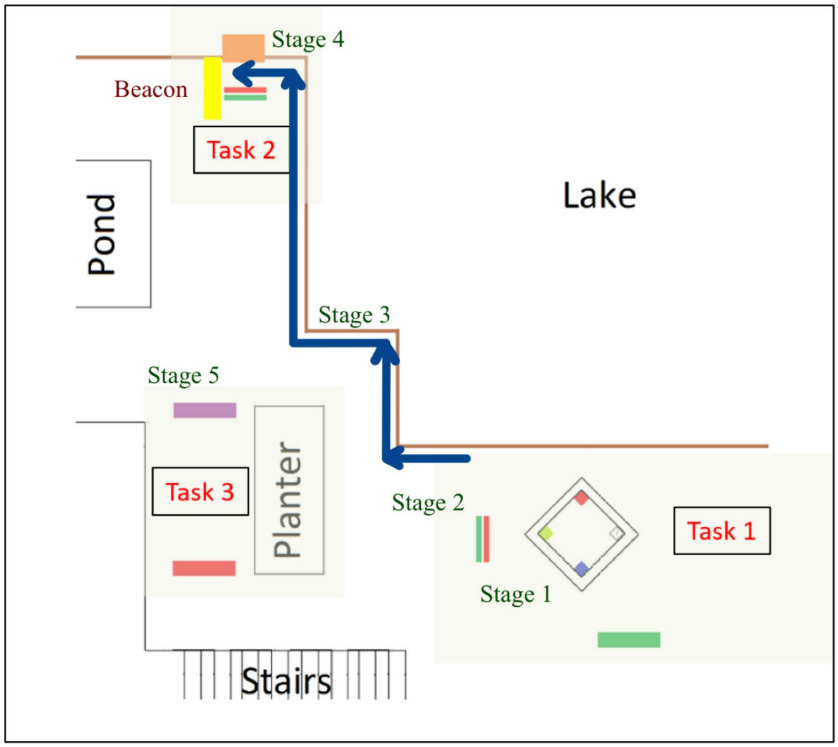


Figure 76 Route Map from Color Recognition to Finding Point

After finishing the color recognition, the ultrasonic sensors are regarded as the main tools, and three ultrasonic sensors have been installed in the front, left and right sides of the car respectively. In addition, to ensure the accuracy of detecting values avoiding convey wrong instructions, it would calculate average distance value of every four detecting data.

Initially, the code is designed for the following purpose: in details, if the right-side ultrasonic sensor detect a distance less than 10 cm from railing, the OpenMV will call the function adjust_left, which is a 10-degree fine adjustment in the direction of the car away from the railing. Similarly, if the right-side ultrasonic sensor detects a distance larger than 20 cm from railing, function adjust_right will be called to execute a 10-degree fine adjustment in the direction of the railing. Moreover, if the forward sensor detects the distance less than 10cm, it will turn left.

```
def turn_left():
    stop()
    direction='21090'
    print('turn left',direction)
    stm_info=str(direction)
    UART_stm.write(stm_info)
    #time.sleep(2)
```

```
    #go_forward()

def turn_right():
    direction='22090'
    print('turn_right',direction)
    stm_info=str(direction)
    UART_stm.write(stm_info)
    time.sleep(5)
    go_forward()

def adjust_left():
    direction='31010'
    print('adjust_left',direction)
    stm_info=str(direction)
    UART_stm.write(stm_info)
    time.sleep(1)
    go_forward()
    time.sleep(0.2)
    go_forward()
    time.sleep(0.2)
    go_forward()
    time.sleep(0.2)

def adjust_right():
    direction='32010'
    print('adjust_right',direction)
    stm_info=str(direction)
    UART_stm.write(stm_info)
    time.sleep(1)
    go_forward()
```

Code 46 Initial Driving Instructions for Trolley

When the car completes its last turn, in order to stop accurately at the feeding point, OpenMV starts to detect the position of the beacon through color recognition and at the same time, the forward ultrasonic sensor would detect the distance from the beacon, when it finds the beacon with a distance of 20cm, it will stop.

However, when the actual vehicle test in carried out on the field, the completion effect is not very satisfactory. In fact, When the adjustment angle of the trolley is less than 10 degrees, the trolley does not directly steer but only fine-tunes the angle. Therefore, in the field test, the effect of this fine-tuning is not obvious. When the car exceeds the set straight distance range, the corresponding fine-tuning starts, but because the fine-tuning effect is not obvious, the driving

direction of the car cannot be corrected in time, so it will continue to deviate, for instance: the right side of the car is detected by ultrasonic senor on the right side of the car. If the distance from the side to the railing is too large, it will be corrected in the direction of the railing, but in fact, he will continue to deviate from the railing, resulting in the inability to complete the task normally.

In response to this situation, the code has been modified accordingly, modifying the adjustment method of the trolley from fine-tuning during the traveling process to in-situ adjustment. Specifically, when the ultrasonic detection on the right side of the trolley detects that the distance between the trolley and the railing exceeds the set distance range, the trolley will stop advancing and turn to the corresponding angle on the spot, and then continue going forward. For instance, if it is too far away from the railing, it will turn on the spot in the direction closer to the railing for 11 degrees. As a fact, the turning angle cannot be less than 10 degrees, which means the trolley could only be fine-tuned under the traveling conditions, and it cannot be achieved when the trolley stops. However, the field test situation is not ideal. Once the car deviates from the correct direction and starts to adjust, its rout will become Z-shaped, and maintain until finish the whole task.

In addition, when the trolley shifts, the distance from the railing detected by the ultrasonic on the right is no longer the actual distance perpendicular to the railing, as shown in the figure below.

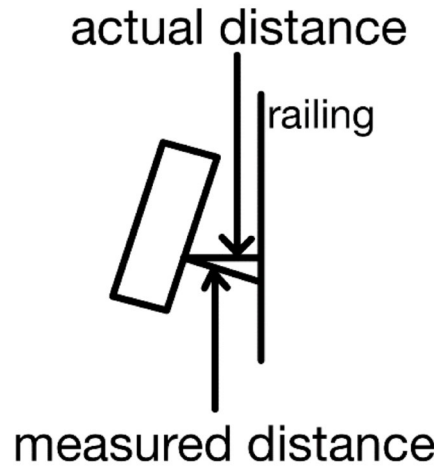


Figure 77 Difference between Actual Distance and Measured Distance

Apparently, the measured distance is larger than the actual distance, and as the angle of deviation of the trolley is greater, the difference between the ultrasonic measured distance and the actual distance will be greater. The result of this phenomenon is that the car will constantly adjust in the opposite direction, so that it will aggravate the skew of its driving route.

In general, these two adjustment methods are not ideal for correct the direction of the car. Finally, it is found that in the field test, after the car drives to the railing, sending a straight forward command to the car directly instead of the mentioned adjust functions performs the

best effect.

```
def stop():
    direction='0'
    print(direction)
    stm_info=str(direction)
    UART_stm.write(stm_info)
    time.sleep(0.5)

def go_forward(distance):
    velocity=0.45
    run_time=int(distance/velocity)
    direction='5'
    print(direction)
    stm_info=str(direction)
    UART_stm.write(stm_info)
    time.sleep(run_time)
    stop()

def go_straight():
    direction='5'
    print(direction)
    stm_info=str(direction)
    UART_stm.write(stm_info)

def turn_left():
    #time.sleep(1.5)
    direction='21090'
    print('turn left',direction)
    stm_info=str(direction)
    UART_stm.write(stm_info)
    #global isTurn
    #while True:
        #if isTurn == 1:
            #print(isTurn)
            #isTurn=0
            #break
    time.sleep(15)

def turn_left_92():
    #time.sleep(1.5)
    direction='21092'
    print('turn left',direction)
```

```
    stm_info=str(direction)
    UART_stm.write(stm_info)
    #global isTurn
    #while True:
        #if isTurn == 1:
            #isTurn=0
            #break
    time.sleep(15)

def turn_right():
    #time.sleep(1.5)
    direction='22090'
    print('turn right',direction)
    stm_info=str(direction)
    UART_stm.write(stm_info)
    #global isTurn
    #while True:
        #if isTurn == 1:
            #isTurn=0
            #break
    time.sleep(15)
```

Code 47 Final Driving Instructions for Trolley

Data processing of ultrasonic measurement:

During the traveling of the trolley, the ultrasonic ranging would detecting series wrong data, which will cause the trolley to execute improper instructions. One of the ways to improve the accuracy of the measured distance is to average the measured data:

```
right_total=0
for i in range(3):
right_distance_i=wave_distance_process(wave_echo_pin_3,wave_trig_pin_3)
  right_total=right_total+right_distance_i
right_distance=right_total/3
print("right:",right_distance)
forward_total=0
for i in range(3):
forward_distance_i=wave_distance_process(wave_echo_pin_2,wave_trig_pin_2
)
  forward_total=forward_total+forward_distance_i
forward_distance=right_total/3
print("forward:",forward_distance)
```

Code 48 Processing of Data Detected by Ultrasonic Sensors

In addition, when the car bumps, the distance detected by the ultrasonic wave may be much greater than the actual distance, causing the car to execute wrong instructions, turn early, and deviate from the route. In order to prevent this phenomenon, when the measured data meets the turning conditions, the car will stop moving, and the distance will be measured again and the conditions will be judged in a static state. When the measured data meets the turning conditions, the car will perform a turn Order, otherwise it will continue straight ahead.

Moreover, the car judges whether to execute a right turn command mainly based on the data measured by the ultrasound on the right side, while judging whether to turn left is mainly based on the data measured by the frontal ultrasound. Therefore, in order to shield the influence of the frontal ultrasonic detection on the selection of the trolley command, when the trolley travels to specific sections, the forward ultrasonic detection data will be automatically ignored. In the same way, when the car judges whether to execute the left turn command, it will also shield the ultrasonic detection data on the right side.

```
#stage 3 along the handrail
turns=0
go_straight()
while(turns<4):
    if turns == 0 or turns == 2:
        while True:
            right_total=0
            for i in range(3):
              right_distance_i=wave_distance_process(wave_echo_pin_3,wave
_trig_pin_3)
               right_total=right_total+right_distance_i
            right_distance=right_total/3
            print("right:",right_distance)
            if right_distance>200:
                time.sleep(1.2)
                stop()
                time.sleep(1)
              right_distance=wave_distance_process(wave_echo_pin_3,wave_t
rig_pin_3)
                if right_distance>250:
                    turn_right()
                    go_forward(2)
                    turns=turns+1
                    print("Turns:",turns)
                    go_straight()
                    break
```

```
                else:
                    go_straight()


        if turns == 1 or turns == 3:
            while True:
                forward_total=0
                for i in range(3):
                  forward_distance_i=wave_distance_process(wave_echo_pin_2
,wave_trig_pin_2)
                    forward_total=forward_total+forward_distance_i
                forward_distance=forward_total/3
                print("forward:",forward_distance)


                if forward_distance<18:
                    stop()
                    time.sleep(1)
                  forward_distance=wave_distance_process(wave_echo_pin_2,w
ave_trig_pin_2)
                    if forward_distance<30:
                        turn_left()
                        turns=turns+1
                        go_straight()
                        break
                    else:
                        go_straight()
```

Code 49 Judgment and Processing Method of Executing Instructions

5 Conclusion

In this TDPS course, our group successfully designed, assembled and programmed an intelligent robot cart to complete tasks of two patios. The experimental results proved the feasibility of our car design.

In the execution system of the car group, it mainly divides into two patios with different functions. In patio 1, STM32 decode the PWM value calculated by the distance and lightness parameters from Open MV and it transmit the value to motors. With the help of power driver module, car accomplish the right tracing. Moreover, gyroscope plays an important role not only in rotation and but also in crossing bridge with straight direction. In patio 2, dynamic self-adaption mode based on the data from gyroscope overcome the difficulty in adjusting rotation speed on tile surface and cobblestone surface with different friction coefficients, which assist car go straight. The robotic arm with exquisite mechanical structure finished the feeding task. In the driver module, optocoupler is preferred to add between control circuit and L298N, avoiding the overcurrent and overvoltage to breakdown the STM32 board.

As for decision-making system of the image group, in patio1, we set up a flow chart consisting of four main parts: Raw image processing, image information utilization, beacon recognition and self-correction. For raw image processing, we adopted simple edge detection method as recognizing the roughness difference between the road and the off-road path. For image information utilization, we calculate the horizontal deviation distance between the road middle line and car moving direction and use this as PID input. Also, ultrasonic sensor is used for beacon recognition to prepare for crossing the bridge. Self-correction mechanism as well as three other mechanism are added to improve our algorithm robustness. The result of our tracing is quite good. Car could successfully finish all tasks under different weathers. Considering Patio 2, when the car completes the task of color recognition, the thresholds of the color block need to be adjusted according to weather conditions to ensure that the target color can be successfully recognized and the corresponding color block with the same color can be found. After the recognition is completed, use the ultrasonic sensor in front of the car to drive to the railing to start the next task. In the next stage, ultrasonic sensors on the right side and forward of the trolley are regarded as the main tools to finish the rest of tasks in Patio 2. In order to ensure the accuracy of the transmitted instructions, the ultrasonic detection data needs to be filtered and averaged to prevent the car from executing wrong instructions.

References

[1] "What Is A Microcontroller?". Elektormagazine.Com, 2021, https://www.allaboutcircuits.com/technical-articles/what-is-a-microcontroller-introduction-component-characteristics-component/. Accessed 25 June 2021.
[2] "STM32F411 - Stmicroelectronics". Stmicroelectronics, 2021, https://www.st.com/en/microcontrollers-microprocessors/stm32f411.html.
[3] "Micropython - Python For Microcontrollers". Micropython.Org, 2021, https://micropython.org/.
[4] Shilpashree, K. S., H. Lokesha, and Hadimani Shivkumar. "Implementation of image processing on Raspberry Pi." International journal of advanced research in computer and communication engineering 4.5 (2015): 199-202.
[5] Nelson, Anthony Edward. Implementation of image processing algorithms on FPGA hardware. MS thesis. Vanderbilt University, 2000.
[6] Small - Affordable - Expandable. OpenMV. (2021). Retrieved 25 June 2021, from https://openmv.io/.
[7] He, Anson. "Build Your Own Robot Car: Choosing The Right Motor-Driver - Latest Open Tech From Seeed Studio". Latest Open Tech From Seeed Studio, 2021, https://www.seeedstudio.com/blog/2019/07/29/build-your-own-robotic-car-choosing-the-right-motor-driver/.
[8] "Interface L298N DC Motor Driver Module With Arduino". Lastminuteengineers.Com, 2021, https://lastminuteengineers.com/l298n-dc-stepper-driver-arduino-tutorial/.
[9] "Ultrasonic Ranging Module HC - SR04". Cdn.Sparkfun.Com, 2021, https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf.
[10] "Difference Between Locked Rotor Current And Starting Current | Electrical Concepts". Electrical Concepts, 2021, https://electricalbaba.com/locked-rotor-starting-current/. Accessed 26 June 2021.
[11] Datasheets.Maximintegrated.Com, 2021, https://datasheets.maximintegrated.com/en/ds/DS3231.pdf.
[12] Sinha, Utkarsh. "The Canny Edge Detector: Introduction The Edge Detector - AI Shack". Aishack.In, 2021, https://aishack.in/tutorials/canny-edge-detector/.
[13] Campbell, Scott. "Basics Of The I2C Communication Protocol". Circuit Basics, 2021, https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/.
[14] “Car tracking control based on image sensor” .wenku.baidu.com, 2021, https://wenku.baidu.com/view/30729334182e453610661ed9ad51f01dc281573a.html
[15] “Introduction to openmv camera and advanced application of color block recognition and two dimensional code”blog.csdn.net,2020, https://blog.csdn.net/smart_99/article/details/104505812
[16] Dawoud, D., 2021. Serial Communication Protocols and Standards RS232/485, UART/USART, SPI, USB, INSTEON, Wi-Fi and WiMAX | River Publishers eBooks | IEEE Xplore. [online] Ieeexplore.ieee.org. Available at: <https://ieeexplore.ieee.org/servlet/opac?bknumber=9218891>

[18] “Ultrasonic Collision Detection”. Synthiam.com, 2021, https://synthiam.com/Support/Skills/Ultrasonic-Distance/Ultrasonic-Collision-Detection?id=16102
[19] “Ultrasonic Detection-Basics& Application”. Elprocus.com, 2021, https://www.elprocus.com/ultrasonic-detection-basics-application/

Appendix I Timetable and Gantt Chart

Here we demonstrate the time arrangement of our project until 2021.6.16 and the Gantt chart.

Mission	Start Time	Days	Group
Team Foundation	2021/3/3	1	All
First Contact (Offline,2h)	2021/3/5	1	All
OpenMV PCB Design	2021/3/6	1	Image
Image Group Pre-Study	2021/3/6	8	Image
Car Group Pre-study	2021/3/6	8	Car
OpenMV Circuit Board Production & BOM Purchase	2021/3/7	5	Image
Seminar 1 (Offline,2h) Week2	2021/3/14	1	All
Car Purchase and Assembling	2021/3/15	4	Car
L298N PCB Design	2021/3/16	2	Car
OpenMV Board Soldering	2021/3/19	1	Image
L298N Circuit Board Production & BOM Purchase	2021/3/19	4	Car
OpenMV Tracing Approach	2021/3/20	60	Image
Seminar 2 (Online,30min) Week3	2021/3/21	1	All
L298N Board Soldering	2021/3/25	1	Car
Car Wiring	2021/3/27	1	Car
Seminar 3 (Offline, 2h) Week4	2021/3/28	1	All
OpenMV Color Recognition Approach	2021/3/30	25	Image
Motor Control using L298	2021/4/2	8	Car
Seminar 4 (Online, 30min) Week5	2021/4/4	1	All
Initial Report and Initial Lab Book	2021/4/5	7	All
Seminar 5 (Offline, 2h) Week6	2021/4/11	1	All
Gyroscope communication	2021/4/12	27	Car
Ultrasonic Sensor Usage & Comunnication	2021/4/15	3	Image
Initial Presentation	2021/4/19	1	All
Manipulation Strategy Integration	2021/4/22	7	Image
Seminar 6 (Online, 30min) Week8	2021/4/25	1	All
HC-12 Communication	2021/5/2	2	Image
PID control	2021/5/2	10	Car
Inter-board Communication using UART	2021/5/8	2	Image
Orientation Control in Place	2021/5/9	14	Car
Seminar 7 (Online, 30min) Week10	2021/5/9	1	All
Car Dynamic Direction Adjustment with OpenMV	2021/5/12	9	Car

OpenMV interface with stm32 for car motion	2021/5/16	4	Image
Seminar 8 (Offline, 1h) Week11	2021/5/16	1	All
Robotic Arm Design and Test	2021/5/20	5	Image
Low Latency Control (Image-end PID)	2021/5/21	4	Car
Road Tracing Test & Improvement (Patio 1)	2021/5/22	15	Image
Color Recognition Test & Improvement (Patio 2)	2021/5/23	13	Image
Seminar 9 (Offline, 1h) Week12	2021/5/23	1	All
Car dynamic self-adaption direction maintainance	2021/5/24	10	Car
Ultrasonic Sensor Handrail Tracing (Patio 2)	2021/5/27	7	Image
Seminar 10 (Offline, 30min) Week13	2021/5/30	1	All
Ultrasnoic Beacon Test	2021/6/1	3	Image
Car system optimization	2021/6/4	10	Car
Patio 1 Test & Adjustment	2021/6/6	5	Image
Seminar 11 (Online, 30min) Week14	2021/6/6	1	All
Patio 2 Test & Adjustment	2021/6/8	7	Image
Final Report and Final Lab Book	2021/6/9	14	All
Seminar 10 (Onfline, 30min) Week15	2021/6/13	1	All
Final Presentation	2021/6/16	1	All

Table 6 Time Arrangement

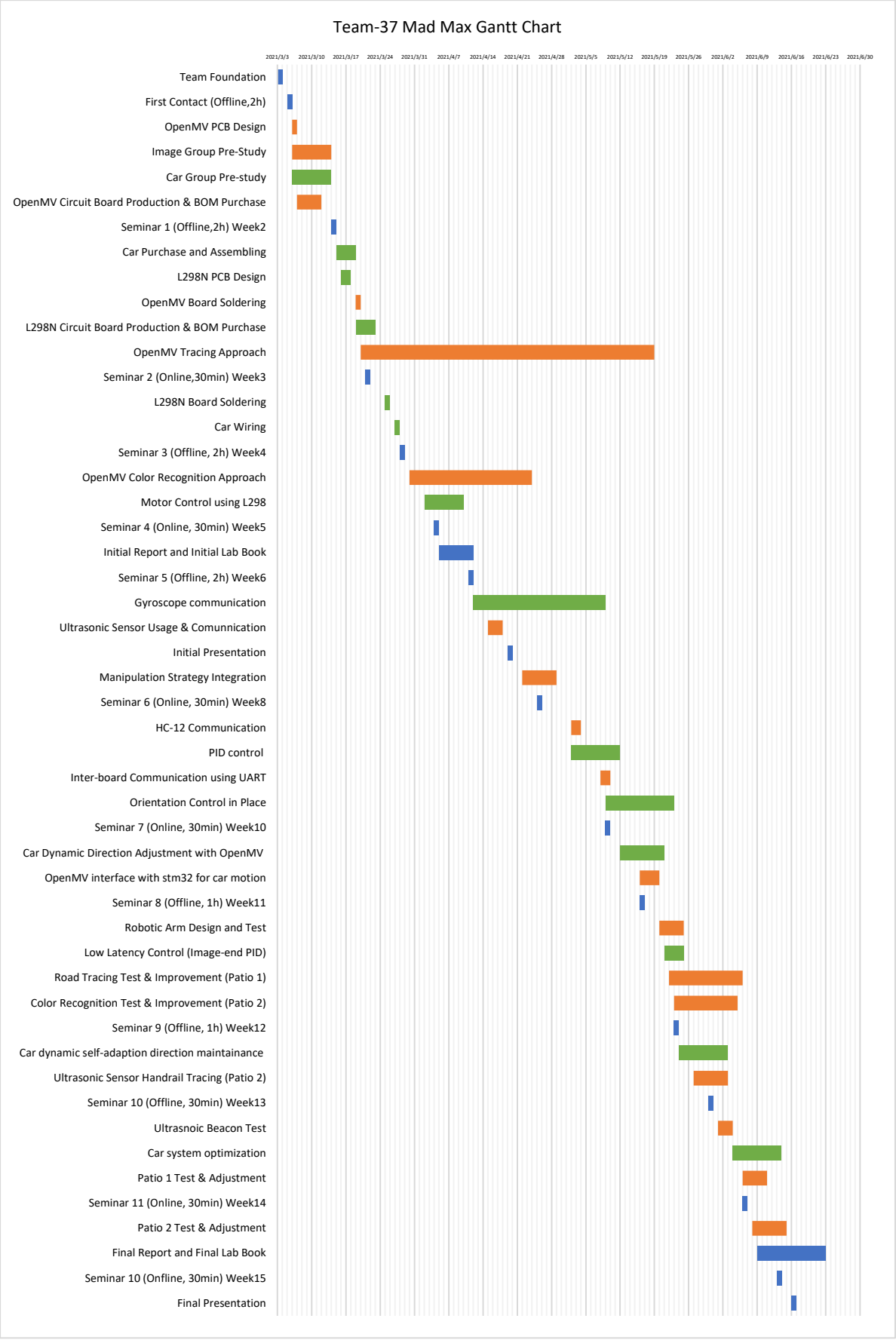


Figure 78 Gantt Chart

Appendix II Bills of Materials

Classification	Name	Quantity	Total Price
Microcontroller & Driver	L298N PCB Production	1	5.4
	L298N BOM	1	27.6
	STM32H743(OpenMV)+Camera	1	147.9
	PyBoard Mini (STM32F411)	1	79
		Cost	**259.9**
Mechanical Part	Car + Motor + Velocity Sensor + Battery	1	390
	Robotic Arm	1	106
	OpenMV Holder+ Steering Engine	1	71.64
		Cost	**567.64**
periphery Modules	GY-901 Gyroscope Module	1	109
	Power Module 12V to 3.3V+5V	1	18
	HC-12 Communication Module (Pairs)	2	28.8
	HC-SR04 Module (Lab)	4	0
	HC-SR04 Module Holders	4	5.4
	DS3231 Clock Module	1	10.15
		Cost	**171.35**
		Total Cost	**998.89**

Table 7 Bills of Materials