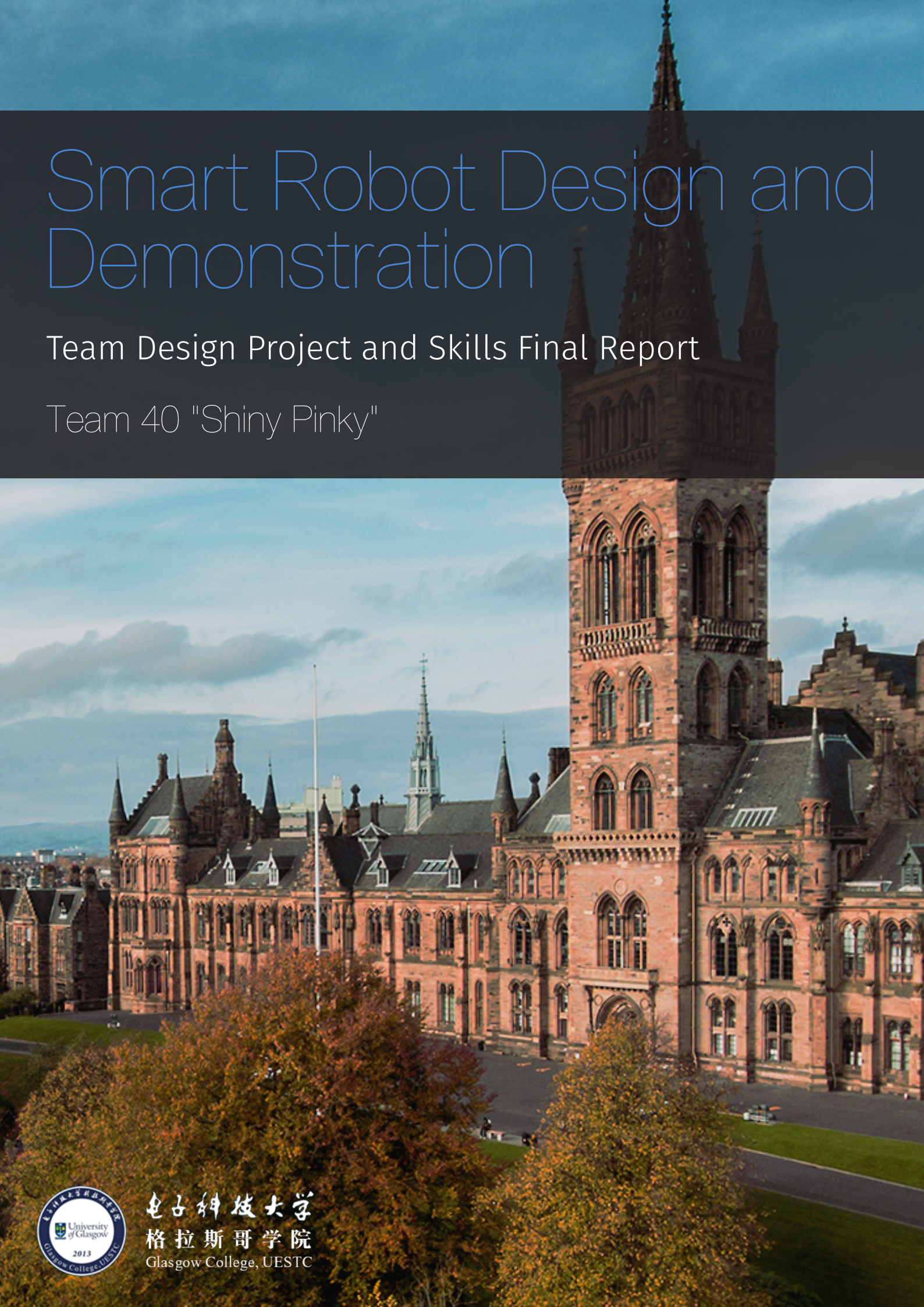
Smart Robot Design and Demonstration

Team Design Project and Skills Final Report

Team 40 "Shiny Pinky"



电子科技大学
格拉斯哥学院
Glasgow College, UESTC

Smart Robot Design and Demonstration

Team Design Project and Skills Final Report

by

Team 40
"Shiny Pinky"

Member Name	UESTC Matric	UoG Matric
Liu Shiyuan	QAQ	QWQ
Yan Jintao	QAQ	QWQ
Jin Jing	QAQ	QWQ
Liu Yilin	QAQ	QWQ
Xie Shuangyang	QAQ	QWQ
Chen Zhuohan	QAQ	QWQ
Dai Yiting	QAQ	QWQ
Wei Zhengjie	QAQ	QWQ
Yao Yifei	QAQ	QWQ
Liang Xuanyu	QAQ	QWQ

Instructors:	Dr. Abdullah Al-Khalidi, Dr. Wasim Ahmad
Institution:	Glasgow College, UESTC
Place:	University of Electronic Science and Technology of China
Project Duration:	March, 2021 - June, 2021

University of Glasgow

电子科技大学
University of Electronic Science and Technology of China

Abstract

Robots are the combination of modern technological breakthroughs and fruits. Capable of performing a wide spectrum of tasks, smart robots have been increasingly helpful in assisting people's daily work and become an indispensable part of modern lives. The *aim* of Team Design Project and Skills (TDPS) is to design and construct smart robots via teamwork. The design scheme features an integrated electronic system which is expected to perform specific functions, and the tasks will be accomplished with collective efforts by a team of ten students. Within a given budget, the team has carte blanche to choose the hardware components necessary to realize its blueprint.

The *main objective* of the project is to build a rover-like smart robot that performs multiple tasks, including line cruising, visual recognition, wireless communication, etc. The robot is generally anticipated to be equipped with microcontroller(s), a chassis and wheels, a motor and a battery pack. The superstructure should be designed dedicately to complete the desired tasks.

The team (*Team 40 "Shiny Pinky"*) proposes and materializes a design scheme for the robot. The design integrates an *mbed* microcontroller that controls the motors, autoloading ejector and other hardware modules with another *STM32* microcontroller embedded in an *OpenMV Cam* module on which the visual perception algorithm is performed. To guarantee better maneuverability on different land textures, the robot adapts tank-like tracks as the driving system and motors power the system to perform the required maneuvers. In addition to the motors and driving system, the ejector and wireless communication module are integrated to the robot as part of its superstructure. The smart robot was tested and improved throughout the final testing days, and managed to complete all required tasks on both patios. The final performance has been evaluated to provide advice for future research and enhancements.

Each team member has contributed to the overall implementation of the robot in materializing the subsystems and integrating the general robot control system. Project management has been excellent and the team schedule has helped team members significantly in coordinating their respective work and completing the tasks in time. This would be invaluable experience of teamwork for every team member.

Acknowledgements

Several months earlier, we concluded the robot rover design and construction project. This project has been a traditional milestone and challenge of every student of Glasgow College, UESTC. Since our entrance to the third academic year, we had heard so much about the possible difficulties and plights that the team may encounter in the span of this robot-building project – project malmanagement, technical impasses and disputes between team members (even toxic when the situation has become very extreme) - from our upper graders. The picture of a painstaking semester-long teamwork project where team members are challenged profoundly and comprehensively could hardly be more vivid in our minds.

Through our collective intellect and efforts, the design task was planned, managed and implemented smoothly and the smart robot constructed by the team has demonstrated seamless performance in the final robot race. The successful conclusion of the design project is a testament to all team members' extraordinary intellect, dexterity and dedication. Here, we want to thank all our companions for their contribution to the smart robot design and implementation. You have played an indispensable role in the project, and pleasure is all ours to share such a great success.

Besides our team members, the implemented robot design would have never been possible without the support from people outside the team, and we want to show our sincere gratitude to all of them. Dr. Abdullah Al Khalidi and Dr. Wasim Ahmad have instructed the overall project management, responded to our questions and given prompt feedback to our progress throughout the semester, and the project progress has become much smoother because of their instructions and advice. Technical aspects of the robot programming are realized mainly on OpenMV MicroPython and STM32 embedded C programming languages, and we would like to thank their developers for making the design possible. Besides, the typesetting of this report is based on the LaTeX template provided by Daan Zwaneveld at `https://dzwaneveld.github.io/report`, with several significant modifications and customized features added to the original template, and a special credit is also given to him. At last, the team want to thank our families, friends and those who have given us support and encouragement in times of hardship. We owe our success to everyone mentioned above.

At the end of this acknowledgement letter, we want to again thank everyone who is involved in or contributed to the project. This teamwork project has brought us together to achieve something that we had never imagined of, and this experience, as a shared memory of a milestone event in our adolescence, has bounded us all. Hopefully this connection would be a perpetual record of what we have achieved and a herald of what we are about to achieve. May these golden hours of a mortal soul live in the immortal spirit of youth.

P.S. To readers of this report, this version is specially recompiled to demonstrate the thorough process of the project for your reference. Notice this document is not to disseminate publicly, but should remain in person-to-person information channels. We hope our report could help you in any aspect of your impending project work.

Team 40 "Shiny Pinky"
Chengdu, January 2022

Contents

Nomenclature

Abbreviations

Abbreviation	Definition	Category
AC/DC	Alternating/Direct current	General Terms
ADC	Analogue-to-digital conversion/converter	General Terms
BN	Batch normalization	Visual Recognition
CNN	Convolutional neural network	Visual Recognition
DAC	Digital-to-analogue conversion/converter	General Terms
DL	Deep learning	Visual Recognition
ENB	Enable pin on modules	Hardware Design
ESL	Equivalent series inductance	Hardware Design
ESR	Equivalent series resistance	Hardware Design
FC	Fully Connected (layer)	Visual Recognition
GPIO	General-purpose input/output	General Terms
I/O	Input/Output	General Terms
IC(C)	Integrated circuit (component)	General Terms
LED	Light emitting diode	General Terms
MCU	Microcontroller unit	General Terms
MSE	Mean squared error	Visual Recognition
OLED	Organic light emitting diode	Hardware Design
PCB	Printed circuit board	Hardware Design
PID	Proportional-integration-derivative (algorithm)	Algorithm
PSNR	Peak-signal-to-noise ratio	Visual Recognition
PWM	Pulse width modulation	General Terms
RAM	Random Access Memory	General Terms
ReLU	Rectified Linear Units	Visual Recognition
RGB	Red-green-blue	Visual Recognition
ROI	Region of interest	Visual Recognition
SAP	Super Absorbent Polymer	Hardware Design
UART	Universal Asynchronous Receiver/Transmitter	Communication

Symbols

Symbol	Definition	Category
P_{out}	Proportional term of PID algorithm	Algorithm
I_{out}	Integration term of PID algorithm	Algorithm
D_{out}	Derivative term of PID algorithm	Algorithm
K_P	Constant non-negative proportional gain	Algorithm
K_I	Constant non-negative integration gain	Algorithm
K_D	Constant non-negative derivative gain	Algorithm
s	Laplacian domain variable	Algorithm
$L[x(t)]$	Laplace transform of $x(t)$	Algorithm
∇^2	Laplacian operator	Visual Recognition
κ	Standard constant $\kappa = 903.3$	Visual Recognition
ϵ	Standard constant $\epsilon = 0.008856$	Visual Recognition

List of Figures

List of Tables

Listings

1 Introduction

The design project is an exclusive teamwork course of the college. This chapter *provides* background information on smart robots, *reviews* the objectives of the teamwork project, and *explains* the structure of the report.

1.1. Background Information

Smart robots have been contributing to the modern society significantly in the past decades. Their have different sizes and functionalities. Some may not resemble real humans in many aspects, but they are more capable of undertaking difficult and dangerous tasks that overwhelm humans. They are combinations of technological breakthroughs and fruits, and designing a robot requires much interdisciplinary collaborations.

The purpose of Team Design Project and Skills (TDPS) is to build the teamwork experience by designing an integrated electronic system able to perform certain tasks and implementing the blueprint to examine its practicability. Each team consists of ten students from both IE and CE programs of Glasgow College, UESTC, and ought to complete the design project on its own. The project objective is specified to the design and construction of a smart robot that could perform multiple assigned tasks on two distinctive patios, including line cruising along a given path, identifying and reacting to different color chunks, carrying and releasing an object, transmitting information. In the meantime, several general rules regarding teamwork, choice of hardware etc. should not be overlooked.

1.2. Project Objectives

The smart robot should complete two distinctive sets of tasks on two patios. At the fruition of the design project, the robot should demonstrate smooth and seamless performance on both patios. The required tasks are briefed in the following section.

1.2.1. Patio 1 Tasks

The preassigned tasking areas on Patio 1 are illustrated in Figure 1.1. On this patio, the robot rover needs to

- Move from the green starting point and follow the path of colored tiles to reach the destination at the red box *[Task 1]*;
- Recognize and cross a blue bridge *[Task 2]*;
- Identify and move through an arch located just in front of the destination *[Task 3]*.

In order to assist the robot to accomplish the tasks, the team is allowed to utilize two visual reference points on the patios. These visual points could be placed at where it seems necessary to further ensure the robot performs the task as anticipated. In this section, the use of visual references (beacons) is not introduced and discussed.

(a) Task 1 **(b)** Task 2 **(c)** Task 3

Figure 1.1: The tasking areas on Patio 1

1.2.2. Patio 2 Tasks

The preassigned tasking areas on Patio 2 are illustrated in Figure 1.2. On this patio, the robot rover needs to

- Start moving from a green starting point, and move to the colored square at from the right-hand side of the diamond;
- Determine the color of the square and move to the corresponding color square, with each color identified corresponding to a different behavior *[Task 1]*;
- Move to the fish food release point at the lakeside in any possible navigation if general rules are not contravened, and release the food into the lake *[Task 2]*;
- Move to the planter area and stop to transmit a message to a laptop. The message received at the laptop should contain
  - team name,
  - team member names,
  - time of day (24-hour clock);
- Proceed with line cruising after successful transmission is indicated to reach the final destination marked by the red box *[Task 3]*.

The transmission is completed via a radio signal at a frequency of 433 MHz at fixed data rate using Wavesens HC-12 wireless transceiver.

(a) Task 1 **(b)** Task 2 **(c)** Task 3

Figure 1.2: The tasking areas on Patio 2

1.2.3. General Rules

The robot rover must be trained and programmed in advance, and the program should be downloaded to the microcontroller(s) beforehand. In the meantime, real-time instructions are prohibited during the final demonstration.

In order to assist the robot to accomplish the tasks, the team is allowed to utilize two beacons as visual references for navigation on each patio. The beacons could be placed at where it seems necessary to further ensure the robot is navigated as anticipated.

The electrical systems and all connections to the circuit components and subsystems must be rugged and reliable. Whereas the majority of the components can be directly purchased online, the team is expected to construct a motor driver circuit on a Printed Circuit Board (PCB) on its own.

The team will be given carte blanche to choose, design and implement hardware modules of the robot rover to complete the assigned tasks should the total cost of the project not exceed 1000 RMB. All sources of financial cost should be clearly recorded, itemized and displayed, and full justification for the choices of components further contribute to an excellent project design and management.

1.3. Report Structure

After this introducing chapter, the author first *presents* the team's overall design approach where tasks on both patios are broken down and the functionalities of the smart robot are decided. This chapter does not include concrete design approaches, but the cardinal ideas of the overall robot design are all illustrated.

In the sequel, the author *explains* the technical solutions which are contributed by each individual team member based on the task breakdown and proposed design scheme. This chapter exhibits each team member's contribution to the implementation and integration of all subsystems, and abounds with technical details which records the team's findings, changes of plan and justifications for the final adapted design.

The next chapter *concentrates on* the results of experiments that the components and modules have been involved as an intact robot, and discusses several prominent issues regrading the performance of the smart robot. This chapter features introspective discussions on the robot design and the potential space for improving the overall stability and robustness of the system.

The report concludes with a short summary, and other pertinent information regarding the system design and teamwork is provided in the appendices. Appendix A focuses on the nuances of program design which are left out in the previous content, and Appendix B focuses on the project management and teamwork.

2

Overall System Design

The smart robot is expected to complete a combination of tasks on two patios. Considering the intricacy of the required design, the team peruses and analyzes the task briefing materials and presents a general design scheme of the rover-like smart robot. In this chapter, the author breaks down the required tasks on both patios, presents and elucidates the overall design approach the team has adapted. The design scheme only presents an outline of the smart robot structure and functions; technical solutions and details are beyond the scope of this chapter.

2.1. Task Breakdown

According to the task briefing in the previous chapter, on Patio 1, the required tasks can be summarized into several categories:

1. Line tracking: travel along the preassigned paths to the target position;
2. Turning: make 90° turns at the turning points on the piece-wise linear paths;
3. Crossing bridge: climb over a bridge placed on the path;
4. Passing arch: drive through an arch conclude Patio 1 tasks.

On Patio 2, the the required tasks can also be summarized into several categories:

1. Line tracking: travel along the preassigned paths to the target position;
2. Color recognition: recognize and find identical color chunks to perform different maneuvers;
3. Fish food release: drop pond pellets into the lake with special devices;
4. Wireless communication: transmit message to the laptop terminal.

Based on the task breakdown above, the team continues to discuss on how to complete the tasks. Before discussing the concrete hardware and programming approaches, the team concentrates on the functional structure of the robot.

2.2. Initial Proposal

The team compartmentalizes the smart robot into five functional sectors – central control, visual recognition, wireless communication, robot arm, motor & driving system, and power supply unit. Their relationships and connections are depicted in Figure 2.1. Since the motors & driving system, robot arm and wireless communication modules are incapable of performing independent computation and data feedback, they can be referred to as a collective group of *peripherals*.

In overview, the central control module is an mbed microcontroller which connects to the remaining parts of the smart robot. The controller receives feedback data from the visual recognition module, conducts operations and algorithms, and gives commands to other sectors. An OpenMV Cam is chosen to be the visual recognition sector, sending feedback data to and receiving commands from the main controller via bidirectional serial communication. The wheels are substituted with a pair of tracks and two motors are adapted to thrust and propel the tracks. The robot arm would be modelled and produced by 3D printing to be suitable for carrying and releasing fish food. As is required in the course information, the wireless communication in Patio 2 would be completed on Wavesens HC-12 wireless transceiver.

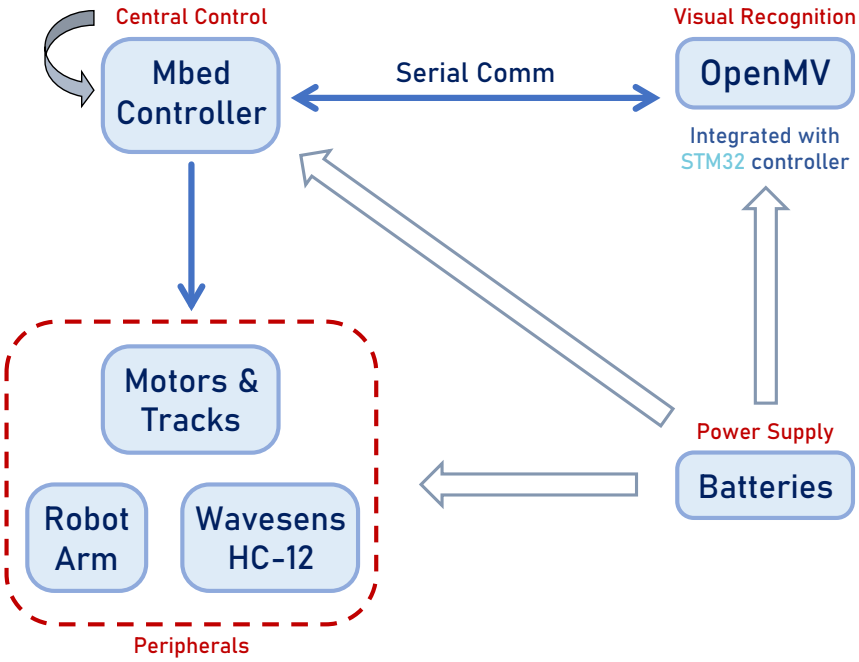


Figure 2.1: The team's initial proposal of hardware selection and design

2.2.1. Central Control

Located at the highest level of the data hierarchy in the smart robot, the sector is the center of decision making, as well as the core of programming inside this sophisticated machine. Taking other modules as "sub-functions" of its own, it receives information from other functional sectors like visual recognition module as input data to perform various algorithms, and the outcomes of these operations would be processed to generate new commands for other functional sectors connected to the main controller. For instance, the navigation data processed and transferred from the visual recognition module would update the input of the driver algorithm that commands the motor and driving system, and after algorithmic computation the system receives a renewed command to perform a certain maneuver. As a result of the maneuver, the visual recognition module experiences drastic input changes. This triggers another process of data processing and feedback, and the central control module would update the inputs, run the algorithms again to refresh commands and information to send to other modules. In short, the central control module is the core of the robot, connecting and commanding other functional sectors to work in coordination.

In this design, the mbed processor is chosen to be STM32L432KC. STM32L432KC provides sufficient ports to connect with the visual recognition module and peripherals. Two Pulse Width Modulation (PWM) output ports are allocated to the motors. One of the Universal Asynchronous Receiver/Transmitter (UART) ports is connected to HC-12 wireless transceiver to enable message transmission, and the other is used to communicate with the visual recognition module, receiving update from and sending commands to OpenMV. Besides, the general-purpose input/output (GPIO) ports could be used to connect and control peripherals such as the robot arm, and a power switch can be implemented using the InterruptIn port.

Besides, the controller's built-in programmability and computing power would satisfy the need to design, test and run algorithms. The team would write the robot control program and algorithms that support decision-making processes in Embedded C which can be integrated to and executed by the microcontroller with ease, and with sufficient computing power the controller can perform operations and run algorithms autonomously to command the remaining sectors.

2.2.2. Visual Recognition

The main function of the module is real-time image capture and processing. It should be integrated with a camera and support autonomous computation to enable color detection and transmitting

feedback information on the identified colors to the main controller for further decision making. In addition, the image processed by the module ought to identify the robot's current heading and the target path for driving. If the measurements fail to agree while the robot is cruising along a preassigned path, the main controller would conduct operations and generate new commands to eliminate the difference in directional readings and navigate the robot back to the correct direction. In general, the visual recognition module processes live image, detects different colors, identifies directional differences and sends feedback data to the central control module.

After much research, the visual module is chosen to be the OpenMV Cam which is programmable in MicroPython. The OpenMV Cam is a machine vision module programmable in MicroPython operating system. It integrates a camera and a STM32 microcontroller, and is capable of performing digital image filtering, face and color tracking, etc. Since the module can be programmed in Python, which the team is familiar with, and its developers "have done the difficult and time-consuming algorithm work", the team initially concurred that the OpenMV Cam meets the requirements of the visual recognition tasks and is comparatively simple to program and operate. The module would be introduced in detail later in this report.

2.2.3. Peripherals

As is mentioned in the first paragraph, the motor & driving system, robot arm and wireless communication module are categorized as *peripherals* because of their incapability of independent computation and data transmission, and they all receive commands from the central control module to perform certain tasks. The motor & driving system comprises a chassis and wheels, and motor(s) powering the wheels, and apparently is responsible for propelling the robot rover and controlling its forwarding directions. In this design project, the robot arm is merely utilized to carry and release the fish food, and the wireless communication module is only used to transmit information on a radio signal. The chassis adapts tank-like tracks to propel the robot, wireless communication module is chosen to be Wavesens HC-12 transceivers, and the fish food release device is a robot arm.

2.3. Final Design Scheme

The final design scheme resembles the initial proposal. The main difference is that the central control unit is switched to the built-in microcontroller of the OpenMV Cam. Several modifications are done on the design to achieve certain functions that support the task completion in the robot race. The schematic diagram in Figure 2.2 shows the main idea of robot design.

The individual mbed microcontroller initially used to connect half of the hardware moudles is reprogrammed to perform one mere task - wireless communication. It controls the HC-12 transceiver and completes the wireless communication. All the remaining tasks are assigned to the STM32 board integrated in the OpenMV Cam module. This requires most of the pins and ports on the visual module be occupied for external connection with other hardware modules, so the hardware resources need to be carefully managed. Additional ultrasonic distance measuring sensor and accelerometer are added to the system. The travelling robot requires live update on the heading and distance parameters to locate and navigate itself, so the measurements of distance and relative direction from the target positions are of great importance. The initial robot arm was later considered to have occupied excessive hardware resources while having unstable performance and difficult maintenance requirements, so it was replaced by an autoloading ejector. The ejector, essentially a electric gearbox, is similar to an automatic gun. Once activated, the gearbox moves back and forward rapidly to eject the fish food.

Besides all the "must-have" components mentioned above, another slot for additional hardware components is also planned for future modifications. This increases the flexibility of the design scheme, as there is additional room for new modules.

As an important part of the robot implementation, the program design of the smart robot which was not included in the initial proposal is also added to the final proposal. As each patio should be completed separately, the robot main program is designed as two separate sections for tasks on each patio. The main structure of robot control programs is explained in the following section.

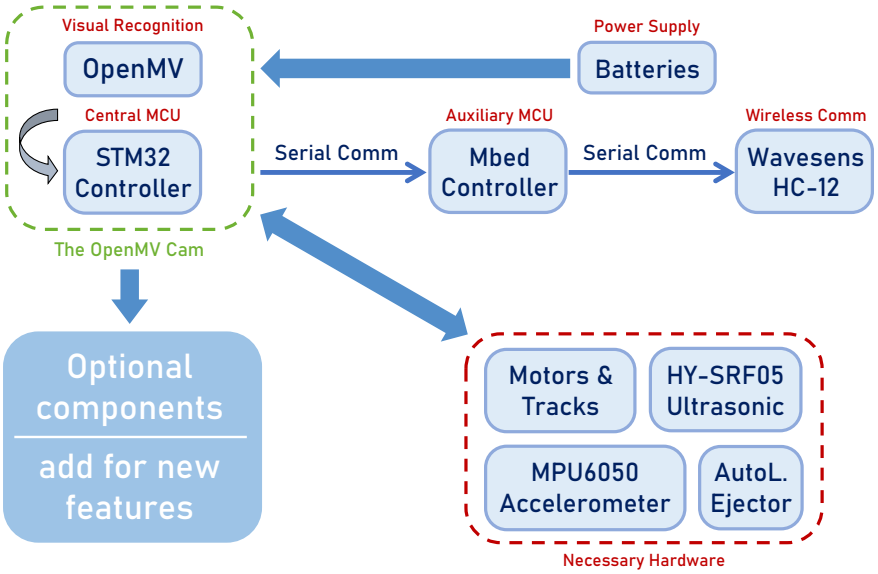


Figure 2.2: The schematic diagram of the final design scheme

2.4. Main Program Structure

The team designs the overall program execution plans before the actual programming. On Patio 1, the OpenMV Cam is first initiated to identify the preassigned paths. Together with the PID direction control algorithm, the smart robot is able to travel along the paths. The ultrasonic sensor is activated at the beginning and once the measured distance falls below a given value, the robot stops line tracking and makes a right turn to cross the bridge. After that, the robot detects the distance and once the distance goes below a given level, it turns left to cross the arch. Figure 2.3 shows the task flow diagram and the execution process.

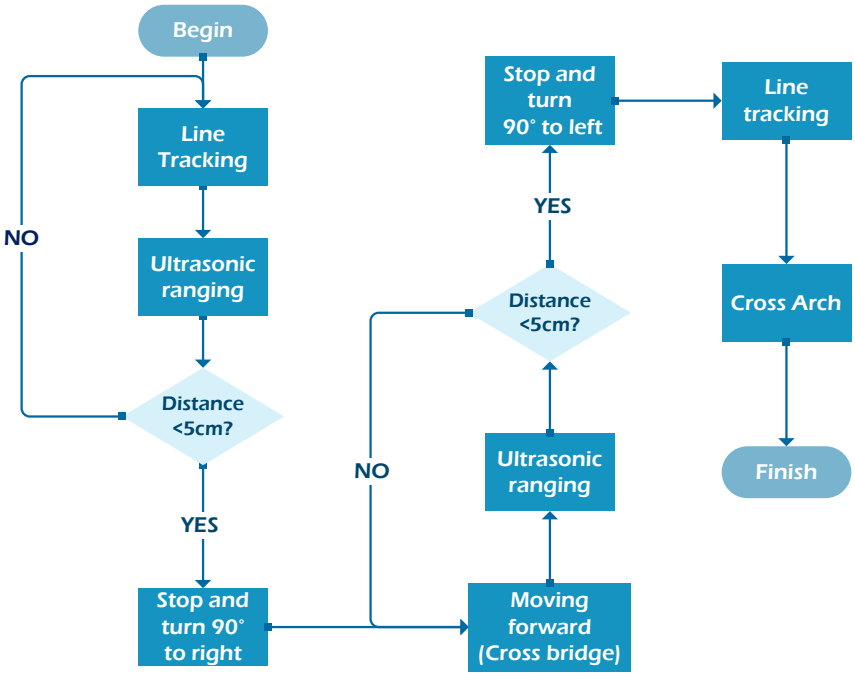


Figure 2.3: The flow diagram of the adapted program design on Patio 1

On Patio 2, the robot needs to complete “color recognition”, “line tracking”, “fish food release” and “wireless communication”. After recognizing the colors, the robot needs to travel over the recognized color chunk, and starts again to arrive at the lakeside for fish food release via preassigned paths. After completing the task, the robot continues to drive to the wireless communication region to transmit messages to the laptop terminal. Figure 2.4 shows the execution process of all tasks.

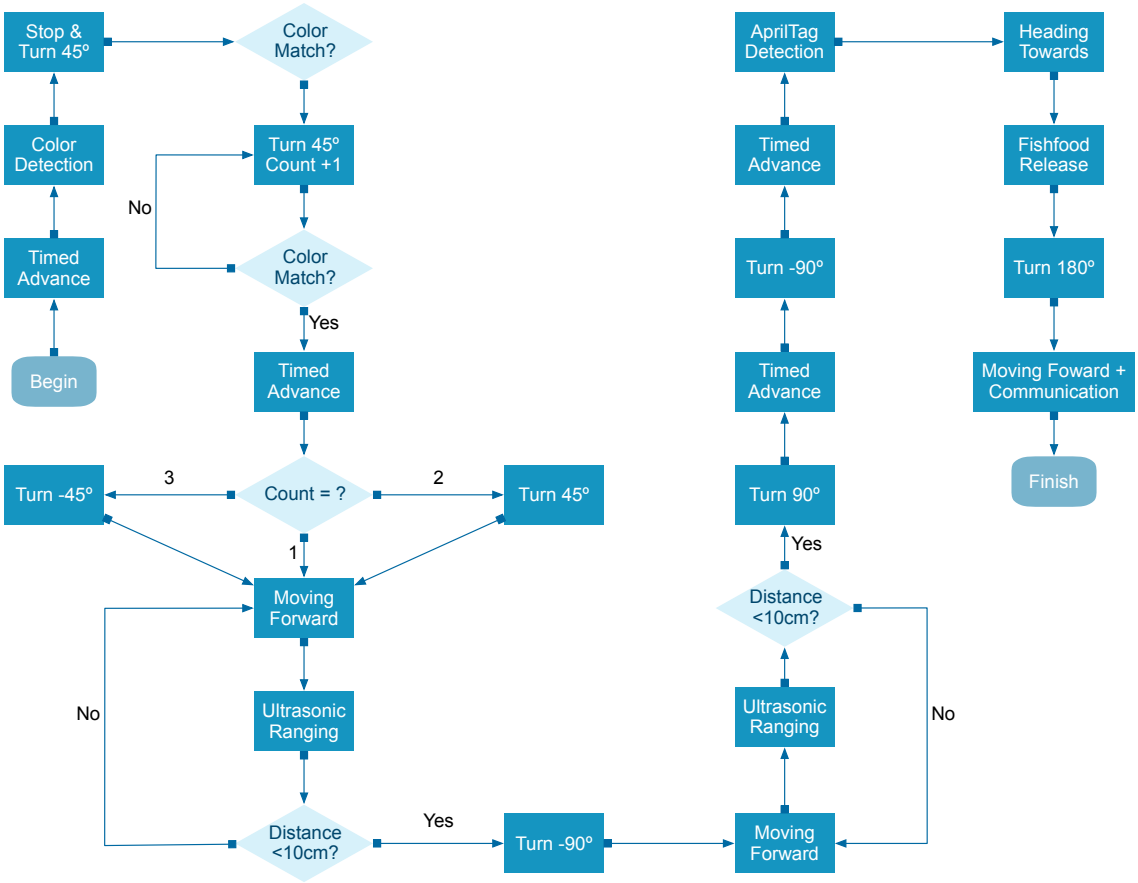


Figure 2.4: The flow diagram of the adapted program design on Patio 2

3

System Implementation

With the initial design scheme, the team proceeds to construct and program the smart robot, implementing, experimenting on and enhancing the design of each subsystem. These subsystems are assembled, further tested and enhanced to optimize the performance. In this chapter, the author *introduces, explains, discusses* and *evaluates* the design attempts and final solutions adapted by the team throughout the span of the project in great detail.

This chapter is group effort, and each individual member's contribution to the overall smart robot design has been clearly marked. Meanwhile, note that final demonstration results and programming details are not fully covered in this chapter: further technical details are attached to other chapters of the report. Relevant information has been marked and linked for convenience if the reader intends to have a more comprehensive view of the project.

3.1. Hardware Design

The smart robot needs to master several important functions to possibly complete the tasks on both patios, and the implementation of these functions requires some key components and modules be materialized, tested and troubleshot. This section concentrates on the physical robot design approaches and elucidates the rationales behind these hardware solutions.

3.1.1. Solution Overview [1]

The overall hardware design and implementation comprises multiple tasks: the motors & tracks, the physical structure of the smart robot, the robot arm (later changed to be an autoloading ejector) used to "release" the fish food.

The main skeleton of the smart robot is a triple-deck structure, and each deck are supported by several metal columns. The first deck, or the chassis of the robot, accommodates the motors and drive circuit. Substituting the traditional wheels, the team installs a pair of tracks on each side of the chassis to enhance the stability of the machine while travelling on rough terrains and the performance while overcoming obstacles. To further guarantee the robot's performance while climbing along a slope or overcoming obstacles, the team utilizes plastic materials to construct the main body of the robot (except the supporting metal columns).

In initial phases of the project, the author proposed that the OpenMV Cam and battery be placed on the second deck and the robot arm be installed on the top deck. This arrangement avoids the difficulty of hardware installation caused by limited headroom on the second deck and guarantees that the robot arm operates normally without any spatial constraints. More importantly, installing the OpenMV Cam on the second deck ensures the normal functioning of the module, because numerous experiments have indicated that excessive height of installation would render the camera unable to accurately capture and recognize the paths and color chunks, which only supports an image definition of 320 × 240 in the working mode adapted in the project.

While experimenting with the initial design, one immediate problem emerged that the OpenMV Cam cannot function as required because it is still installed at inappropriate height and angle. Besides, the center of mass of the robot was considered high because of the top deck robot arm, which

[1] *Wei Zhengjie, UoG Matric: 2429522W, Contribution: Sections 3.1.1 - 3.1.4*

undermines the stability of the machine when travelling on both patios. Although the robot arm was considered sufficiently flexible and versatile for the fish food release task, the complex circuit design and massive space required to implement the robot arm have added to the difficulty of achieving a balanced and well-established physical structure.

To handle these problems, the team abandoned the robot arm design and adapted an autoloading ejector which ejects the fish food in a fast projectile. Since the space required to install the ejector is comparably small, the ejector is installed on the second deck, and the OpenMV is reinstalled on the top deck with a downward angle so that the captured images help optimize the performance of the module. This modification has contributed to a better balanced and arranged structural design.

In the initial design scheme, the team adapted the YX-7015AM chip as the motor control circuit. After several experiments, however, the chip was considered not suitable for the robot. The reasons are that the circuit with the chip has a relatively low respond rate and, more importantly, that the chip operates normally only on DC power sources, while the PID algorithm requires a PWM source to achieve real-time direction control by creating velocity difference on tracks. As a result, the team substituted the YX-7015AM chip with a L298N integrated circuit (IC) unit, which can operate on PWM power sources and achieve differential steering.

The power supply of motors is guaranteed by PWM sources. Once the motors start to rotate, it propels the driving wheels on both sides of the robot and the wheels brings the tracks into a move. As the magnetic brush of the motors might generate electromagnetic pulses that are likely to undermine the functionality of the surrounding electronic components, such as the microcontrollers and as the frequency of PWM power sources increases, the noise caused by the working motors intensifies, the team attaches a decoupling ceramic capacitor in parallel with the motors to mitigate the detriment of the electromagnetic pulses and enhance the stability of power supply.

In the following content, the author explains the cardinal ideas of the hardware design and technical details of the hardware components included in the design scheme.

3.1.2. Metal Column

The columns used in the robot design are made of copper, and are durable and not susceptible to rust. Because the strength of the product structure is Class 4.8, that is, the critical point of deformation is 320MPa. At the same time, there are flexible options of metal columns of different lengths ranging from 4cm to 26cm, so the team could choose the appropriate length to realize the design. This is a great convenience for our work, because it is highly difficult to predict the precise height between each deck at the beginning of the construction, and this also has made future adjustments and modifications much simpler. In our project, four metal columns are used to support the three decks, and the structural strength has been sufficient for the project tasks. Installing and removing these columns are also very simple: they could be removed without using extra tools, and can be installed manually.

Figure 3.1: The metal columns used in robot physical skeleton

3.1.3. Power Source

The power supply is provided by two 3.7V DC lithium batteries. The batteries are connected in serial to offer a total source voltage of 7.4V, as the motors and fish food release device cannot operate

efficiently under a smaller voltage. Compared to other 7.4V batteries, serial-connected batteries provide a larger output power, while it is smaller in size and requires less recharge time. One notable issue of the serial connection pertains to stability of the power source, which might influence the performance of the smart robot on either patio. These are beyond the scope of this section and will be included in the **Results & Discussion**.

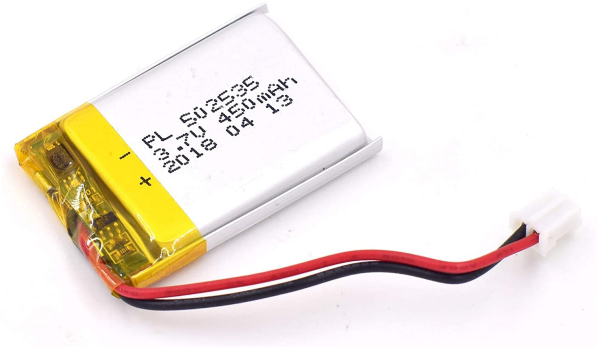


Figure 3.2: The 3.7V lithium battery used as the power source

3.1.4. Tracks

Instead of installing wheels to the chassis, the conventional approach of rover design, the team decides to utilize a pair of tracks as the main body of the driving system. Resembling the chassis of military tanks, the design improves the robot's mobility in various land textures and ability to overcome obstacles while cruising on rough terrains. In this design scheme, two motors are used and each is connected to one track on either side of the machine; each track is powered by an individual motor and thus functions independently. Figure 3.3 shows the conceptual sketch of a track-equipped chassis.

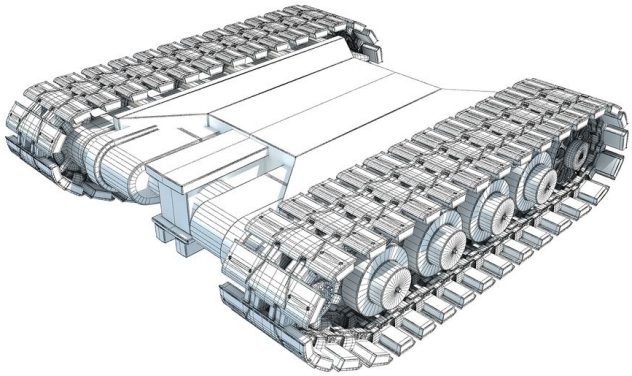


Figure 3.3: The conceptual sketch of track-equipped chassis

To detail the tracks' advantages over the traditional, basic functions of tracks should be reviewed. The tracks are designed to support the vehicle, turn the torque output by the engine through the transmission device into the traction force to drive the vehicle, transfer the braking force from the ground to brake the vehicle, and provide a continuous track surface of the load-bearing wheel, so that the vehicle possesses good trafficability. Compared with the traditional wheels, tracks have high performance during traffic, on bumpy lands, and when climbing on slopes. At the same time, the reliability of tracks is relatively high, and good meshing between components prevents them from falling off the robot, which would be one key contributor to the successful completion of our project

[1]. Comparing both metal tracks and rubber tracks, we choose rubber tracks for the low device load it has exerted on the robot, long service life, low driving noise and excellent shock absorption.

Basic Maneuvers

There are five fundamental maneuvers that the robot is likely to perform: move forward, move backward, turn left, turn right, rotate. The first two maneuvers requires both tracks to move at an identical velocity in the same direction, while the latter two maneuvers are performed by creating a velocity difference on the tracks. The greater the velocity difference is, the faster the turning is, and the highest turning rate is obtained when the tracks are moving at the maximum speed but in opposite directions. All five maneuvers are illustrated in Figure 3.4.

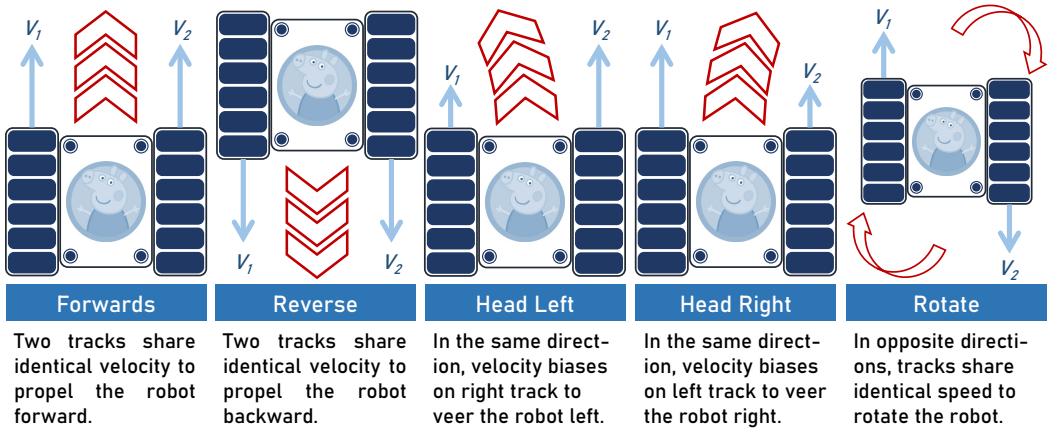


Figure 3.4: Five basic maneuvers of track-equipped robot

As the hardware foundation of the direction control module, the tracks should enable the robot to perform the aforementioned maneuvers required to complete the tasks. To achieve this, each track is powered and controlled by a separate motor to produce different speed at each track. Still, the team needs a motor control circuit that manages output signals from the central controller to achieve the objective.

Motor Control Circuit

The purchased motors work with DC power sources, and the team initially designed and constructed a motor-control circuit mainly with a YX-7015AM controller, which functions flawlessly when powered by direct current sources. YX-7015AM is a DC bidirectional motor drive circuit, which is suitable for toy motor drive, automatic valve motor drive, electromagnetic door lock drive, etc. It has two logic input terminals to control the motor to move forward, backward and brake. The circuit has excellent properties such as good anti-interference, small standby current and low output internal resistance. At the same time, it also has a built-in diode, which can release the reverse impulse current of inductive load. The controller unit has simple structure and functions at a power of 1.998W under a 3.7V DC power source (calculated with the measured working current of 0.54A), with negligible energy loss.

However, as the turning maneuvers of the robot require velocity differences on its tracks, the power supply of the circuit needs to be converted to PWM sources, and later evidence has proven YX-7015AM unsuitable for PWM signal control. In the performance test, when the PWM output signals are set to have a frequency of 1kHz and a duty cycle of 50% and are examined on an oscilloscope, the transition from low to high level in the signals are unexpectedly disordered – some have excessive delays or even fail to transit to the next state, producing no signal pattern that resembles a square wave. One immediate ramification of this finding follows that even if both motors receive identical PWM input signals, the rms values of the input voltages are not equal on both sides and the power of the motors are not identical as well, so the speed of each track will experience random and asymmetrical fluctuations. This way, the robot would travel along zig-zag paths even if it is given commands to move straight ahead.

In the meantime, as was mentioned in a previous section, the team has doubled the output voltage to increase the cruising speed of the robot. This modification of the power source from 3.7V to

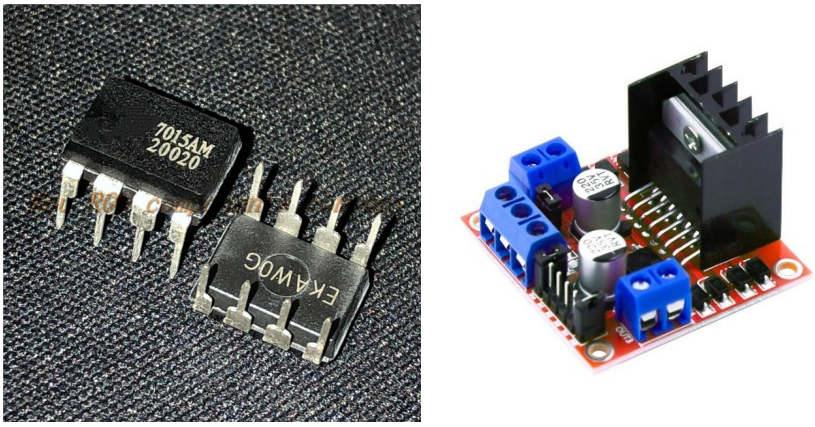


Figure 3.5: YX-7015AM motor driver (left) and L298N motor drive IC unit (right)

7.4V has breached the maximum supported working voltage. Though the module can still function with PWM power sources, it experiences severe overheat and is in immediate peril of burning out. On the other hand, if the frequency of PWM sources is set comparatively low to prevent excessive heating on the controller, the overheat occurs on the motors and because of the serious dissipation of thermal energy, the efficiency of the motors deteriorates drastically.

After much discussion, the team abandoned the scheme and adapted a new circuit design featuring an L298N IC component, on which the velocity differences could be created while the motors experience no excessive heating and other aforementioned problems.

L298N (Multiwatt Vert.) is a high voltage motor driver, and can drive both the DC motor and stepper motor. One L298N driver chip can control two DC reduction motors to do different actions at the same time. It can output 2-ampere current in the voltage range of 6V to 46V, and features overheating self-breaking and feedback detection [2]. L298N can directly control the motor, set its control level through the input of the main control chip, and then it can drive the motor in forward and reverse rotation, with simple operation and excellent stability, which meets the large current driving requirements of DC motor. Figure 3.6 shows the top-view pin configuration of L298N and its connection with the motors.

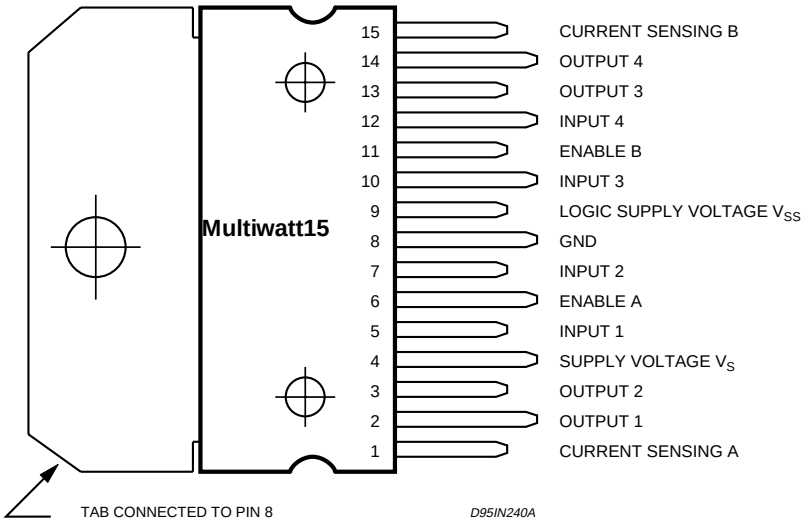
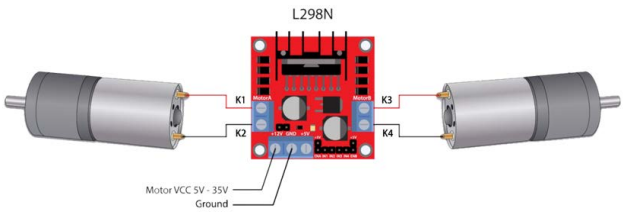


Figure 3.6: The pin configuration of L298N motor drive IC unit [2]

The main purpose of the module is to realize PWM direction control with DC-powered motors. In the connection above, in order to adjust the velocity of tracks with PWM signals, two input ports – IN1 and IN2 – are initialized to be ready for upcoming signals. Once the commands are transmitted,

the circuit outputs PWM signals to the enable (ENB) end and corresponding signals to the input ports to adjust the velocity of the tracks. The truth table of this drive circuit is as follows.



ENB	IN1	IN2	State
0	X	X	Stop
1	0	0	Brake
1	0	1	Forward
1	1	0	Reverse
1	1	1	Brake

Figure 3.7 & Table 3.1: L298N connection with motors and truth table

When the ENB value is 0, the motor is in *stop* state; when the enable signal is 1 and IN1 and IN2 are 00 or 11 respectively, the motor is in *brake* state to prevent unwanted movement.

3.1.5. Printed Circuit Board (PCB) [2]

The performance of L298N controller can be further optimized by adding several more components. These features are expected to be integrated into one single PCB, which is portable and easy to arrange on the robot. The author has proposed the schematic design of the PCB as show in Figure 3.8. In the figure, *U2* represents L298N controller module, the core of motor control circuit, and its circuit can operate with 3.3V input signals from the mbed microcontroller, whose maximum current output could be up to 2 ampere; *U1* represents a 78M05 voltage regulator, which provides a 5V power source to L298N. The LED bubble (*LED1*) beside the regulator illuminates when the power source works normally, and team members could be immediately informed of an anomaly in the power source. *D1, D2,...,D8* are flyback diodes, which are implemented to mitigate the voltage stress at the moment of internal switch changes: since the inductive load of motors are prominent, when the switches open/close inside the controller module, the currents in motors cannot change immediately because of their intrinsic inertia, and this exerts a burdening force on the module.

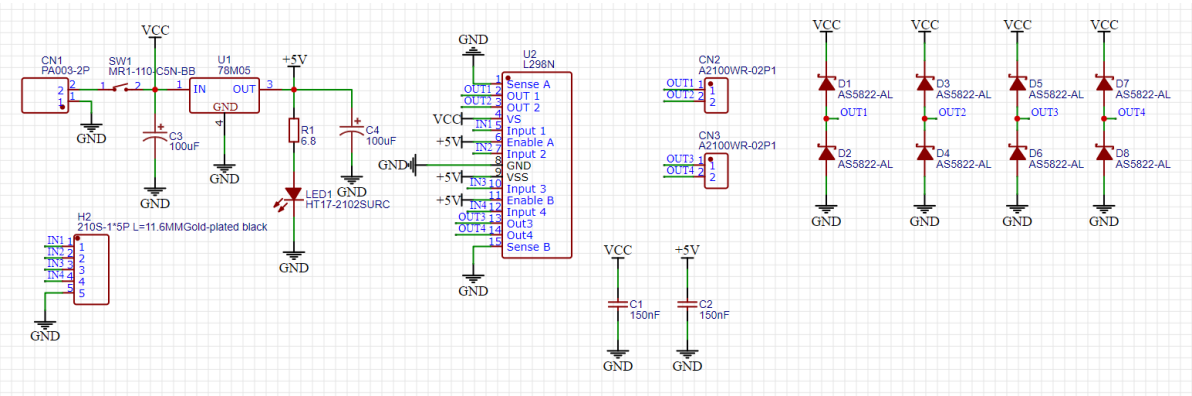


Figure 3.8: The schematic layout of the PCB used to enhance the performance of motors

Having checked the design, the author prints out the PCB and installs the board at the rear part of the chassis, i.e. the first deck. The top view and 3D model of the PCB are shown in Figure 3.8. On the PCB, all output channels are arranged with 100-mil lines. At a temperature lower than 10°C, the circuit's theoretical maximum current load is 4.5A, where there is sufficient headroom to tolerate relatively high current values. The PCB adapts rocker switches to avoid frequent connection and disconnection of lithium batteries, which might cause serious problems because of the inductive load of motors. Besides, aware of the fact that the current flowing through the L298N chip is comparably high and there is a voltage drop of approximately 2V at the chip, the author has designed several slots for the heat sink to handle overheat issues.

[2] *Chen Zhuohan, UoG Matric: 2429526C, Contribution: Sections 3.1.5 - 3.1.6*

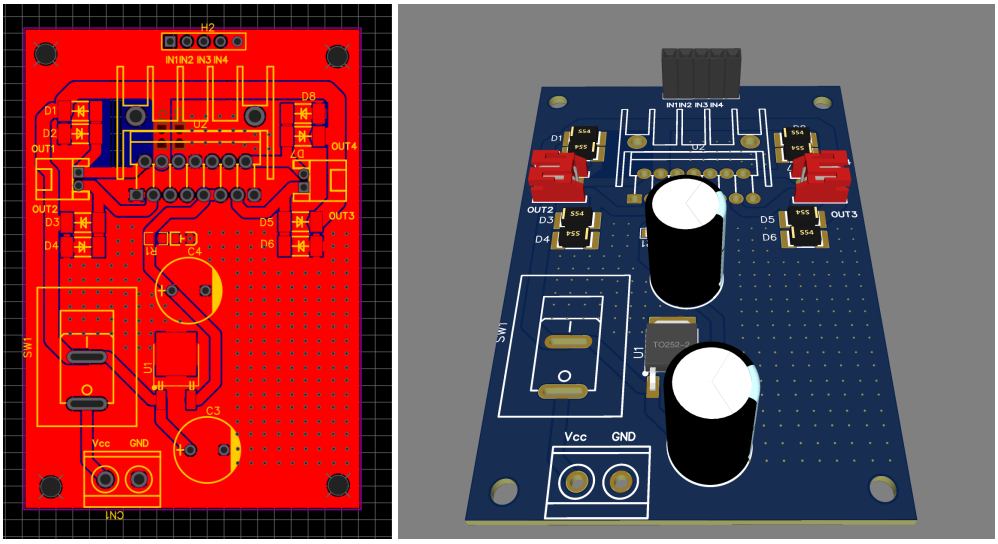


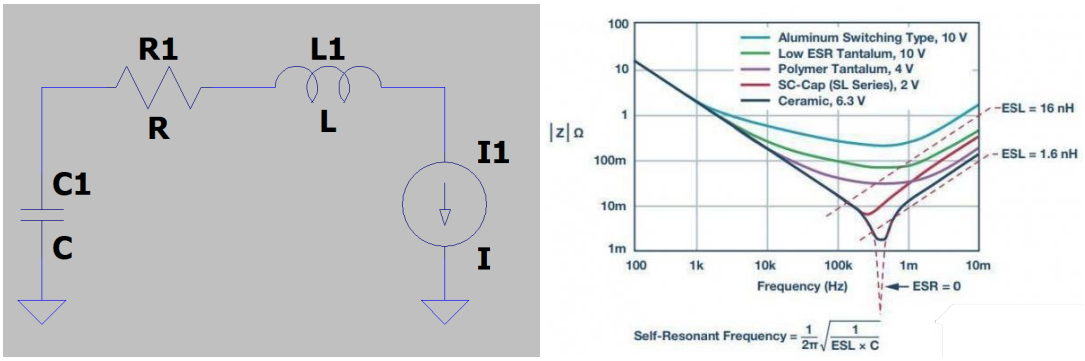
Figure 3.9: The 3-dimensional model of the adapted PCB

3.1.6. Decoupling Ceramic Capacitor

Alongside the sophistication on the motor control circuit, the performance of motors has also been improved from another degree – the overall stability of the driving system. The practical approach to achieve better system stability is to add a decoupling capacitor to the circuit.

To understand the functionality of decoupling capacitor, one should look into the internal structure of motors. The motors contain magnetic brush that might generate electromagnetic pulses. The pulses are likely to impair the circuit's components, such as the microcontroller and L298N module. As the frequency of PWM power sources increases, the noise generated by the operating motors intensifies, and this is detrimental to the system.

The decoupling capacitor aims to provide AC current flows near the load component to mitigate the detrimental effect of the noise. This capacitor is no simple capacitor: it contains equivalent series inductance (ESL) and equivalent series resistance (ESR), and the equivalent circuit arrangement can be depicted as shown in Figure 3.10a. In this case, *I1* is AC current source, and its frequency equals the PWM source frequency. If the overall capacitance of the circuit is sufficiently large to be commensurate to a voltage source, the ESL and ESR are expected to be as small as possible to stabilize the circuit voltage.



(a) The equivalent circuit of the capacitor **(b)** Multiple capacitors compared

Figure 3.10: Technical details regarding decoupling capacitors

The material of which the capacitor is made of is also one important consideration. In Figure 3.10b, several capacitors that share the same capacitance but differ in materials are compared in terms of their impedance with changing frequency. In the figure, the lowest point on each curve corresponds to the resonance frequency of the capacitor, and at this point the impedance is the ESR.

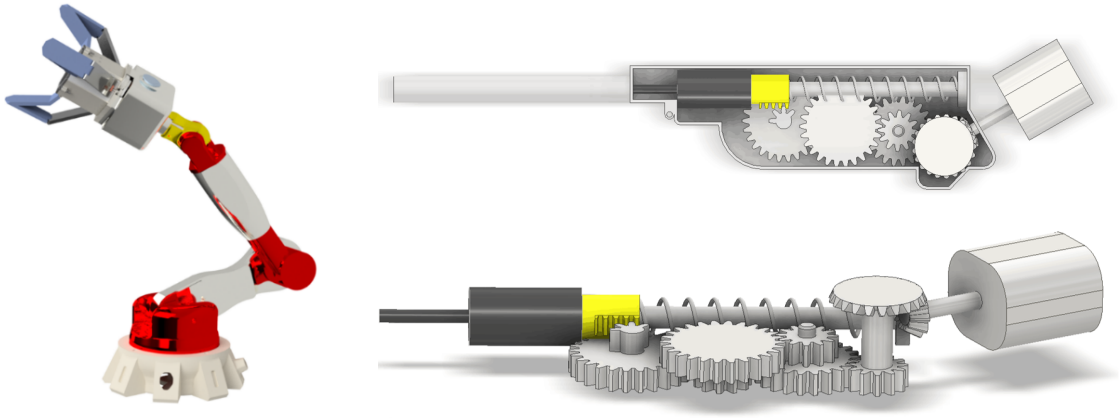
Conclusion can be drawn that the higher the resonance frequency, the lower the ESR; the lower the impedance, the lower the ESR. From these views, it can be indicated from the figure that ceramic capacitors outperform other capacitors in terms of ESR and ESL, so the author chooses a ceramic capacitor to decouple the circuit.

3.1.7. Fish Food Release Device [3]

At the start of the project design, the team tended to construct a robot arm to carry and release fish food. However, as hardware assembly and actual experiments proceeded further, design and construction difficulties emerged and this original proposal was compromised for a mixture of considerations. The new solution is to eject the fish food using an automatic ejector, which is in essence an electric gearbox.

Robot Arm

The role of the robot arm is to complete the transport task, that is, to carry and drop specific items to the specified location. Therefore, we designed a mechanical arm with several mechanical joints. A steering engine is installed at the base, which enables the arm to fine-tune the angle. There is also a steering engine at each joint to ensure that the drop distance of the manipulator can be controlled. Another steering gear is designed to control the opening and closing of the "finger". In the initial design scheme, the robot arm will be fixed at the top of the car. The robot arm's rubber-made fingertip helps to increase the friction and grasp the target items more firmly. At the same time, the main part of the arm is made of plastic materials to reduce the overall weight and ease the slope climbing process. The 3D model of the robot arm is provided in Figure 3.11a.



(a) The mechanical robot arm **(b)** The electric autoloading tube ejector (gearbox)

Figure 3.11: Proposed fish food release devices

As the project proceeded, the team found it difficult to program and arrange the robot arm on the top deck: the device requires multiple I/O ports be programmed to control the mechanical joints, and its large size has left little room for the remaining components. To tackle these issues, a new proposal is formed that fish food, usually in granular shapes, can be ejected from the robot to the target position. Utilizing an electrical gearbox, the team implements an automatic ejector tube that consecutively ejects fish food once activated. Figure 3.11b shows the 3D model of the electric ejector.

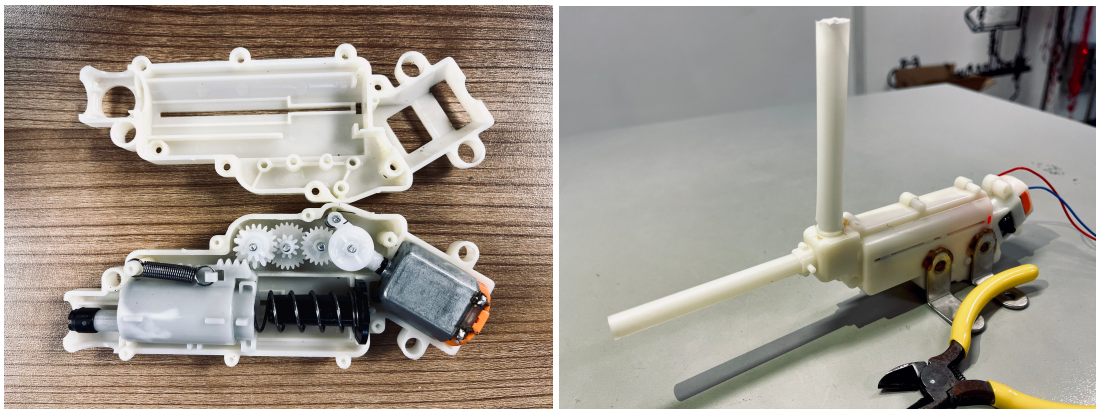
Autoloading Tube Ejector

The device is essentially an electric gearbox. One trigger signal from the central microcontroller activates the gearbox to operate until a 'disengage' command is signaled. The device resembles an automatic rifle, but the trigger directly comes from the microcontroller instead of external human intervention.

While ejecting an object, the piston was powered by the gearbox to spur forward at a high speed, accelerating the loaded fish food before it exits the tube and enters a very fast projectile. A vertical

[3] *Yao Yifei, UoG Matric: 2429227Y, Contribution: Section 3.1.7*

straw is attached on the top of the loading bay as the magazine to store the fish food granules. As is shown in Figure 3.11b, the yellow part is the piston and the black part is the valve. When the gearbox is powered, it will pass through four gears to reduce the speed, increase the torque, and finally transmit the power to the pinion. The important characteristic of a pinion is that only half of its perimeter has teeth. In the toothed part, the pinion drives the piston, and the yellow piston moves backward and compresses the spring. After arriving at the place without teeth, the piston enters a relaxed state, and the spring ejects part of the piston, launching fish food at a very fast speed. Figure 3.12 shows the actual ejector internal layout and the vertical straw magazine the team adapts.



(a) The internal structure **(b)** The vertical straw magazine

Figure 3.12: The actual autoloading ejector adapted by the team

The proposal embraces structural simplicity, and its reduced size has allowed the team to install the device on the second deck without taking excessive amount of space. This way, the mass center falls to a lower level and the stability of the machine increases dramatically. Besides, the overall weight of the device has also been curtailed by one kilogram, thus reducing the burden of motors while climbing a slope.

The ejector is much easier to program and operate than the robot arm. Steering gears attached to the mechanical arm require coordinated efforts from each other to operate the device to complete the task. This demand has added to the complexity of program design and the uncertainty regarding system stability. However, the electrical ejector only needs one pair of I/O ports – the power switch – to operate, so the programming difficulty is reduced to low level: one mere command to output a high-level signal triggers the automatic device to eject the granular fish food.

In practice, the arm structure has limitations in its working efficiency. The maximum reach of the robot arm would fall on a circle centering at its installed position, with a measured diameter of 1 meter, and the arm needs much time to perform each single transport. On the other hand, the ejector can automatically eject fish food at a comparably high rate of reloading, approximately 9 rounds per second, while covering a much wider range of at least 5 meters. Since the objective is to release fish food into the lake, these advantages would suffice to qualify the ejector as a better choice.

Data from initial experiments indicates that the standalone working power required by the ejector is estimated to be 10.73W, a staggering figure that exceeds the maximum input power allowed by the mbed microcontroller and the maximum permitted power of CMOS switches – one MAX4618 module used in the experiment circuit was burnt because of the excessive current power produced by the operating gearbox.

After much discussion, the team adds another MX1508 drive component to support the gearbox. This module could be viewed as a simplified version of L298N, and significantly improves the compatibility of the ejector with the remaining hardware components. Despite this change, the ejector still only needs two ports to connect with the microcontroller.

Ejector Installation

The team has designed two positions to install the ejector: the outer left side of the robot and the middle of the second deck. Both positions are designed to bring convenience to the final demonstration. If the ejector is installed at the left side of the robot, the overall mass distribution of the

robot components is asymmetrical, causing the machine to bank left while travelling straight ahead. Therefore, the motor control program needs to be calibrated to correct the intrinsic deviation before entering a new patio. On the other hand, if the device is installed on the middle second deck, the mass center is rebalanced but it is difficult to arrange the wires of the internal circuits. Since both installations have pros and cons, the team needs to decide which to adapt according to the environmental conditions of the final demonstration.

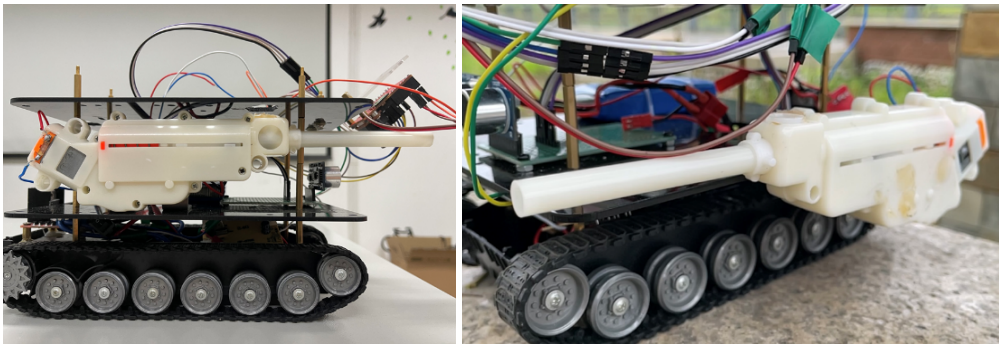


Figure 3.13: Two slots designed for installing the ejector

3.2. Direction Control (PID) Algorithm [4]

Since the smart robot is tasked with path tracking on Patio 1 and cruising on rough terrains on Patio 2, the machine should be able to adjust course according to the difference between its current heading and the target direction. To realize this function, the team introduces the proportional-integration-derivative (PID) algorithm to enable the robot to automatically adjust the current heading to remain on the target path.

3.2.1. PID Controller Theory

A proportional–integral–derivative controller (PID controller or three-term controller) is a control loop mechanism employing feedback that is widely used in industrial control systems and a variety of other applications requiring continuously modulated control [1]. In its normal functioning, it automatically applies an accurate and responsive correction to a control function, which can not only accelerate the response speed of the system, reduce the oscillation, overcome the overshoot, but also effectively eliminate the static error, and greatly improve the static and dynamic quality of the system. The correction provided by the algorithm is based on three terms of system control:

1. **Proportional:** an output value proportional to the current error value (multiplied by a constant proportional gain), denoted as
$$P_{out} = K_P e(t),$$
where K_P is the constant non-negative proportional gain and $e(t)$ is the instantaneous error at current moment t.
2. **Integration:** the accumulated instantaneous error over time (multiplied by a constant integral gain), denoted as
$$I_{out} = K_I \int_0^t e(\tau) d\tau,$$
where K_I is the constant non-negative integral gain.
3. **Derivative:** the slope of the error over time and multiplying this rate of change by the derivative gain, denoted as
$$D_{out} = K_D \frac{de(t)}{dt},$$
where K_D is the constant non-negative derivative gain.

[4] *Liu Yilin, UoG Matric: 2429448L, Contribution: Section 3.2, 3.5.3 - 3.5.6*

Define the system output as $u(t)$, and the overall control function is the contribution of all three terms mentioned above, that is

$$u(t) = P_{out} + I_{out} + D_{out} = K_P e(t) + K_I \int_0^t e(\tau)d\tau + K_D \frac{de(t)}{dt}.$$

The standard form of this equation is given by

$$u(t) = K_P \left(e(t) + \frac{1}{T_I} \int_0^t e(\tau)d\tau + T_D \frac{de(t)}{dt} \right),$$

where the original constant gains are replaced as K_P/T_I and $K_P T_D$. The apparent physical interpretation of T_I and T_D is respectively the integration time and derivative time [3]. Another useful variant of the PID controller output function is the equivalent form in Laplacian domain, namely

$$L[u(t)] = U(s) = \left(K_P + \frac{K_I}{S} + K_D S \right) E(s).$$

In addition, while constructing such a PID controller, it is viable to discuss each term separately, and they are usually referred to as *P, I* and *D* controls in the context. Not every term has to contribute to the correction of output value: a *PID* controller, for instance, could be used as a *PI* controller where *D* control causes problems regarding amplified noise and clashes with control operations.

3.2.2. Practical Application

After reviewing the theories and analyzing the task requirements, the team proposes a direction control model backboned by the PID algorithm, as depicted in Figure 3.14. The control system loop starts by subtracting the current output angle with a preassigned reference angle to find the instantaneous error at current moment, and the error value is inputted to three control components – *P, I* and *D* respectively – to calculate the corrected angle. The robot's heading is then updated to be the calculated angle value. The algorithm iterates throughout the entire process.

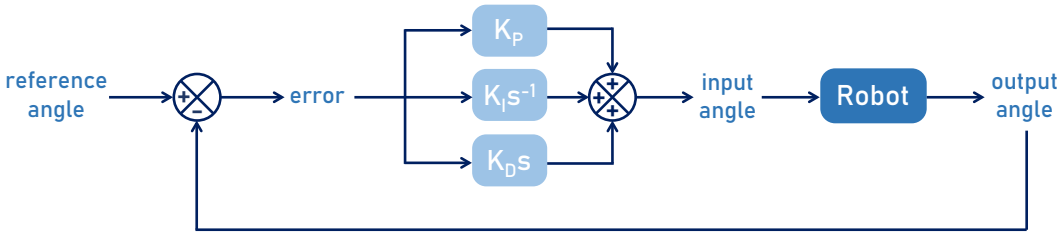


Figure 3.14: The calculation process of PID algorithm

3.2.3. Program Design

Now that the author has reviewed the theory of PID controller, the algorithm could be implemented using Python. In the program, a new data class is first defined to carry all necessary parameters in the algorithm, then the three terms are defined one by one.

Data class The team discusses the PID algorithm and writes a program to implement the model. To start with, the team defines a new data class carrying all necessary information required by the algorithm and initializes the three control parameters **`kp, ki, kd`**, as shown below.

Listing 3.1: Defining control parameters

```
class PID:
    _kp = _ki = _kd = _integrator = _imax = 0
    _last_error = _last_derivative = _last_t = 0
    _RC = 1/(2 * pi * 20)
    def __init__(self, p=0, i=0, d=0, imax=0):
```

```
        self._kp = float(p)      # proportional gain constant
        self._ki = float(i)      # integral gain constant
        self._kd = float(d)      # derivative gain constant
        self._imax = abs(imax)   # auxiliary variable in I control
        self._last_derivative = float('nan')   # auxiliary variable in D control

# defines new data class that receives and carries information required by the algorithm to
    ease data operation
```

P control The output of P control is proportional to the input error signal and can greatly reduce the error between the current output and the target output. The effectiveness of proportional control is not only related to the error $e(t)$, but also depends on the proportional constant **kp**. The smaller the proportional constant **kp**, the smaller the control effect and the slower the system response; on the contrary, the larger the proportional constant **kp**, the stronger the control effect, and the faster the system response. However, excessive **kp** results in large overshoot and oscillation, contributing to poor stability of the system. Therefore, the value of **kp** should be chosen according to the characteristics of the controlled object to compress the static error of the system to a tolerable range, meanwhile guaranteeing a faster response speed. In this case, P control is programmed as follows.

Listing 3.2: Preparing P control

```
self._last_t = tnow
delta_time = float(dt) / float(1000)
output += error * self._kp

# programs the proportional control section individually
# this part describes the proportional contribution to the overall function output, and seems
     like the basis for all other controls...
```

I control In I control, the output of the controller is proportional to the integral of the input error signal. The integral control effect pertains to the presence of error $e(t)$. As long as there is deviation in the system, the integral control will continue to function, integrating the input error, so that the output of the controller and the input of the system loop will vary continuously, reducing the system's deviation. The effectiveness of I control depends on the integral time constant t: the greater the constant t, the weaker the integral effect, and vice versa. However, if the integral effect is too strong, the system will experience increased overshoot and oscillation. Since the error of the integral term increases with time, the integral term will increase significantly with time despite a relatively small initial error. As the control operation iterates, the output of the controller increases and the steady-state error further decreases until it approaches zero. Based on the discussion, the team programs I control as follows.

Listing 3.3: Preparing I control

```
if abs(self._ki) > 0 and dt > 0:
    self._integrator += (error * self._ki) * scaler * delta_time
    if self._integrator < -self._imax: self._integrator = -self._imax
    elif self._integrator > self._imax: self._integrator = self._imax
    output += self._integra

# programs the intrgral control section individually
# this part describes the intrgral contribution to the final system output, and can be switch
     to zero if desired...
```

D control In D control, the output of the controller is proportional to the differential of the input error signal (i.e. the slope of the error function). The PI controller may oscillate or even lose stability when correcting the error, because components and processes may have large inertia or delay. Since these components tend to restrain errors from increasing, their changes always fall behind the change of errors. One solution to eliminate this delay is to reduce error ahead of time. When the error approaches zero, the tendency to restrain errors will cease to exist. Therefore, in addition to introducing the "proportion" term to amplify the error amplitude, we also need to add the "differential

term" to predict the trend of error change. This way, with such PD controllers, the error restraining tendency could be adjusted to zero or negative values in advance to prevent overshoot.

Listing 3.4: Preparing D control

```
if abs(self._kd) > 0 and dt > 0:
    if isnan(self._last_derivative):
        derivative = 0
        self._last_derivative = 0
    else:
        derivative = (error - self._last_error) / delta_time
    derivative = self._last_derivative + \
                          ((delta_time / (self._RC + delta_time)) * \
                              (derivative - self._last_derivative))
    self._last_error = error
    self._last_derivative = derivative
    output += self._kd * derivative
output *= scaler

# programs the derivative control section individually
# this part describes the intrgral derivative to the final system output, and can be switch
    to zero if desired...
```

With these preparations done, the program is ready for testing and troubleshooting in practice. The computer simulation results on the system and experiment data collected from actual robot runs is displayed and discussed in **Results & Discussion**.

3.3. Visual Recognition

After adequate online research, the author proposes using the OpenMV Cam module to achieve the desired visual recognition functions. The OpenMV Cam is a machine vision module programmable in MicroPython operating system. It integrates a camera and a STM32 microcontroller, and is capable of performing digital image filtering, face and color tracking, etc. Since the module can be programmed in Python, which the team is familiar with, and its developers "have done the difficult and time-consuming algorithm work", the team initially concurred that the OpenMV Cam meets the requirements of the visual recognition tasks and is comparatively simple to program and operate. In the following content, the author introduces the OpenMV Cam module in detail, explains the algorithms used to implement the visual recognition functions and the rationale of the team's decisions.

3.3.1. The OpenMV Cam [5]

OpenMV is an open source, low cost and powerful machine vision module. It integrates an STM32F427 processing unit as the core, OV7725 camera chip as the image input source. On this small hardware module, the core machine vision algorithm is implemented efficiently with C language, while allowing convenient Python programming interface. Users (including inventors, amateurs and intelligent device developers) can take advantage of the machine vision function provided by OpenMV in Python language to increase the distinctive competitiveness of their products and inventions.

The machine vision algorithms on OpenMV include color chunk searching, face detection, eye tracking, edge detection, sign tracking etc. Its application has been widely seen in illegal intrusion detection, defective product screening, tracking fixed markers and numerous other fields.

The advantages and prospect of application of the OpenMV Cam are salient. Users only need to write simple Python codes to complete a variety of machine vision related tasks. The delicate design has increased the module's functional versatility, allowing it to be applied to many innovative products. For example, installing OpenMV modules can enable robots to perceive the surrounding environment, and visual line tracking functions can be integrated into the smart vehicle systems. Furthermore, it would support defective product quality monitoring to the factory production line.

The STM32 core processor that the OpenMV Cam integrates accommodate adequate hardware resources for further development and innovation. The module is design with sufficient external sockets to allow UART, I2C, SPI communication and PWM, ADC, DAC and GPIO signals, which makes it more convenient to connect peripherals to the microcontroller and expand the module's functions.

[5] *Yan Jintao, UoG Matric: 2429276Y, Contribution: Sections 3.3.1 - 3.3.2*

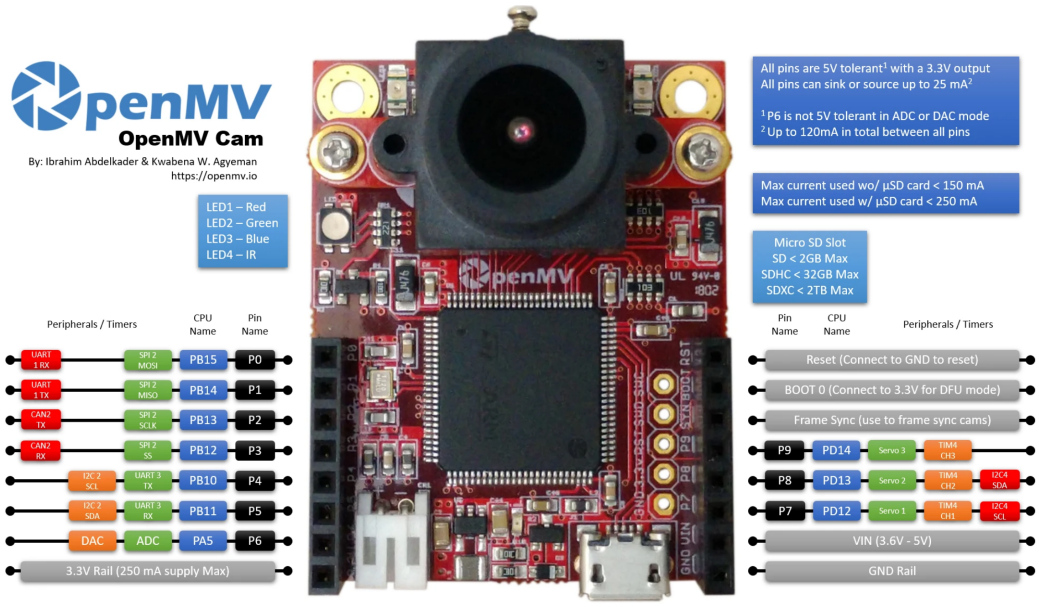


Figure 3.15: The top-view pin configuration of the OpenMV Cam

The USB interface is used to connect the integrated development environment of OpenMVIDE on the computer to support Python programming, debugging and firmware updating. TF card slot supports large capacity TF card that could be used to store programs and save photos.

The OpenMV Cam resembles an "Arduino" board with machine vision function. It can control other hardware through UART, I2C, SPI, AsyncSerial and GPIO, and even MCU modules, such as Arduino, Raspberry PI, etc. In fact, it can also be controlled by other MCU modules. This feature brings greater flexibility to cooperate with other popular modules to achieve advanced product functions.

In this design, the team utilizes the OpenMV Cam to achieve the visual recognition tasks – mainly line tracking, color detection and beacon recognition. Relevant technical details are elucidated in the remaining content of this section.

3.3.2. Line Tracking

Line tracking is an essential function of the smart robot on both patios. The overall process of executing the function involves edge detection which identifies the target path, calculating the deviation angle and distance and applying the directional control algorithm to adjust the robot's heading such that it travels along the target path. This task can be broken down into two main procedures: *edge detection* and *path angle calculation*.

Edge Detection

There have been a great variety of eligible algorithms to implement edge detection. The modern ways of edge detection may be grouped into two main categories - gradient and Laplacian methods [4]. The gradient method detects the edges by identifying the maximum and minimum in the first derivative of the image. The Laplacian method searches for the zero crossings in the second derivative of the image to find edges. To decide on the edge detection algorithm to use, the author introduces and compares several popular options of edge detectors – Prewitt, Sobel, Roberts, Canny and Marr-Hildreth (LoG) operators – to justify the team's choice.

Robert's cross operator Roberts, Sobel and Prewitt operator methods are all based on gradient operations. Roberts (cross) operator method uses the local differential operator to identify the edges. The simple approximation to gradient magnitude-based operator provides a 2 × 2 neighborhood of the current pixel [5]. The approximation to the gradient magnitude can be written as

$$G[f(i,j)] = |f(i,j) - f(i+1,j+1)| + |f(i+1,j) - f(i,j+1)|.$$

With convolution masks, the approximation can be simplified as

$$G[f(i,j)] = |G_x| + |G_y|,$$

where G_x and G_y are the convolution masks defined as

$$G_x = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \quad G_y = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}.$$

In the matrices above, G denotes the gradient, and x, y are respectively the horizontal and vertical mask axis. G_x and G_y are combined to identify the gradient at each pixel point. As is traditional in 2×2 gradient operations, the differences are computed at the interpolated point $(i + 1/2, j + 1/2)$ rather than at the point (i, j) [4].

Sobel operator This algorithm is based on first derivative convolution, analyzing derivatives and computing the gradient of the image intensity at every single pixel. After the calculation, it gives the direction in which the image intensity at each point surge from light to dark. Like Robert's cross operator, Sobel operator is gradient operation based and has two convolution 3×3 matrices (kernels), one estimate gradient x-axis, i.e. rows and another one evaluates y-axis slop, which are columns [5]. Its convolution masks are defined as

$$G_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, \quad G_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}.$$

In the matrices above, G denotes the gradient, and x, y are respectively the horizontal and vertical mask axis. G_x and G_y are combined to represent the gradient at each pixel point, namely

$$|G| = \sqrt{(G_x)^2 + (G_y)^2}.$$

One important feature of the operator is that it is placed on emphasizing pixels that are closer to the center of the mask.

Prewitt operator The Prewitt operator highly resembles Sobel operator in that it is gradient based operator which estimates the first-order derivative. In fact, their mere distinction is the convolution masks: Prewitt operator has the convolution masks defined as

$$G_x = \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}, \quad G_y = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}.$$

The matrices above indicate that this operator, unlike Sobel operator, does not place any emphasis on pixels closer to the center of the masks [4].

Canny operator The essence of the method is simply to extend a traditional intensity-based edge detector into the color space, and seeks to take advantage of the known strengths of the traditional edge detector and tries to overcome its weaknesses by providing more information in the form of three-color channels instead of a single intensity channel [6]. The Canny Edge Detection Algorithm includes the following steps:

1. *Smooth* the image with a Gaussian filter;
2. *Compute* the gradient magnitude and orientation using finite-difference approximations for the partial derivatives;
3. *Apply non-maximum suppression* to the gradient magnitude, and *use* the double thresholding algorithm to detect and link edges.

Canny edge detector approximates the operator that optimizes the product of signal-to-noise ratio and localization. It is generally the first derivative of a Gaussian. The method is the current standard for intensity-based edge detection [6].

Marr-Hildreth detector Marr-Hildreth uses the Gaussian smoothing operator to improve the signal response to noise, which is differentiated by the Laplacian of Gaussian called the LoG operator [7]. The LoG operator is given by

$$\nabla^2 g(x,y) = \frac{1}{\sigma^2}\left(\frac{x^2+y^2}{\sigma^2} - 2\right)\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right).$$

It is a gradient based operator which uses the Laplacian to take the second derivative of an image. The idea is that if there is a step difference in the intensity of the image, it will be represented by in the second derivative by a zero crossing. The general Marr-Hildreth algorithm can be summarized as the following steps:

1. Use a Gaussian to *smooth* the image, which reduces the amount of noise;
2. Apply a *two-dimensional Laplacian* to the image (function), namely

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2};$$

3. Loop through every pixel in the Laplacian of the smoothed image and identify sign changes. If a sign change is detected and the slope across this sign change is greater than some threshold, mark this pixel as an edge and proceed with the remaining pixels.

To evaluate the performance of the aforementioned edge detection techniques, D. Poobathy and R. Manicka Chezian define and utilize two criterions – Mean Squared Error (MSE) and Peak Signal to Noise Ratio (PSNR) – in [5]. The MSE measures the average squared difference between the estimator and the parameter, and specifies the average difference of the pixels throughout the original ground truth image with edge detected image. It is defined as

$$MSE = \frac{\sum_{M,N}\left[I_1(m,n) - I_2(m,n)\right]^2}{M,N},$$

where I_1 is the original image, I_2 is the edge detected image, m and n are respectively the height and width of the image. Intuitively, the MSE should be smaller when optimizing the performance of the algorithm, but greater MSE could exist to guarantee that it finds more edge points on the image and can also detect weak edge points. This indicates that simple comparison of the MSE of edge detectors is insufficient to conclude which algorithm has generally better performances. Therefore, the PSNR is introduced to provide another dimension of evaluation in the general picture of edge detection.

The PSNR refers to the ratio between the maximum possible power of a signal and the power of noise that affects the fidelity of its representation. The quantity is calculated and given in terms of the decibel scale by

$$PSNR = 10\log_{10}\left(\frac{R^2}{MSE}\right),$$

Where R is the maximal variation in the input image data, and $R = 2^n - 1$ for an n-bit unsigned integer data [5]. Although a higher PSNR generally indicates that the reconstruction is of higher quality in image compression, but in some situations like edge detection, PSNR should be smaller to achieve proper results.

Combined evaluation using both MSE and PSNR can offer a rough opinion on the overall performance of an edge detecting algorithm. The experimental data and comparison in [5] are cited in Figure 3.16. As we can see in the figure, Canny operator has the smallest PSNR and the largest MSE, and according to the aforementioned criteria, it has generally better performance than other operators. Although numerous experiments have indicated that Canny operator is more costly and time consuming compared to its counterparts [4][5], these drawbacks do not constitute the central concern of the project design and they would be very unlikely to cause a major malfunction. Overall,

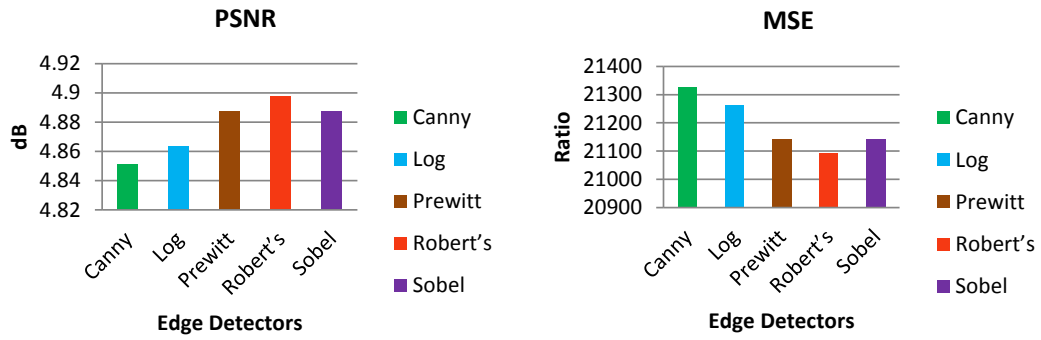


Figure 3.16: The PSNR and MSE of five different edge detectors in [5]

the author highlights the conclusion that *Canny operator has optimal performance* and is considered suitable for the edge detection task.

Now that the team has decided to apply Canny edge detector, the author proceeds to implement the algorithm. Programming the algorithm requires the following steps:

1. *Calculating* the gradient magnitudes and identifying all possible edge pixels,
2. *Thinning* the edge to single pixel width with non-maximum suppression,
3. *Filtering* the edge pixels with a low and high threshold such that pixels whose gradient values are higher than the high threshold value is marked as 'strong edge pixel', those whose gradient values fall between the high and low threshold values area marked as 'weak edge pixels', and those whose gradient values are lower than the low threshold are suppressed,
4. *Removing* weak edge pixels caused by noise/color variations by hysteresis.

Instead of programming from scratch, the author finds an integrated **`image`** library in the documentation of MicroPython utilized by the OpenMV Cam. This library provides an edge-finding function which can be directly used with the format

`image.find_edges(edge_type[, threshold])`,

where **`edge_type`** can be one of the two predefined options:

- **`image.EDGE_SIMPLE`**: Simple thresholded high pass filter algorithm,
- **`image.EDGE_CANNY`**: Canny edge detection algorithm.

Note that **`threshold`** is a two valued tuple containing a low threshold and high threshold. You can control the quality of edges by adjusting these values. Its default value is $(100, 200)$. [8]

By using the function immediately after the image was converted to grayscale, the edge detection was realized. This programming simplicity is a testament to the convenience of the OpenMV Cam.

Path Angle Calculation

Having identified the edges of the path, the smart robot needs to identify the path directions and obtain a rough angle at which the robot should turn in order to remain on the path at the current moment. The author proposes two methods featuring respectively the use of deep learning methods and geometric centroid. The approaches are introduced and explained in the following section.

Deep Learning (DL) Solution To enable the smart robot to travel along the given paths, the most straightforward idea is to simply "teach" the robot to visually recognize the situations where it should turn left, turn right and go straight ahead, and one of the most popular and direct approaches to achieve such intelligence is to train the robot with a convolutional neural network (CNN). Before the robot is able to recognize the paths and make decisions on its own, the machine needs to be trained with a sufficiently large dataset. This DL application requires much time and computing

power for both training the visual recognition model and visual recognition per se, but its accuracy and reliability excel in many aspects once the robot is properly trained.

Convolutional networks are a specialized type of neural networks that use convolution in place of general matrix multiplication in at least one of their layers [9]. A convolutional neural network consists of an input layer, hidden layers and an output layer. In any feed-forward neural network, any middle layers are called hidden because their inputs and outputs are masked by the activation function and final convolution. In a convolutional neural network, the hidden layers include layers that perform convolutions, and these layers include a layer that performs a dot product of the convolution kernel with the layer's input matrix. This product is usually the Frobenius inner product, and its activation function is commonly *ReLU*. As the convolution kernel slides along the input matrix for the layer, the convolution operation generates a feature map, which in turn contributes to the input of the next layer. This is followed by other layers such as pooling layers, fully connected layers, and normalization layers [10].

To obtain the dataset for training the robot, the author collects approximately 300 images captured from the OpenMV Cam: 100 for straight lines, right turns and left turns respectively. These images are pre-processed first using Canny operator to identify the edges and then using one-hot encoder for labelling. To further enlarge the dataset, the author applies data augmentation on the current dataset, increasing the sample base to a total number of 900 images by modifying fundamental parameters of each image to produce different images.

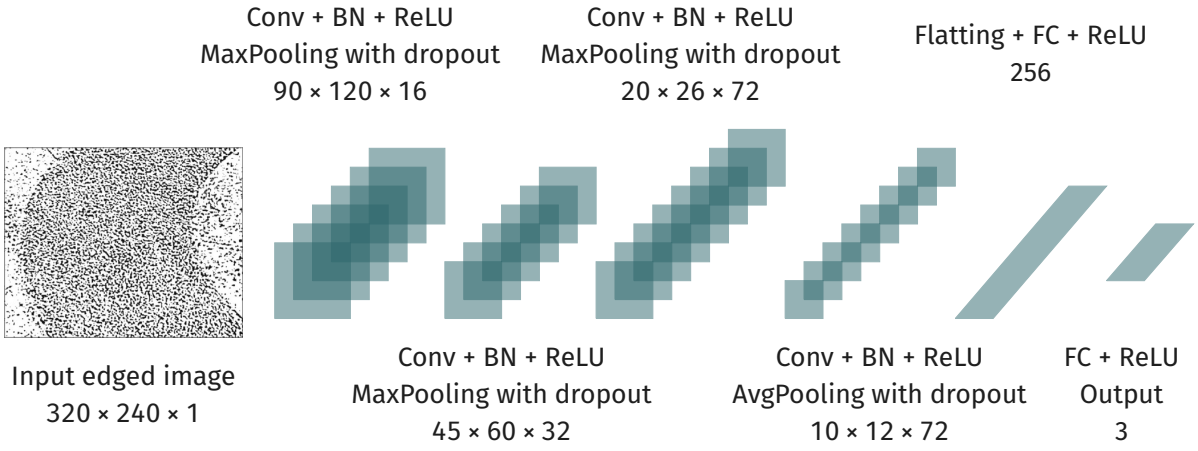


Figure 3.17: The architecture of convolutional neural netwrok (CNN) proposed and experimented by the team in initial phases of the project (abandoned for insufficient computing power)

With the dataset ready for training, the author utilizes the CNN architecture shown in Figure 3.17 to train the robot. In this architecture, convolutional layers (Conv) convolve the input and pass its result to the next layer. In these layers, the convolution operations are conducted in high detail, with various specified hyperparameters controlling the performance of the operations. One important rule regarding the convolution layers is that one layer's input channels must equal the number of output channels (also called depth) of its input [10]. The (batch) normalization (BN) layers mainly aim to expedite the training process by eliminating the Internal Covariate Shift – the change in the distributions of internal nodes of a deep network - through "a normalization step that fixes the means and variances of layer inputs" [11]. The process is also beneficial to the gradient flow through the deep network because it reduces the dependence of gradients on the scale of the parameters or of their initial values. Rectified Linear Units (ReLU) layers feature the commonly used activation function $ReLU(x) = \max[0, x]$, and allow faster and more effective training of deep neural architectures on large and complex datasets. Pooling layers functions mainly by reducing the dimensions of data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. The pooling methods used in this architecture are two of the most frequently used ones: MaxPooling which takes the maximum value of each local cluster of neurons in the feature map and Average Pooling (AvgPooling) which takes the general average value [10]. Fully connected (FC) layers connect every neuron in one layer to every neuron in another layer. The final output contains three vectors

that correspond to each possible case – turn left, turn right and go straight ahead.

Despite theoretical advantages of the DL-based solution, the training process overburdened the robot because its computing power is insufficient for such massive data processing and the experimental results were not as satisfactory as expected, so the team eventually abandons the solution and turns to another mathematical approach.

Multiple Centroid Algorithm The algorithm utilizes centroid coordinates of edge pixel clusters to calculate the angle at which the robot needs to turn to remain on course. One round of calculation is completed in the following steps:

1. Place *multiple rectangles* in the edge detected image to cover unoverlapped regions of the edged path;
2. Find the *centroid of white edge pixel clusters* in each rectangle using statistical techniques;
3. *Calculate, weight* and *sum* the horizontal distances of centroids to the vertical central line;
4. *Convert* the weighted centroid sum to an angle that allows the robot to veer along the path.

The sections where the rectangles are to be drawn is referred to as regions of interest (ROI), and they are preassigned in the initializing part of the program. The algorithm is visualized in Figure 3.18 to illustrate the process of calculation. Note that the angle calculation based on the deviation from the central line is nonlinear: the farther the deviation line falls, the faster the angle response increases to veer the robot back on course. This enables the robot to travel along the detected path, thus completing the task of line tracking.

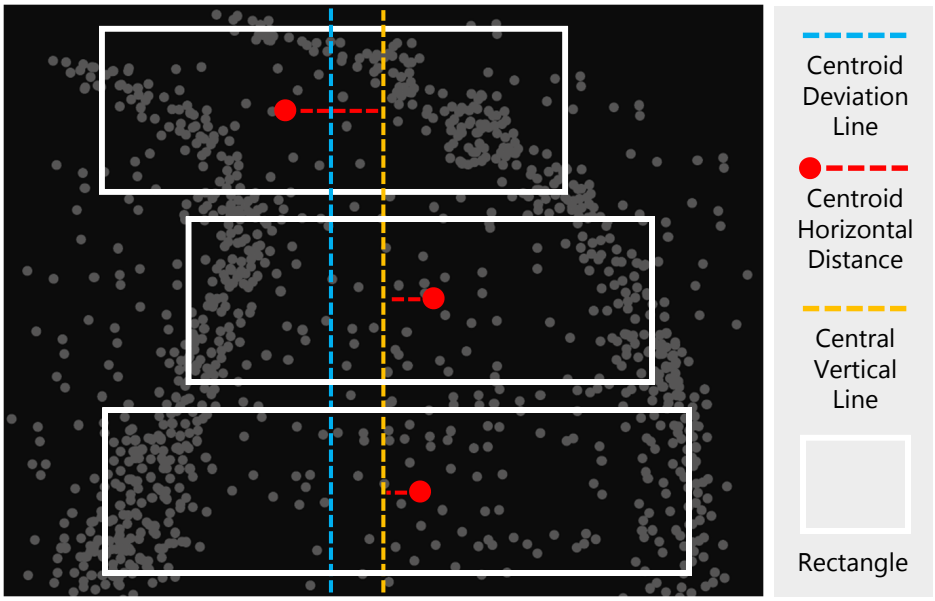


Figure 3.18: The visualized calculation of multiple centroid algorithm

3.3.3. Color Recognition [6]

The first task on Patio 2 requires the smart robot to recognize three different color chunks and travel along the corresponding preassigned paths. Therefore, the visual module should be programmed to distinguish these three different colors. In this section, the author explains the solution adapted by the team to materialize the function and discusses pertinent technical details.

[6] *Xie Shuangyang, UoG Matric: 2429274X, Contribution: Sections 3.3.3 - 3.3.4*

General Approach

Since the color chunks used in the final demonstration may differ from the standard colors, the reference thresholds used to support the robot's color recognition need to be found with the actual color chunk samples. This requires pictures containing the color chunks at the demonstration site be taken to obtain the actual color values. The easiest way is to manually extract the RGB values using simple color tools provided in Windows Office and convert them to CIELAB colors. The reference thresholds are chosen to cover all these converted sample colors to achieve accurate recognition.

While the robot is travelling forward, the visual module continues to receive and process image data. As the machine approaches the area of color recognition, the camera captures images and detect any colored sections in these samples. In the captured images, all pixels are scanned to identify whether they resemble any of the three target colors; the robot utilizes threshold segmentation to compare the inputted colors' parameters with their respective reference threshold values to conclude which one of the three preassigned colors it is. Technical details are provided in the following content.

RGB color space

The Red-Green-Blue (RGB) color model is an additive color model in which red, green, and blue light are added together in various ways to reproduce a broad array of colors. The name of the model comes from the initials of the three additive primary colors, red, green, and blue [12]. An RGB color space is any additive color space based on the RGB color model. A particular color space that employs RGB primaries for part of its specification is defined by the three chromaticities of the red, green, and blue additive primaries, and can produce any chromaticity that is within the 2D triangle defined by those primary colors [13].

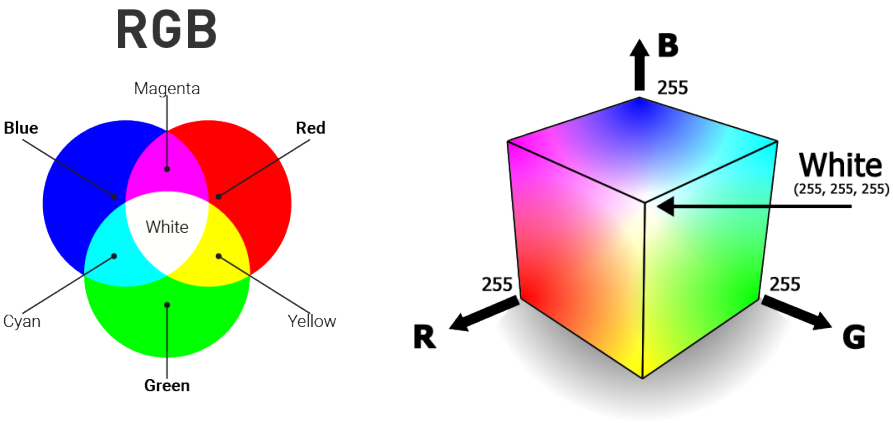


Figure 3.19: The 2- and 3-dimensional representations of RGB color space

To take advantage of the RGB color space in practice, it is important to master the numerical representations of RGB colors. A color in the RGB color model is described by indicating how much of each of the red, green, and blue is included. The color is expressed as an RGB triplet (R, G, B), each component of which can vary from zero to a defined maximum value. If all the components are at zero the result is black; if all are at maximum, the result is the brightest representable white. Several commonly used numerical representations are listed as follows:

- $R, G, B \in [0, 1]$: This representation is used in theoretical analyses, and in systems that use floating point representations.
- $R, G, B \in [0, 100\%]$: Each color component value can also be written as a percentage.
- $R, G, B \in [0, 255]$: In computers, the component values are often stored as unsigned integer numbers in the range 0 to 255, the range that a single 8-bit byte can offer.
- $(R, G, B) \in [\#000000, \#FFFFFF]$: This is an alternative to the computerized 0-to-255 representation. Colors are often represented as either decimal or hexadecimal numbers. [12]

In Figure 3.19, a fundamental RGB color model and an RGB color space mapped into a cube are displayed to illustrate the idea of'additive color synthesizing'.

CIELAB color space

The CIE 1931 color spaces are the first defined quantitative links between distributions of wavelengths in the electromagnetic visible spectrum, and physiologically perceived colors in human color vision [14]. From a series of color experiments, the CIE 1931 RGB color space was created, and this further derives the CIE 1931 XYZ color space, which formed the basis of the current CIELAB color space.

Compared to the RGB color space, the application of CIELAB color space has not been equally widespread, yet its functionality in computer vision should never be overlooked. CIELAB color space is a device-independent, "standard observer" model. Unlike RGB colors that are device-dependent, the colors defined by CIELAB are not relative to any particular device such as a computer monitor or a printer, but instead are aimed to emulate the perceptions via the human vision system. In other words, the CIELAB color space provides a numeric way to describe the colors as human eyes perceive them. Figure 3.20 shows the color spectrum of CIELAB, which is much larger than RGB color spaces.

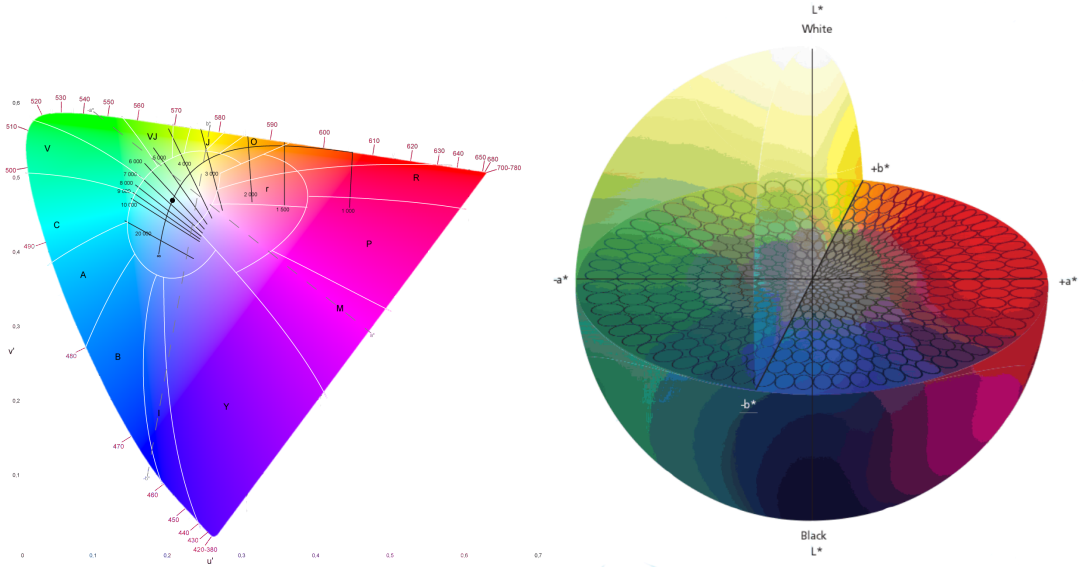


Figure 3.20: The spectral and spherical representation of CIELAB color space

One obvious distinction between RGB and CIELAB is their representation of lightness. In any one of the three channels of RGB, white and black represent the brightness of the color. For instance, when there is white or gray in any channel, the overall color cannot be black, because all three channels of R, G and B to form these colors altogether. CIELAB, on the flip side, separates the lightness parameter from the color channels and defines an individual channel to describe the overall brightness of the color.

CIELAB's distinctive lightness description better suits the demand of color detection in the current robot design. Since the robot should be able to operate in different situations where the lighting conditions are not uniform due to weather changes, color recognition free of lightness influence is crucial to the completion of tasks in Patio 2. With an individual parameter describing the overall image brightness, the team can apply a wide threshold interval to $L*$ to increase the tolerance for varied lightness, while applying a narrow threshold interval to color channels $a*$ and $b*$ to guarantee accurate color recognition.

RGB-to-CIELAB Conversion

Since the most widely used format of colors in computer image processing is RGB color model, color samples in the captured images should be converted to CIELAB format before they are inputted into the color recognition functions. The RGB colors cannot be directly translated to CIELAB colors: completing the conversion requires CIEXYZ as an intermediate between RGB and CIELAB. To implement the conversion, the mathematical theory behind it should be researched and studied.

The conversion from RGB to CIEXYZ is thoroughly explained at [15]. The process requires a reference white (X_w, Y_w, Z_w). For a given coordinate (x_R, y_R), (x_G, y_G) and (x_B, y_B) in the RGB system, its representation can be translated to CIEXYZ by operating a linear transformation

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = M \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix},$$

where M is the transform matrix. Different RGB color spaces correspond to different transform matrices. The general transform matrix has the form

$$M = \begin{bmatrix} S_R X_R & S_G X_G & S_B X_B \\ S_R Y_R & S_G Y_G & S_B Y_B \\ S_R Z_R & S_G Z_G & S_B Z_B \end{bmatrix},$$

Where all the parameters can be calculated by

$$\begin{bmatrix} S_R \\ S_G \\ S_B \end{bmatrix} = T^{-1} \cdot \begin{bmatrix} X_W \\ Y_W \\ Z_W \end{bmatrix}$$

and

$$T = \begin{bmatrix} X_R & X_G & X_B \\ Y_R & Y_G & Y_B \\ Z_R & Z_G & Z_B \end{bmatrix} = \begin{bmatrix} x_R/y_R & x_G/y_G & x_B/y_B \\ 1 & 1 & 1 \\ (1 - x_R - y_R)/y_R & (1 - x_G - y_G)/y_G & (1 - x_B - y_B)/y_B \end{bmatrix}.$$

In this project, the team uses the 8-bit sRGB scheme which gives the transform matrix

$$M = \begin{bmatrix} 0.4124564 & 0.3575761 & 0.1804375 \\ 0.2126729 & 0.7151522 & 0.0721750 \\ 0.0193339 & 0.1191920 & 0.9503041 \end{bmatrix}.$$

This way, the conversion formula becomes

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.4124564 & 0.3575761 & 0.1804375 \\ 0.2126729 & 0.7151522 & 0.0721750 \\ 0.0193339 & 0.1191920 & 0.9503041 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}.$$

After obtaining the CIEXYZ data, the color can be then translated to CIELAB by

$$\begin{bmatrix} L* \\ a* \\ b* \end{bmatrix} = \begin{bmatrix} 116 f_y - 16 \\ 500(f_x - f_y) \\ 200(f_y - f_z) \end{bmatrix},$$

where

$$f_x = \begin{cases} \sqrt[3]{x_R} & \text{if } x_R > \epsilon \\ \frac{\kappa x_R + 16}{116} & \text{otherwise} \end{cases}, \quad f_y = \begin{cases} \sqrt[3]{y_R} & \text{if } y_R > \epsilon \\ \frac{\kappa y_R + 16}{116} & \text{otherwise} \end{cases}, \quad f_z = \begin{cases} \sqrt[3]{z_R} & \text{if } z_R > \epsilon \\ \frac{\kappa z_R + 16}{116} & \text{otherwise} \end{cases},$$

and

$$x_R = X/X_R, \quad y_R = Y/Y_R, \quad z_R = Z/Z_R.$$

According to CIE standard, $\epsilon = 0.008856$ and $\kappa = 903.3$ [16]. After these calculations, an RGB color could be converted to CIELAB and be ready for color processing and recognition.

By referring to the online OpenMV MicroPython documentation at [8], the author finds that the **`image`** module provides readily-programmed functions that could achieve RGB-to-CIELAB conversion

and color detection with CIELAB colors. By directly importing the module and calling these functions defined in the module, the author could program the robot to recognize the preassigned color.

While visiting the patios, team members took numerous pictures of the sample color chunks. The author manually extracts the RGB values of color chunks in each picture using simple color extractor tools and translates the representations to CIELAB. In doing so, the team uses a function defined in the **image** module,

image.rgb_to_lab(rgb_tuple),

which inputs an RGB triplet and returns 3-value CIELAB tuple. This straightforward function achieves the conversion in one mere step. The results help the author find appropriate reference thresholds to accurately identify the target colors. Reference thresholds of one color should be written as

color = (minL, maxL, minA, maxA, minB, maxB),

to be inputted to the color recognition function.

With the CIELAB colors as references, another function is called to execute the color recognition:

image.find_blobs(thresholds, roi=Auto, x_stride=2, y_stride=1, invert=False, area_threshold=10, pixels_threshold=10, merge=False, margin=0, threshold_cb=None, merge_cb=None)

There are numerous parameters that controls the process, but in the case of color detection, only **thresholds** is needed to find all the colors in the captured colors. **thresholds** must be a list of tuples **[(lo, hi), (lo, hi), ..., (lo, hi)]** defining the ranges of color you want to identify. In this case, thresholds is directly assigned to be the target reference color. In the listing below, an exemplary program showcasing the use of the color detection are displayed, and this partially forms the basis of the main program used in the actual robot demonstration.

Listing 3.5: Color detection example

```
red = (Lr_min, Lr_max, Ar_min, Ar_max, Br_min, Br_max)
blue = (Lb_min, Lb_max, Ab_min, Ab_max, Bb_min, Bb_max)
yellow = (Ly_min, Ly_max, Ay_min, Ay_max, By_min, By_max)
# setting all measured LAB thresholds of different colors

img = sensor.snapshot()
# capturing the current image for color recognition

red_blobs = img.find_blobs([red])
blue_blobs = img.find_blobs([blue])
yellow_blobs = img.find_blobs([yellow])
# identifying one single color each time

color_blobs = img.find_blobs([red, blue, yellow])
# identifying multiple colors
```

3.3.4. AprilTag as Supportive Navigation

In the initial experiments, the team discovered that at some critical points, the smart robot cannot automatically escape a directional deviation even with visual recognition functions and algorithms: external guidance is needed to maintain the robot on track. Therefore, the team applies the AprilTag beacons to navigate the machine at these critical points.

AprilTag is a visual fiducial system, useful for a wide variety of tasks including augmented reality, robotics, and camera calibration. Targets can be created from an ordinary printer, and the AprilTag detection software computes the precise 3D position, orientation, and identity of the tags relative to the camera [17]. The AprilTag library is implemented in C with no external dependencies. It is designed to be easily included in other applications, as well as be portable to embedded devices. Real-time performance can be achieved even on cell-phone grade processors.

AprilTags are conceptually similar to QR Codes, in that they are a type of two-dimensional bar code. However, they are designed to encode far smaller data payloads (between 4 and 12 bits), allowing them to be detected more robustly and from longer ranges. Further, they are designed for

high localization accuracy – you can compute the precise 3D position of the AprilTag with respect to the camera. With these advantages and functionalities, AprilTags are considered suitable for supportive external navigation in this project. In this project design, the team adapts two readily made AprilTags as shown in Figure 3.21.

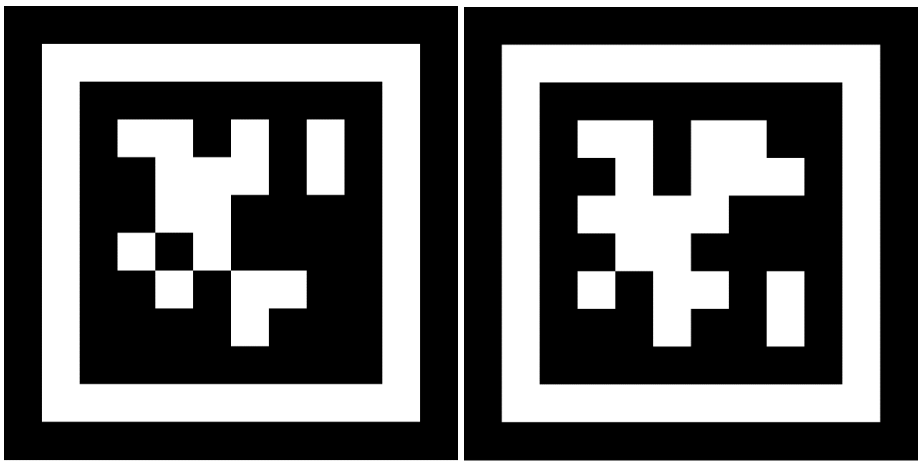


Figure 3.21: The AprilTags (TAG36H11 family) adapted by the team as external navigation source

The basic idea of supportive navigation in the context of AprilTag is that by recognizing AprilTags using OpenMV, the robot obtains the distance and relative angle from itself to the tags, which will help guide the robot to target destinations and perform corresponding tasks. Apparently, the robot should first identify any AprilTags in captured images before executing any further commands. The identification of tags is achieved in three main steps:

1. *Identify edges* in the captured image using Canny operator: other gradient-based edge detectors may be used as well, but Canny operator shows very excellent performance in noise reduction although it recognizes less edges than other operators.
2. *Recognize rectangular regions* in the edged image and determine whether the region contain encoded information, or QR code: we need to first apply topological structural analysis of digitized binary images, then find the convex hull of a simple polygon to sift out undesired edges, and use Douglas-Peucker algorithm to approximate the frames of the rectangles/squares [18].
3. *Decode QR codes* and *verify* the validity of the detected tag: the developer team provides a direct solution to this problem - **`self.tagfamily.decodeQuad(self,quads,gray)`** defined in **`tagFamilies`** pack.

Despite the complex theory of navigation using AprilTag beacons, OpenMV provides one-step solution to achieve the result. In OpenMV MicroPython **`image`** library, a class of commands (**`image.apriltag`**) is designed specially to maximize simplicity of AprilTag recognition. This feature is elucidated in **Programming Nuances**.

After the AprilTag is verified, the robot measures the relative direction and distance to the beacon, and these figures will serve as conditions to determine whether the robot has arrived at the preassigned positions.

In the robot race, there are two critical points that require external assistance for the robot: fish food release and wireless communication areas on Patio 2. Therefore, the team has used two readily made tags provided by the developer team, as displayed in Figure 3.21. Both tags are printed and attached to paper boxes so that they could stand on the ground. The practical use of AprilTag on Patio 2 is not focused in this section, but the reader could jump to **AprilTag Deployment** where the use of these beacons are explained.

3.4. Wireless Communication [7]

When the robot arrives at the area preassigned to commence wireless communication, the on-board microcontroller would send a message to the laptop terminal, which contains the team member list and current time information. Wireless communication is realized using the Wavesens HC-12 wireless transceiver, but the author discovers that the microcontroller used in the current design is not equipped with a built-in timer; additional modules are needed to provide time information. After sufficient research, the author decides to add an additional timekeeping module, a DS1302 Trickle Charge Timekeeping Chip, to provide accurate time information at the moment of wireless transmission. Connecting the HC-12 transceiver and DS1302 timekeeping module to the microcontroller enables the robot to complete the wireless communication task as required. Pertinent theories and technical details are introduced and briefly discussed in the following content.

3.4.1. Wireless Communication Theory

UART Communication

Asynchronous serial communication features and is distinguished from synchronous serial communication in that there are no extra clock signals needed to synchronize the transmitted data. The receiving terminal extracts all timing information directly from the signal, and this heightened demand on encoding and decoding target information necessitates more complex designs of transmitter and receiver nodes. In common situations, signals to transmit via the UART protocol should satisfy the following conditions:

- *Predetermined data rate:* both transmitter and receiver are preset to recognize the same data rate, so each node needs an accurate and stable clock source, from which the data rate can be generated;
- *Start & Stop bit:* each byte or word is framed with a Start and Stop bit. These allow synchronization to be initiated before the data starts to flow;
- *Use of parity bit:* the parity bit functions in error detection and is not compulsory in UART transmission, so both the transmitter and the receiver should agree on whether any parity bits are used.

An asynchronous serial port integrated into a microcontroller or peripheral device is generally called a UART, standing for universal asynchronous receiver/transmitter. In its simplest form, a UART has one connection for transmitted data, usually called TX, and another for received data, called RX. The port should sense when a start bit has been initiated, and automatically clock in and store the new word. It can initiate a transmission at any time. [19]

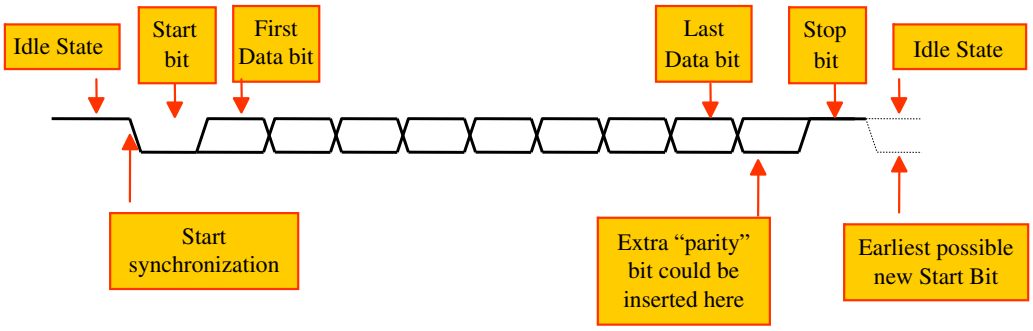


Figure 3.22: Commonly used UART data format (from [19])

Compared to other serial communication protocols such as SPI and I2C, UART communication prevails in many aspects: it does not rely on a separate clock signal to synchronize the data, it doubles the efficiency of bandwidth use because it is no longer necessary for the data signal to share bandwidth with an additional clock signal, and it is suitable for situations where a lower cost

[7] *Dai Yiting, UoG Matric: 2429286D, Contribution: Section 3.4*

of connection is desired. These features should be reflected to decide whether UART communication is felicitous for the target scene of application.

Several terminologies regarding UART communication are worth mentioning. Bits in the structure of data signals have different functions and they are mainly divided into Start bit (indicating the beginning of the data word), Data bit (containing actual useful data to transmit), Stop bit (indicating the end of the data word) and Parity bit (added for error detection). In an 8-bit data signal, a Start bit precedes the Data bits, situated in front of the sequence, and at the tail of the signal locates the Stop bit. No parity bit is present in this instance. In Figure 3.22, the structure of an 8-bit data word and the transmission sequence are illustrated.

The author also highlights the distinction between baud rate and throughput. Baud rate refers to the bit rate of the serial port, that is the total data rate of transmission, while throughput refers to the bit rate of the actual useful data transmitted per second. Apparently, throughput is lower than the baud rate.

3.4.2. Wireless Channel

The UART protocol is still valid when the physical wires for data transmission are substituted with a pair of wireless data transmission devices; as long as the 'asynchronous nature' of encoding and extracting data without using separate clock signals remains unchanged, adapting a different medium on which the target information is transmitted does not undermine the plausibility of the UART protocol. To achieve wireless data transmission, a pair of hardware components are deployed to serve as transmitter/sender and receiver, and in this case the team has chosen two Wavesens HC-12 wireless transceivers to establish the wireless communication channel. Figure 3.23 shows the substitution of data transmission medium.

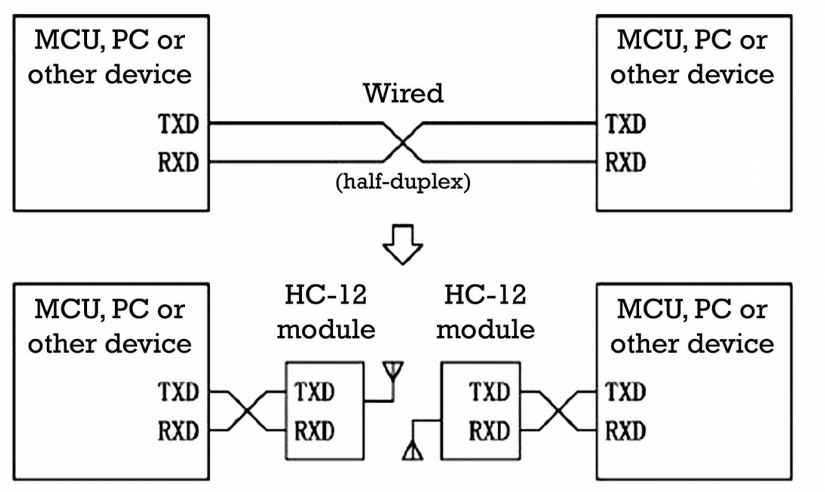


Figure 3.23: The medium on which UART data propagates changes to wireless channel (from [20])

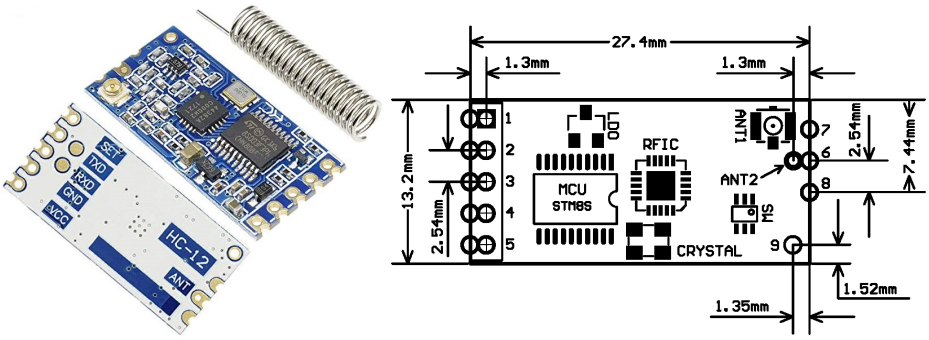
3.4.3. HC-12 Transceiver

The HC-12 is a half-duplex 20 dBm (100 mW) transmitter paired with a receiver that has -117 dBm ($2 \times 10^{-15} W$) sensitivity at 5000 bps. Paired with an external antenna, these transceivers are capable of communicating up to and possibly slightly beyond 1km in the open and are more than adequate for providing coverage throughout a typical house [21]. The author, acknowledging the aforementioned features of the component, reckons that the transceiver is felicitous for implementing wireless message transmission from the robot to the laptop terminal. The component's external appearance and schematic diagram is shown in Figure 3.24.

As is discussed in previous content, the wireless transmission requires the configuration sketched in Figure 3.23 [20]. In this case, two transceivers are required to guarantee a fully functional wireless communication between the robot (sender) and laptop (receiver): the one connected to the on-board microcontroller sends the target information, while the other plugged in the laptop receives the

transmitted message. Since the transceiver is not designed with USB sockets allowing direct connection to personal computers, an additional USB-to-TTL converter is needed to enable the connection at the receiving terminal. After installing the antenna on each transceiver, the wireless channel is established and ready for practical data transmission.

As shown in Figure 3.23, two HC-12 modules can be used in place of physical wiring to replace a wired half-duplex serial communications link carrying TTL-level signals. The left device sends serial port data to the module, and after the RxD port of the left module receives the serial port data, it will automatically send the data over the air via radio wave. The right module receives the data, and restores the serial port data originally sent by the left device and sends it out TxD. The process is the same in both directions. Only a half-duplex link is available between modules, as they cannot receive and send data over the air at the same time.



(a) Actual external appearance **(b)** Schematic Diagram

Figure 3.24: The HC-12 transceiver module and its schematic diagram

According to the datasheet [20], the HC-12 module has four serial port transparent transmission modes, expressed as *FU1, FU2, FU3,* and *FU4*. In operation, these modes hide all the details of wireless communications from attached devices. The modules are usually operated in pairs, with data transmitted by means of a half-duplex link. For successful wireless transmission, the transparent transmission mode, serial port baud rate, and wireless communication channel of the two paired modules must be identical. The factory default module settings are: FU3, 9,600bps (8N1: 8 data bits, no parity, 1 stop bit), CH001 (433.4MHz), 20dBm power (100mW).

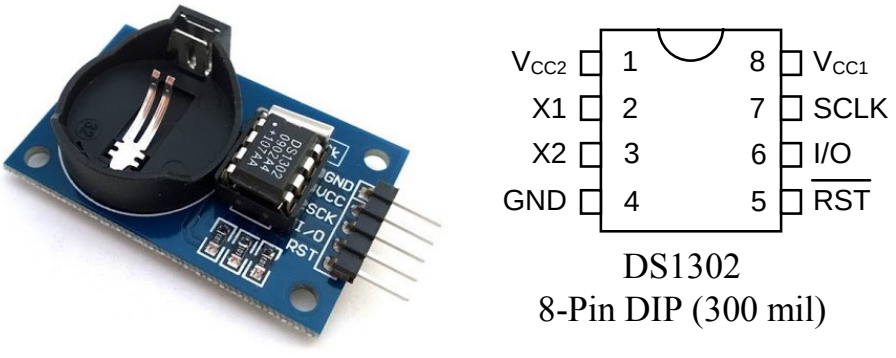
While programming the wireless communication, the team set the baud rate to 9600 bits/second, data bits to 8 bits, no parity and one stop bit for the process. The frequency channel to which our radio signal communication is assigned remains default – at around 433MHz.

The team utilize a serial port tester tool to operate and monitor the wireless communication. After identifying and matching the serial ports that the receiving terminal uses in the software, the main program executing the wireless communication is downloaded to the microcontroller, initiating the data transmission. Notice that the accurate time information is not generated by the on-board microcontroller itself, but by the DS1302 module.

3.4.4. DS1302 Timekeeping Chip

The DS1302 trickle-charge timekeeping chip contains a real-time clock/calendar and 31 bytes of static RAM. It communicates with a microprocessor via a simple serial interface. The real-time clock/calendar provides seconds, minutes, hours, day, date, month, and year information. Only three wires are required to communicate with the clock/RAM: CE, I/O (data line), and SCLK (serial clock). Data can be transferred to and from the clock/RAM 1 byte at a time or in a burst of up to 31 bytes. The DS1302 is designed to operate on very low power and retain data and clock information on less than 1µW. The external appearance and pin configuration of the product are displayed in Figure 3.25. [22][23]

In this design, the module is used to provide accurate time information to the microcontroller which would transmit the information to the laptop terminal. Having perused the user manual and experimented with the timekeeping module, the author finds that once the module is initialized, the microcontroller to which the timekeeping module is connected could still acquire time information continuously from the module even without powering it, because the module is integrated with built-



(a) Actual external appearance **(b)** Schematic Diagram

Figure 3.25: The DS1302 timekeeping module and its schematic diagram

in batteries. This way, whenever the microcontroller switches on to start functioning, it receives current time information from the DS1302 module.

In order to simplify and facilitate the impending system integration, the whole wireless communication process is encapsulated as a self-defined function and header file, so that the process could be called and executed with ease in the central control program.

3.4.5. Programming Overview

In a nutshell, the wireless communication module is programmed to complete the required tasks in three steps, and the program design follows this structure:

1. *Program* the DS1302 timekeeping module to enable time data output;
2. *Encapsulate* the important functions of the time-monitoring program so that they could be used in other programs without redefining everything from scratch;
3. *Program* the HC-12 module so that it takes time data from DS1302 and establishes communication with the laptop terminal.

In the program of DS1302 per se, several functions are defined to make the module fully functional. These functions, on the other hand, are not all necessary for the completion of this project: only crucial ones should be highlighted here. To start the clock for the first time, as is mentioned earlier, the module requires an "initializer" function, and future starting will no longer require such a procedure. The starting of the clock module is commanded by a function defined as shown below. Note that the first line is the "initializer" function and should be deleted once the first initialization is completed.

Listing 3.6: The "start" and "initializer" function

```
void DS1302_SubMain(){

    settime();
    // this line is used for the first initialization,
    // delete the first line after first initialization
    readtime();
}
```

The address-reading function **`cmd_readbyte`** is of paramount importance to monitoring the time throughout a long period of time. The function is defined in the following way:

Listing 3.7: Address-reading cmd_readbyte function

```
uint8_t cmd_readbyte(uint8_t add){  // read commands
    uint8_t i, value;
    ce = 0;
    wait_us(20);
```

```
    ce = 1;
    wait_us(20);

    sclk = 0;
    wait_us(20);

    // write an address of one byte
    io.output();
    for(i=0;i<8;i++){
        value=add&0x01;
        if(value)
            io = 1;
        else
            io = 0;
        add>>=1;
        sclk = 1;
        wait_us(20);
        if(i<7){
            sclk = 0;
            wait_us(20);
        }
    }

    // read one byte
    io.mode(PullUp);
    io.input();
    value = 0;
    wait_us(20);
    for(i=0; i<8; i++){
        sclk = 0;

        value>>=1;
        if(io.read() == 1)
            value|=0x80;
        sclk = 1;
        wait_us(20);
    }

    ce = 0;
    wait_us(20);
    sclk = 0;
    wait_us(20);

    return value;
}
```

By reading the addresses given by the hardware developers, the module realizes the timekeeping function. Reading the time information and categorizing the data into second, minute, hour, week, day, month and year are achieved by the **readtime** function, and its definition is illustrated as follows.

Listing 3.8: The acquisition of time information in categories

```
uint8_t tim[7]={0x30, 0x31, 0x16, 0x28, 0x05, 0x05, 0x21};
// second, minute, hour, day, month, week, year
uint8_t w[7]={0x80, 0x82, 0x84, 0x86, 0x88, 0x8a, 0x8c};
// writes the address bytes
uint8_t r[7]={0x81, 0x83, 0x85, 0x87, 0x89, 0x8b, 0x8d};
// reads address bytes
uint8_t second,minute,hour,week,day,month,year;
// defines categories of time information to carry and provide

void readtime()
{
    second=cmd_readbyte(r[0]);
    minute=cmd_readbyte(r[1]);
    hour=cmd_readbyte(r[2]);
    week=cmd_readbyte(r[5]);
    day=cmd_readbyte(r[3]);
    month=cmd_readbyte(r[4]);
    year=cmd_readbyte(r[6]);
} // reads all the time information from the pregiven address strings
```

In order to expand the use of categorized time information, the author writes a header file as the "bridge" to connect the DS1302 program with other external programs. In this case, sure enough, the module needs to be started in the central control program and all categories of time information should be extracted to be further used in wireless message transmission. Therefore, the header file is written as follows.

Listing 3.9: DS1302 header file

```
#ifndef DS1302_H
#define DS1302_H

#include "mbed.h"

extern uint8_t second,minute,hour,week,day,month,year;
void readtime();
void DS1302_SubMain();

#endif
```

By including the header file in the central control program, all categories of time information can be directly accessed, and functions **`readtime`** and **`DS1302_SubMain`** can be directly executed.

3.5. Patio 1 System Integration

Now that the team has developed and tested all the subsystems and proven that each subsystem functions normally when connected to the microcontroller, team members start to assemble the subsystem hardware and merge the sub-programs that execute individual tasks as listed in the previous chapter. Since two patios are not geographically adjoint and each patio requires the robot to accomplish a different set of tasks, the central control program would be divided into two sections, each executing the tasks required in one specific patio. This requires reprogramming the microcontrollers after the robot transits from Patio 1 to Patio 2. In the following content, the author explains the team's overall integration approach to implement the smart robot.

3.5.1. Integration Overview [8]

One of the primary concerns in system integration is to maximize the reusability of subsystem programs elements. After reviewing and comparing the subsystem programs, the author identifies and summarizes the source code fragments that are repeatedly used in multiple sub-programs throughout both two patios. These fragments are extracted and encapsulated as functions or header files with a uniform input, output format that allows these functional fragments to be directly plugged and executed in the other contexts; the programmer only need to transplant the readily programmed functions instead of reimplementing them from null. These transplantable functional fragments mainly include:

- UART communication between the mbed microcontroller (STM32L432KC controller board) and the OpenMV Cam,
- Directional control where the PID algorithm commands four PWM output pins to generate velocity differences on two tracks to veer the robot,
- Distance calculation via the ultrasonic sensor (SRF05 ultrasonic ranger),
- Angle detection via the angle sensor (MPU6050 accelerometer and gyroscope).

These processes are potential of multiple use in many of the target tasks on both patios, and encapsulating them has made it much more convenient to reuse these functions. The top-view task diagram is depicted in Figure 3.26.

Another concern pertains to the deficiency of UART communication sockets. While the robot design requires at least three sockets to establish communication channels with hardware peripherals and control the driving system, there is one mere socket available on the mbed microcontroller. Therefore, all necessary UART communications have to share one single socket, occupying the channel in different time periods. This time-sharing use of communication channel could be realized using multiplexers/switches. The team has chosen the CD4052 multiplexer to implement the design.

[8] *Jin Jing, UoG Matric: 2429327J, Contribution: Section 3.5.1*

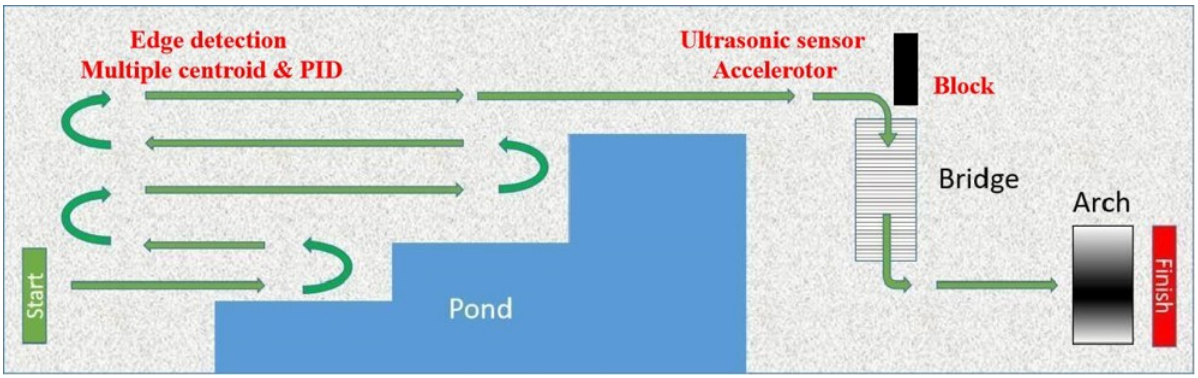


Figure 3.26: The task flow diagram of Patio 1

Aware of the possibility that the robot experiences unexpected issues and is unable to proceed with the remaining tasks, the author proposes that the robot be programmed to restart from the current task instead of the first task of the patio. Therefore, the team adds an OLED screen and a touch pad detector (TTP223 1-key touch pad detector) so that the team members could choose the specific task to restart from when the robot accidentally fails to complete one task.

The general integration of the subsystems requires arranging the structure of the main program such that the functions and commands are called and executed in the order required in the task briefing. After the subsystem codes have been merged as a main program of Patio 1 tasks, however, an immediate problem emerges that the OLED screen obstructs the PWM output pins from normal functioning. A presumed explanation for this conflict is that both PWM output signals and IIC signals used by the OLED screen share the same clock source, so the microcontroller can only generate either category of signals at a time. Besides, the mbed microcontroller has insufficient pins to connect all peripherals, so the team needs to rearrange the circuit to disconnect unused modules before switching to another patio. If any human errors were to occur in rearranging the circuit, the robot would possibly fail the tasks or even crash in the impending patio. Another point to make is that the general decision-making mechanism is complicated and this may lead to many issues that undermine the performance the smart robot.

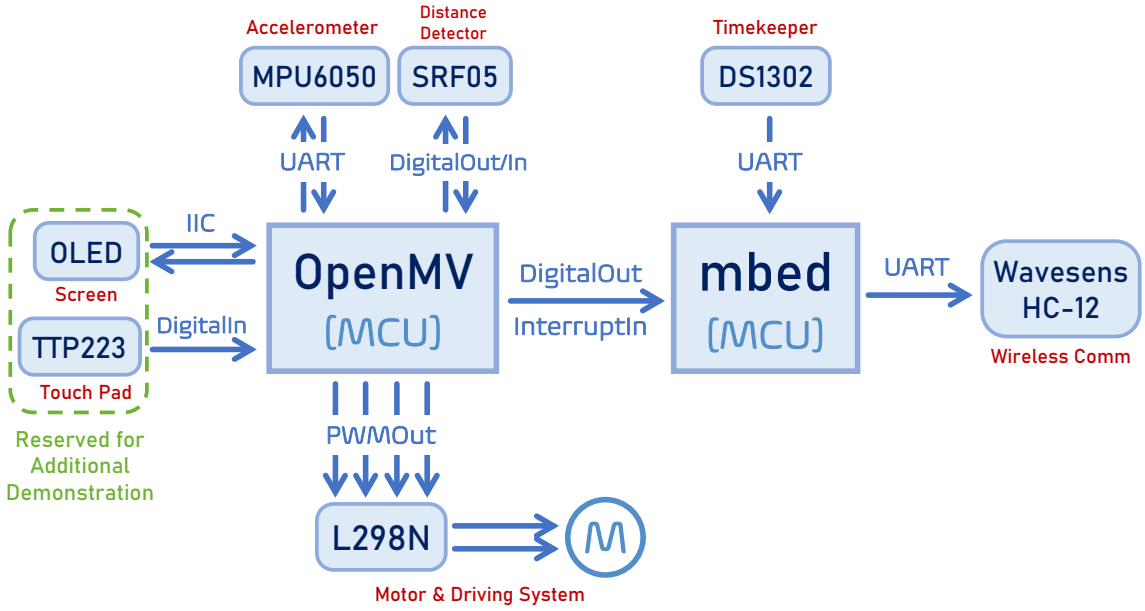


Figure 3.27: The final hardware layout and peripheral connection

To handle the problems, the author suggests the team move the PWM output pins, reconnect the ultrasonic and angle sensor to the built-in microcontroller on OpenMV Cam, and program the direction control algorithm to the visual module as well. This modification requires the team to translate the original embedded C program into a microPython program. In the meantime, as the OLED and the touch pad has taken all the remaining ports on the OpenMV Cam, the team removes these components from the system design to leave several unoccupied ports for emergency use. If the final demonstration proceeds as expected, the team would consider showcasing the use of the OLED and touch pad. The final hardware layout diagram is shown in Figure 3.27.

3.5.2. Distance Detection [9]

Besides visual recognition functions provided by the OpenMV Cam, the robot's distance between the surrounding objects is also of great importance in locating and navigating the robot along the pre-assigned paths. After sufficient research, the team concentrates on two distance measuring sensors that measure the vertical distance between the robot and the objects/barriers in front of it – infrared sensor (GP2Y0A02YK0F) and ultrasonic sensor (HY-SRF05). Figure 3.28 show the external structure of both sensors.

Figure 3.28: The infrared and ultrasonic distance measuring sensors

Their working principles are similar: physical probe signals are dissipated from the devices and reflected at objects/barriers to return to the sending terminal, and the time duration of a sent signal returning to its sender helps calculate the distance between the object/barrier that has bounced the signal back. Their major distinction, on the other hand, is the nature of the probe signal used in distance detection: as their respective names suggest, one utilizes infrared waves while the other utilizes ultrasonic waves.

According to the datasheets, both devices are technically suitable for the robot design. Considering the fact that the demonstration is commenced outdoors, however, the team decides to use the SRF05 ultrasonic sensor instead of the infrared sensor, because the outdoor sunshine, if weather conditions prevail, contains high-level infrared rays and these wave components will mix with the probe signals and obstruct the distance detection functions. Therefore, the team proceeds with the ultrasonic sensor to avoid such unwanted issues.

The working process of the ultrasonic sensor is straightforward, as displayed in Figure 3.29. To start with, the trigger signal channel outputs a 10-μs TTL high-level signal to initiate the whole process. Next, the module sends 8 square pulses in consecution at the frequency of 40Hz, and these pulses serve as probe signals in this case. The echo channel outputs high-level signals when the reflected signals are being received, and the width of the pulse signal is proportional to the distance between the device and the detected object. The calculation of the detected distance is given by

$$d = \frac{T_h v_s}{2},$$

where d is the measured distance, T_h is the width of the high-level signal, or the time during which reflected signals are received, and v_s is the speed of sound waves (often estimated 340m/s).

[9] *Liang Xuanyu, UoG Matric: 2429266L, Contribution: Section 3.5.2*

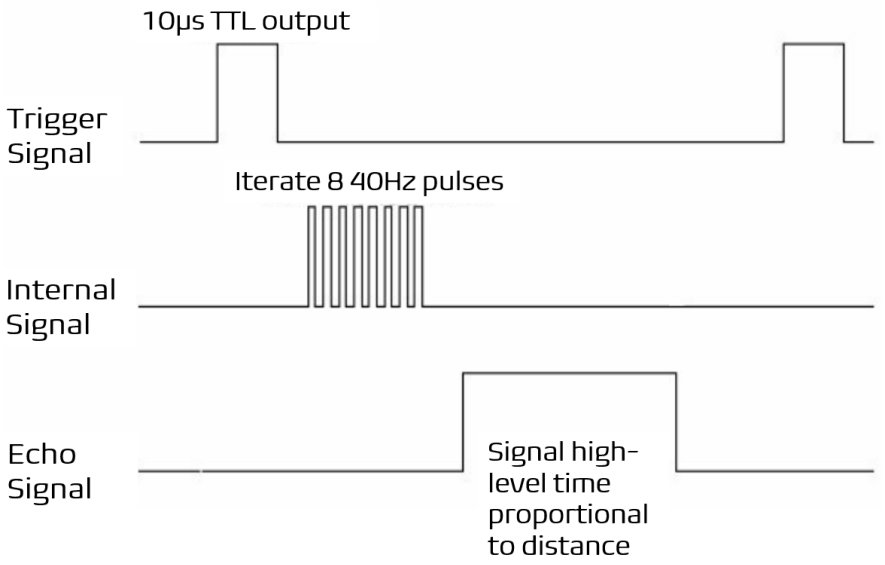


Figure 3.29: The working process of HY-SRF05 ultrasonic sensor (from [24])

After the calculation, the module returns the distance data to the OpenMV Cam's built-in microcontroller. The central control program will run through decision making processes with the updated distance data. The distance detection iterates throughout the entire moving process.

3.5.3. Heading Detection [4]

On Patio 1, the smart robot is required to perform steep turns after travelling along the preassigned paths for some time. During a sharp turn, the quantified direction changes are of vital importance to determine whether the turn has been completed. There are two approaches to obtain the directional data: comparing the absolute magnetic field readings, and directly finding the angular differences. To achieve each goal, a different device is needed.

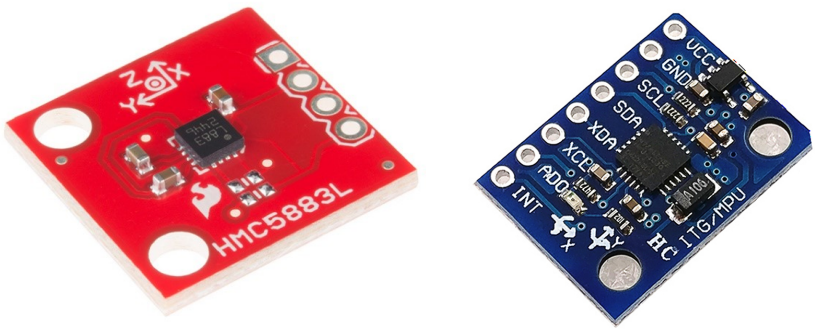


Figure 3.30: The HMC5883L magnetometer and MPU6050 accelerometer

In the initial proposal, the team inclined to use a digital compass device to track the magnetic field directions of the robot. The readings are transmitted back to the microcontroller and used for decision making. The model chosen for the initial design was HMC5883L magnetometer module, as shown in Figure 3.28 (left). By calibrating the module using the methods suggested in [25], the author programmed and tested the validity of the module. However, the measured magnetic headings are checked to be in poor accordance with the actual magnetic field readings, and the reason had

[4] *Liu Yilin, UoG Matric: 2429448L, Contribution: Section 3.2, 3.5.3 - 3.5.6*

remained unknown after numerous experiments. Therefore, the team abandoned the module and turned to directly measure the angular differences.

The new approach requires another module – MPU6050 accelerometer and gyro sensor. The moudle is equipped with a voltage-stablizing circuit, which is compatible to 3.3V and 5V embedded systems. Advanced digital filtering technology effectively reduces the noise and increases the accuracy of the measurements. More importantly, it is programmed with excellent temperature compensation to suppress the influence brought about by temperature changes.

The accelerometer measures four sets of data in three-dimensional space (with x, y, z axis): acceleration, angular velocity, angle (heading) and temperature. The author has recorded the data from one successful experiment and depicted the measured quantities in Figure 3.31.

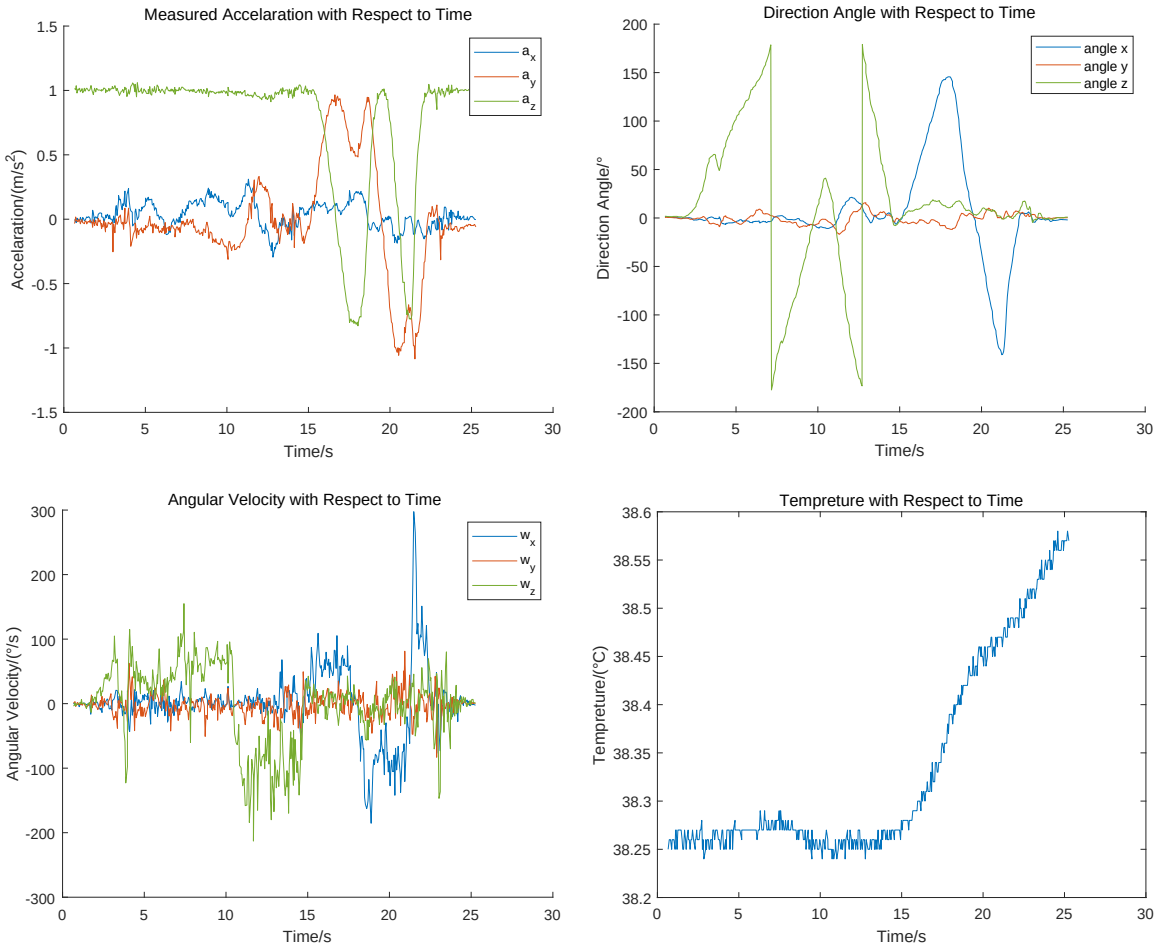


Figure 3.31: The four quantities measured by MPU6050 in an experiment

The data transmission between the module and the computer is executed via UART communication on TTL-USB wire connections. The communication is bidirectional and, in each direction, the information transmitted is different: the microcontroller sends commands to the accelerometer, such as initializing the angle data readings, while the module sends back three data packs – acceleration, angle and angular velocity. Each data pack comprises ten data, and the transmission follows a given order. The baud rate of the UART transmission is set to 9600, and transmits one frame of data in every 50ms.

Since the team only needs the angle data to control the turning process, the robot is programed to only process the data pack starting with *0x55, 0x53*; although other three categories of measured data are also transmitted to the microcontroller via UART communication, they are discarded or neglected by the processor. The complete data encapsulation is shown in Table 3.2. The angle data can be directly retrieved from the data pack.

Data Number	Data Content/Variable Name	Functionality
0	0x55	Header data
1	0x53	Marking the data pack as 'angle data'
2	RollL	*x*-axis angle lower byte
3	RollH	*x*-axis angle higher byte
4	PitchL	*y*-axis angle lower byte
5	PitchH	*y*-axis angle higher byte
6	YawL	*z*-axis angle lower byte
7	YawH	*z*-axis angle higher byte
8	TL	Temperature lower byte
9	TH	Temperature higher byte
10	Sum	Checksum

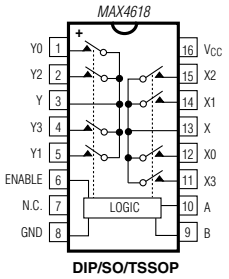
Table 3.2: The structure of the data pack containing measured angle data

Note that the checksum is used to verify the validity of the data, calculated by

$$Sum = 0x55 + 0x53 + RollL + RollH + PitchL + PitchH + YawL + YawH + TL + TH.$$

3.5.4. Multiplexer/Switch

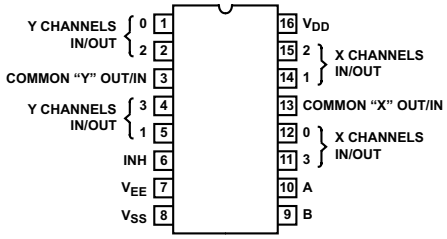
Initially, the team utilized the MAX4618 module to materialize the multiplexing and demultiplexing functions required by the current circuit design. The component's pin configuration and truth table are displayed below.



ENABLE	B	A	State
H	X	X	All switches open
L	L	L	X-X0, Y-Y0
L	L	H	X-X1, Y-Y1
L	H	L	X-X2, Y-Y2
L	H	H	X-X3, Y-Y3

Figure 3.32 & Table 3.3: MAX4618 top-view pin configuration and truth table (from [26])

In the experiments, the author tested the viability of multi-route switching with 3.3V square pulse signals, but was frustrated to find that only the routes X-X0/X1/X2/X3 are valid, but the same routes are not valid in opposite direction. Since UART communication requires bidirectional data transmission, the component cannot be used because it only supports unidirectional transmission.



ENABLE	B	A	State
1	X	X	None
0	0	0	X0, Y0
0	0	1	X1, Y1
0	1	0	X2, Y2
0	1	1	X3, Y3

Figure 3.33 & Table 3.4: CD4052(B) top-view pin configuration and truth table (from [27])

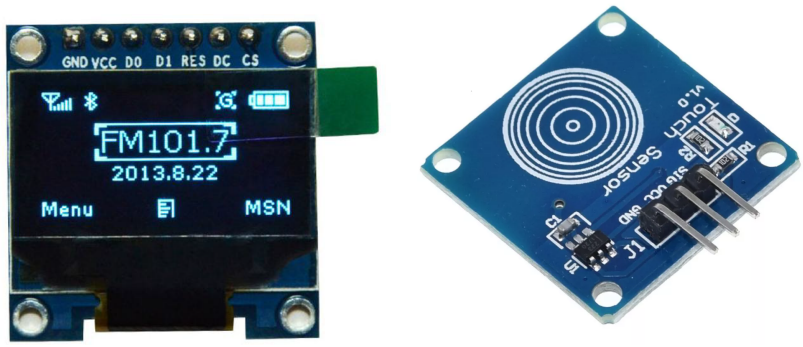
Inspired by the use of the CD4052 component as an expander to handle the insufficiency of UART sockets in [28], the author turned to experiment with this multiplexer/switch model. After several

trails, the module is proven to be suitable for bi-directional communication required by UART protocol, and switching the routes at different times. Therefore, the team proceeds with CD4052. The component's pin configuration and truth table are shown below.

3.5.5. Program Execution Monitoring Module

In the initial design, the team anticipated to add a visual program monitor that displays the progress of the main robot program and helps to restart the machine in the case of a midway system crash, as well as an interactive device (touch pad) that allows the operator to restart the machine from several programmed critical points. This eliminates the uncertainty brought about by restarting from the beginning and also saves much time.

The design adapted an OLED (Organic Light Emitting Diode) screen to display the current status of the smart robot. The task that is being executed by the robot would be displayed, and if the system crashes and needs to reboot, the screen would display several options – critical points from which the system is to restart – for the human operator to choose. To confirm a decision, the operator needs to tap on TTP223 touch pad to transmit an input signal to the robot. This interactive feature brings much convenience to the operator when handling unexpected crashes, but it has also increased the programming complexity.



(a) The OLED display screen **(b)** The TTP223 touch pad

Figure 3.34: The components of robot program execution monitoring module

To accurately detect the human interaction, the debounce feature has been added to the program of the touch pad, as shown in the listing below. This aims to decrease the susceptibility of the module and react only to the human touch. Since high-level input value indicates a possible touch, the program regulates that the input is recognized as a human touch only if the input value remains at high level after a given period of time, i.e. half of the debounce constant. The debounce feature can be realized with the function as follows.

Listing 3.10: Debounce function

```
void ssd_start(void){
    ssd_init(); // initializes the debounce feature
    t.start(); // initializes the clock
    ssd_tasknum();
    while(1){
      if(In.read()==1){
          wait(debounce/2); // waits for debounce time
          if(In.read()==1) {
              wait(debounce/2); // waits again for debounce time
              if(In.read()==1) tasknum+=1;
              ssd_tasknum(); // re-enters another round of debounce
          }
      }
    if(t.read()>=waittime){t.stop();t.reset(); break;} // confirms human input
    }
}
```

In later phases of the project, the team has encountered several unknown clashes between the wiring of the monitor module and other components, so the team has decided to disconnect the module from the main program and reserve it for extra demonstration if the robot manages to complete the race; the team will showcase this monitor module if all tasks are completed and time permits.

3.5.6. UART Communication between OpenMV and mbed microcontroller

The initial design scheme distributes the total workload evenly to OpenMV and mbed microcontroller, and this necessitates the bidirectional data transmission using UART protocol. The data transmission mainly aims to promptly update the angle data and navigation commands, which will be inputted to the direction control algorithm to eliminate angular deviation. The data transmitted via this connection has four components, as is shown in Table 3.5.

Header	Angle (Integer)	Angle (Decimal)	Command
0xFF	0xXX	0xXX	0xXX

Table 3.5: UART communication data pack

The transmission data consists of four sections: header, angle (integer), angle (decimal), command. This package is referred to as a datagram, inspired by the data structure of UDP protocol, where the header (0xFF) is used to locate the data to determine whether there has been a delay, lost data byte or disordered data sequence. Each section contains one byte data, denoted by a hexadecimal number, which can represent any integer between -128 and 127. The direction control algorithm outputs an angle represented by an integer and two decimal digits, so the data is divided into two sections in the datagram. The function materializing the UART communication is listed as follows.

Listing 3.11: Serial communication

```
void serialcom(void){
    int count=0; // initialize the counter
    while(true){
        openmv_buf[count]=com.getc();// receive data from UART
        if(count==0&&openmv_buf[0]!=0xff) continue;
        // determine whether the first data is correct
        count++;
        if(count==4){
            if (openmv_buf[1]>127){openmv_angle=openmv_buf[1]-256;}
            // determine whether the data is negative
            else{openmv_angle=openmv_buf[1];}

            if (openmv_buf[2]>127){openmv_angle+=(openmv_buf[2]-256)*0.01;}
            //determine whether the data is negative
            else{openmv_angle+=openmv_buf[2]*0.01;}

            openmv_order=int(openmv_buf[3]);
            break;
            // endless loop until getting all 4 bytes of data
        }
    }
    return;
}
```

At the rear of the datagram, the command data is used to inform the mbed microcontroller which maneuver to perform at the current moment. The commands defined in the system program include

- *0x01:* line tracking along detected path;
- *0x02:* brake and halt the robot immediately;
- *0x03:* turn to a specific direction;
- *0x04:* cross the bridge.

In later experiments, the decision-making mechanism in this design scheme is considered unnecessarily complicated, and is likely to decrease the frame rate of the OpenMV camera which could

engender inaccurate or erroneous path detection and color recognition. Therefore, the team simplifies the program architecture by transplanting most of the decision-making processes and algorithms to one microcontroller, the other concentrating on specified functions. Eventually, the team leaves the wireless communication on the mbed microcontroller and transplants the remaining parts to the OpenMV Cam.

In this final design, the communication between the control processors is simplified to one single digital signal wire, so the UART communication program is no longer needed here. This section is written to exhibit the thoughts of developing the channel for inter-module data transmission.

3.6. Patio 2 System Integration [10]

Likewise, the author has to review all subsystem design solutions to present a plan that accommodates every module and commands the robot to complete the required tasks one by one. Since all of the necessary components of Patio 2 system integration has been adequately explained and discussed, the author only needs to focus on the arrangement of modules and functions of the smart robot.

3.6.1. Solution Overview

Initial approach

In the team's initial approach, color detection, navigation to color chunks, navigation to feed area and navigation to communication area are executed collectively by both the OpenMV Cam module and mbed microcontroller. In this scenario, OpenMV is responsible for

1. *recognizing* colors that guide the robot to perform different maneuvers,
2. *locating* the AprilTag beacons that assist the robot's navigation,

while the mbed microcontroller is responsible for

1. *generating* PWM output signals to power the motors and tracks,
2. *executing* PID control algorithm (with data input from the MPU6050 accelerometer),
3. *conducting* wireless communication (with HC-12 transceivers and DS1302 timekeeping chip).

Once the OpenMV captures the AprilTag beacon within its sight, it calculates the distance and relative direction of the beacon from the robot's current location, and transmits the data back to the mbed controller to which it is connected. The microcontroller, upon receiving the data, calls and executes the direction control (PID) algorithm to adjust the PWM output values for the motors to veer the robot to the correct path. The entire process of Patio 2 task execution is shown in Figure 3.35.

Limitations

As is mentioned in the team's integration approach in Patio 1, the team understands the difficulty that the individual mbed microcontroller has only one UART socket which has been used to output PWM signals to the motors, but the system design requires two additional UART sockets be available to communicate with the OpenMV Cam and the wireless communication module, and the solution is to introduce an analog switch/multiplexer to the circuit to select the path on which the PWM output/UART communication is to commence at appropriate times. The subsequent problem emerges that unknown clashes between the routes passing through the 1-to-3 switch/3-1 demultiplexer are considered irresolvable. Besides, since almost every pin on the mbed controller is assigned to receive input signals, generate output signals and UART communication, the microcontroller tends to overburden throughout the process, and the transition between two patios demand that the circuit arrangement be manually modified because each patio concentrates on different tasks which rely on different hardware modules, but the deficiency of external sockets precludes every peripheral to connect to one specified and fixed socket.

[10] *Liu Shiyuan, UoG Matric: 2429438L, Contribution: Section 3.6*

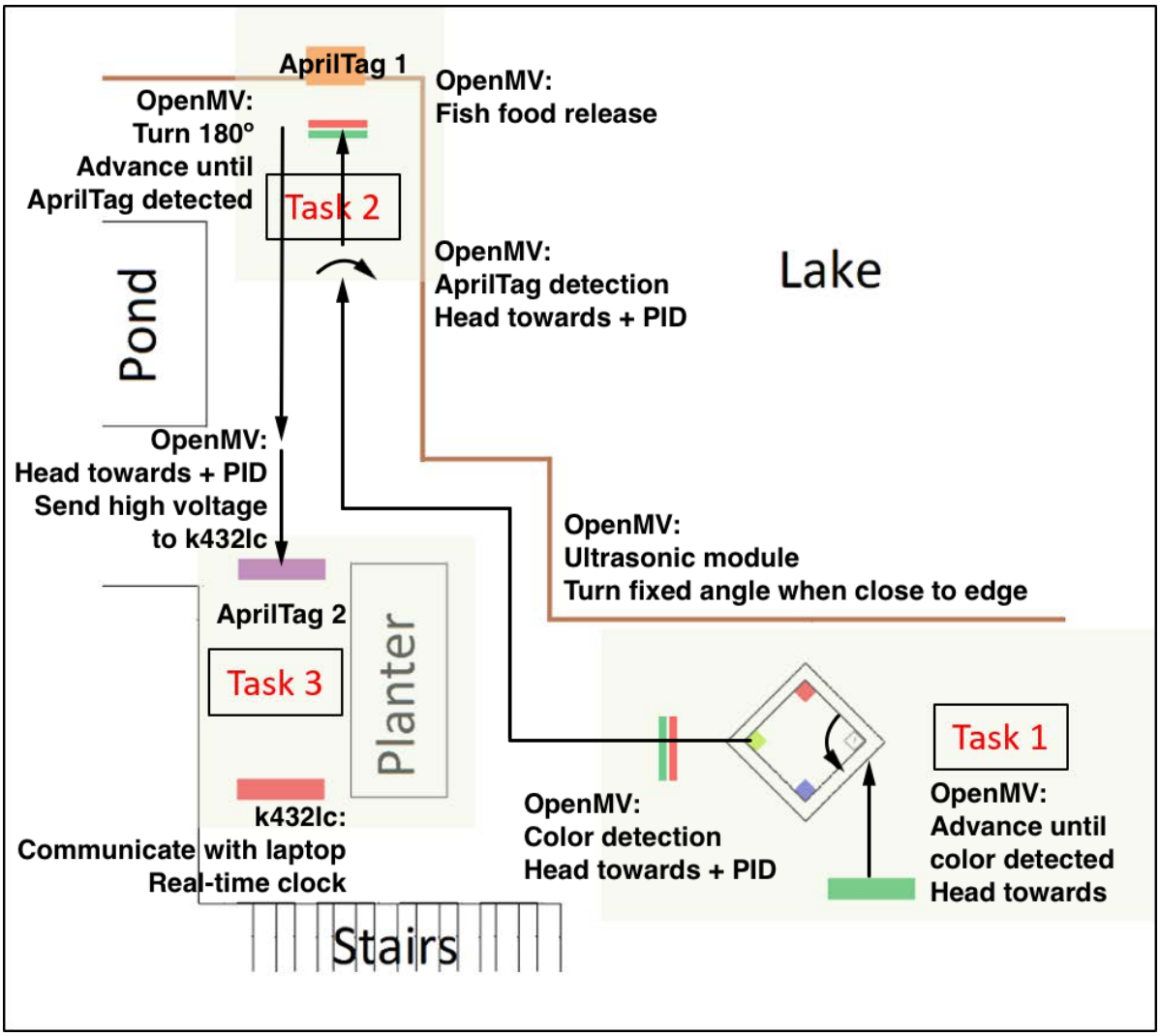


Figure 3.35: The approach to complete tasks required on Patio 2

Improved approach

To handle of the aforementioned problems, the team takes another integration approach. The main distinction between the new approach and the previous one is that some of the functions that used to be performed on the individual mbed microcontroller – the direction control (PID) algorithm, PWM output, fish food releasing, and data input from the accelerometer – are reassigned to the build-in microcontroller on the OpenMV Cam. This change simplifies the initial UART communication to a single link between a pair of digital input (on the mbed microcontroller) and output ports (on the OpenMV Cam), and the wireless communication initiates when an input value of '1' is detected. In a word, the UART communication is substituted with a single-wire channel for an "initializer" signal which triggers the communication process. This approach resolves the circuital clashes caused by the switch/multiplexer by bypassing the hardware component.

In general, the system integration on Patio 2 is demanding in establishing communication channel between the OpenMV module and the individual microcontroller, operating the PID control algorithm, measuring the paths and programming timed movements. The functions defined and utilized in each subsystem are encapsulated as separate function and header files that can be called and executed with ease in the central control program of Patio 2. Technical details are explained in the following content.

The overall design has been tested for multiple times and the robot appears to be capable of performing all the required tasks. However, there are numerous minor issues where possible fu-

ture improvements might focus on. Should the problems be addressed properly in the future, the performance and the robustness of the system could be further guaranteed.

Backup Plan

In case of a fatal crash, the team has prepared a backup plan to guarantee that the robot race can be completed thoroughly. This plan features measuring and timing every single move of the machine and program these simple movement parameters into the robot. Without most intelligent techniques, the robot is most likely to complete the tasks with the precise measurement and timing. The task diagram of this backup solution is displayed in Figure 3.36, and the task flow diagram is displayed in Figure 3.37.

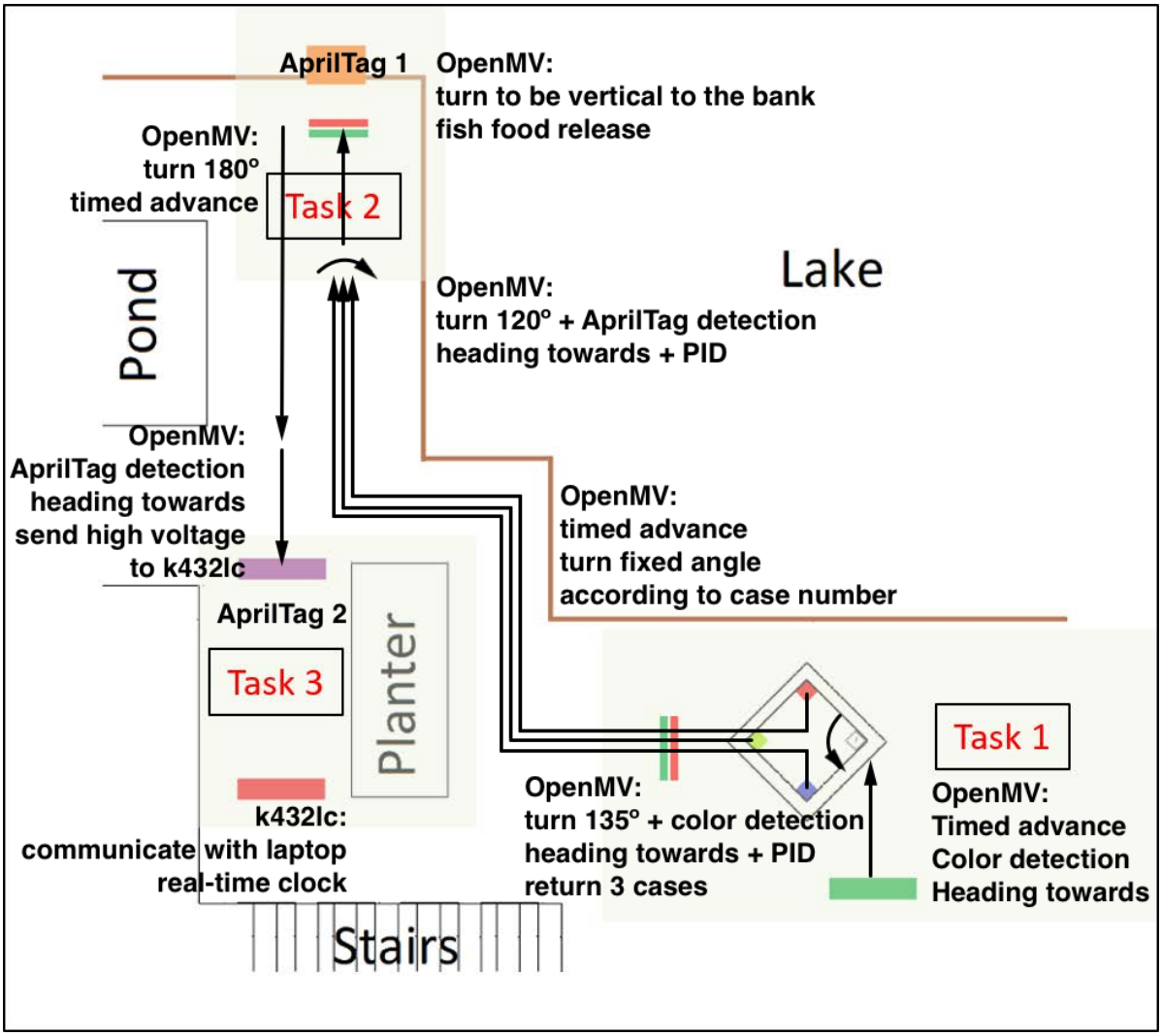


Figure 3.36: The backup approach to complete tasks required on Patio 2

The cardinal factor to accomplishing the tasks is to recognize the color chunks correctly and navigate the robot on the corresponding path. There are three different color chunks, so there are three possible paths. It is clearly noticeable that two of the three paths require the robot to perform turning maneuvers. Although the smart robot is able to utilize the visual module to recognize and navigate itself along the target path, the team reckons that this automatic and intelligent navigation approach relies on complicated programs and consumes excessive computing power of the microcontroller, and therefore adapts a much more straightforward method – preprogramming the path into the robot – to curtail the complexity of program writing and preserve sufficient computing power to prevent a critical mid-way crash. The key idea is that the robot performs fundamental maneuvers,

i.e. moving forward and making turns, to fit the preassigned paths, which is possible because there are only three different paths and each is a combination of limited straight lines and turns; were the paths are generated and assigned arbitrarily, this pseudo-intelligent approach would have been futile. With numerous measurements, calculations and experiments on the paths, the team has fine-tuned the maneuvering parameters, e.g. moving time and direction, and managed to direct the robot on routes that coincides with the preassigned paths. In the program, all movements are precisely measured and timed in order to exactly match the target paths and prevent breaking down.

Another major simplification is disabling the ultrasonic sensor so that the distance measurement and calculation do not occupy any computing power. As the robot's paths are all preprogrammed, the robot would not require the measured distance to support the decision-making process. This leaves more computing power to the remaining algorithms and prevents the robot from crashing.

Although all the paths and movements are preprogrammed, the OpenMV Cam only needs to perform visual recognition tasks – the most important part of Patio 2. Since all the paths are readily measured and timed, the main tasks regarding visual perception are color detection and AprilTag beacon recognition. As all the commands are programmed using simple timed movement sentences, the robot would have little burden processing and executing the commands.

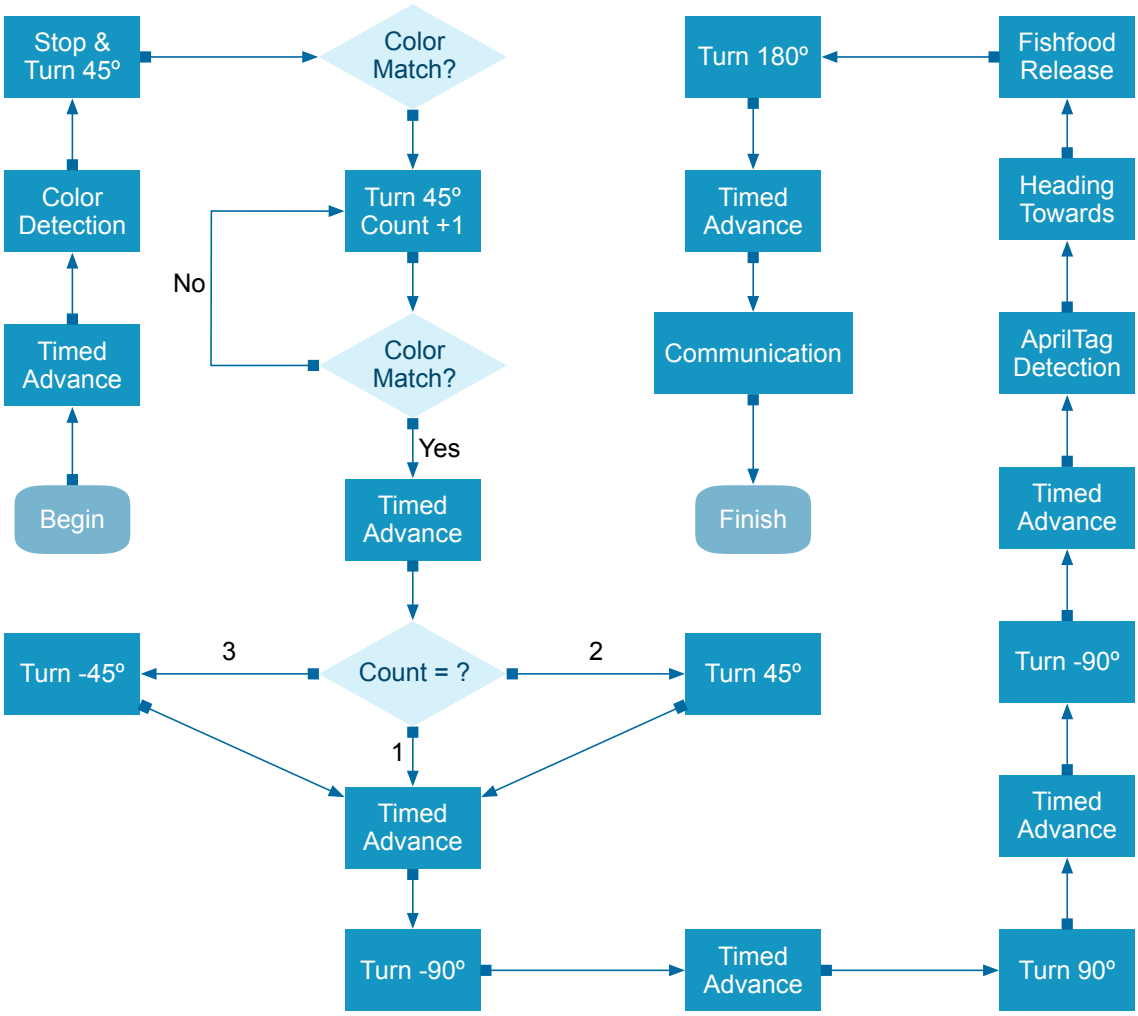


Figure 3.37: The flow diagram of the backup program design on Patio 2

3.6.2. Technical Details

Practical Color Detection

The OpenMV Cam needs to recognize the color placed at the rightmost corner before deciding which path to travel on. When the color chunk is recognized, the robot stops on the position and rotates for

135° to scan all remaining color chunks and identify the same color among them. Since the detection outputs RGB colors, they need to be converted to CIELAB colors to input to color detection functions.

Having identified the identical target color, the robot would head towards the color chunk. Once arrived, the robot would proceed to travel along the given path, with coordinated efforts from OpenMV and the ultrasonic sensor. In the backup plan, however, all the movements are precisely measured and programmed, so the visual and ultrasonic module are not needed.

AprilTag Deployment
On Patio 2, two AprilTags are deployed and utilized to support the robot at two critical points – the fish food release site and wireless communication area, as illustrated in Figure 3.35. As is mentioned in the visual module design, once the beacon is detected and recognized, the robot calculates the distance and the relative angle from its current to the AprilTag beacon. Based on the data, the robot adjusts course and determines whether it has arrived at the tasking area.

The Use of PID Algorithm
Generally speaking, whenever the robot enters a new path from a turn or complete start, the PID algorithm is executed to correct the angle deviation. This allows the robot to travel along straight lines without zig-zag movements.

Fish Food Release
When the distance between the robot and the AprilTag falls to the assigned range, the robot decides that it has arrived at the lake bank. A high-level digital output signal is transmitted to activate the autoloading ejector, and the device starts to eject pond pellets to the lake.

Wireless Communication
Likewise, as the robot approaches the communication area, the system determines whether the machine has arrived at the preassigned region with the measured distance from itself to the second AprilTag beacon. Immediately after this decision, the OpenMV Cam transmits high-level output signal to the mbed microcontroller, and the controller board triggers the communication module to transmit data to the laptop terminal.

Once the wireless communication is initiated, the mbed microcontroller first acquires current time data from DS1302 timekeeper, and combines the required information into one single message. This message is transmitted to the laptop terminal via serial communication through a pair of HC-12 transceivers. The data transmission repeats every second until the process is terminated, and team members can read the received data on the communication terminal interface.

4

Results & Discussion

In the previous chapter, the author has already provided detailed and comprehensive explanation of the design approaches that the team adapted to implement a smart robot to complete all the required tasks on both patios. Since all standalone hardware components and module programs are tested to have expected outcomes as standalone subsystems, the team proceeds to evaluate the functionality of these components and modules. The performance of the robot is also discussed in great detail.

4.1. Hardware Design

The triple-deck structure of the robot has allowed adequate space to accommodate all the necessary components, and guaranteed the visual module a vantage point to capture the desired images. The plastic-made decks have significantly reduced the overall weight of the robot. With the strong power source, the propulsion provided to the robot by the tracks allows the machine to overcome most of the small obstacles on its path and climb on steep slopes. Tests have revealed that the robot could climb on a maximum slope of 45° at a steady speed – adequate for completing the project objectives.

The experiments and final demonstration have proven that the robot is capable of completing all the tasks on both patios. However, there have been numerous minor issues that undermine the overall performance of the robot.

The final robot used for demonstration is not a completely finished product, even far from being an eligible prototype. All the components and circuit arrangements are done by simple welding and Dupont wires and they are not protected or reinforced. This leads to the possibility of detachment of the wire connections and destruction of circuit arrangement in the middle of the robot race.

The rudimentary structure of the smart robot only allows the team to install the most basic motors, as the hardware equipment chosen for robot design does not support more advanced step motors. This has further simplified the hardware design, but increased the complexity of program writing: the primitive functionality of such primitive motors impedes the programming members to include advanced control functions.

The mistaken welding of the wires on the batteries has also given rise to potential risk. The wire was welded in the opposite direction, and repairing this problem requires the team to buy new batteries, because the team members' expertise is not sufficient for correcting the wrongly welded wires on its initial spot. As the first attempts to run the robot concluded without fatal breakdown, the team considers this problem "tolerable" if taken sufficient care of the machine. In following tests, the team notices that there will be a short circuit if any metal is placed at or touches the joint points of the wires when the batteries are not at use.

The battery capacity and remaining power cannot be tracked by data quantification, as the primitive battery design does not allow data feedback. The only sign of low battery level is the anomalously low working efficiency of the robot, and this would not be obvious at times. This implies that the robot could be shut down because the battery has been exhausted in the middle of a task. If advanced equipment is accessible, this problem would have been resolved.

The OpenMV Cam is designed to be fixed on acrylic board to provide the visual module with a downward angle so that the captured images can be better processed. Since the visual recognition

tasks differ in either patio, the downward angle should vary as well to suit the circumstances – 45° and 90° respectively in the final design. Although the design prevents human errors in rearranging the circuit, this feature has brought about another inconvenience: the team has to remove and reinstall the OpenMV Cam in the transition to another patio.

For several times, the robot reacts very slowly to the target objectives. This might be ascribed to the delay due to data transmission. The complicated system design and primitive fashion of hardware assembly have made it more difficult to assess and enhance the performance on more minuscule aspects of the circuit design, and system delay is a common issue with such circuits. Better circuit design would reduce the delay dramatically if more found is guaranteed.

With the complex circuit design and the pressing time limitation, the team has no direct access to more professional hardware components and only could build the entire system using breadboard and single wires. These choices of hardware components are unstable when assembled as an entity: although they could manage to achieve the tasks, these components are still susceptible to changes in circuit arrangements and environmental conditions. The poor compatibility of the implemented hardware is one of the major reasons to blame, and this is mainly due to the deficient finance of the team project that has restricted the purchase of better-quality modules and components.

4.2. PID direction control

To investigate which type of controllers optimizes the smart robot's performance, the team models three different control systems – P controller, PI controller and PID controller – in Simulink and simulates their normal functions to compare their response speed which is indicative of the effectiveness of the control system. By comparing their output and error response at a given parameter setting, the team can evaluate the controllers and conclude which controller optimizes the robot performance. The Simulink simulation is done on the system modeling the three types of controllers, as shown in Figure 4.1. Note that the Laplacian domain equivalent form of the three controller terms are utilized instead of their time domain forms.

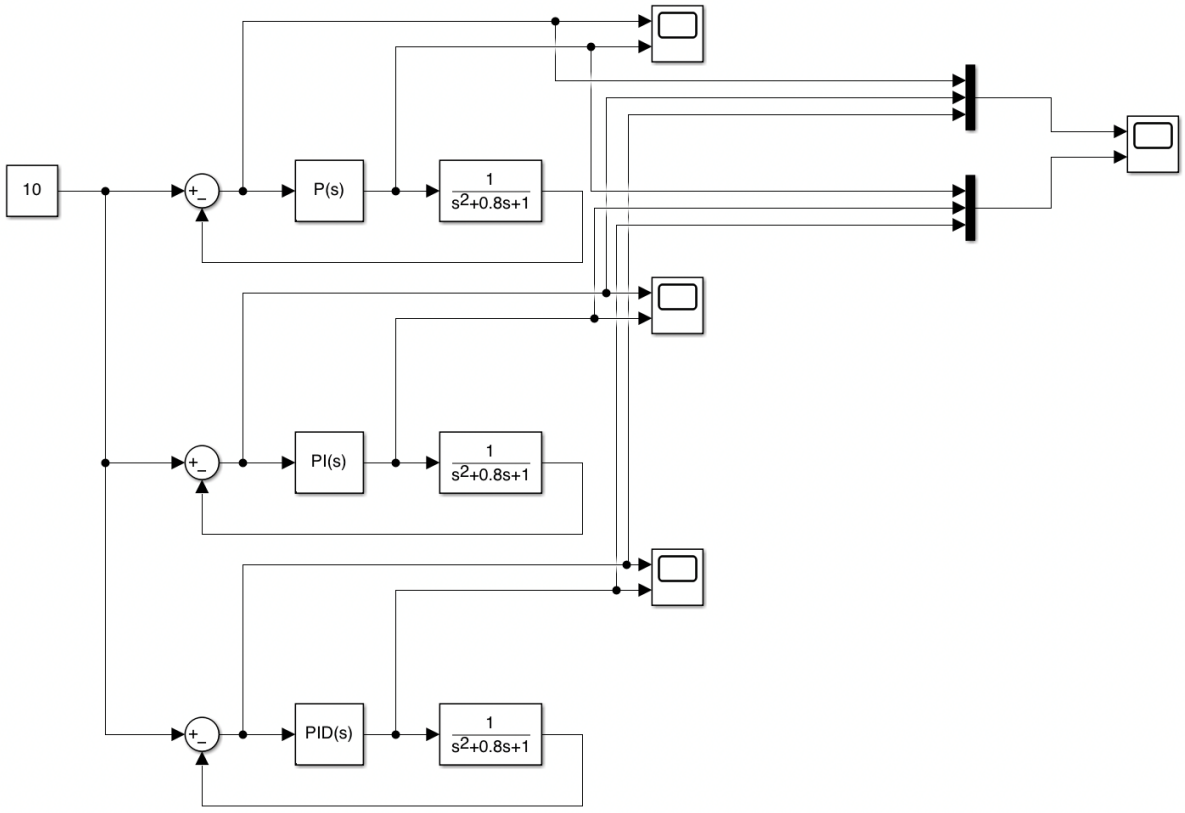


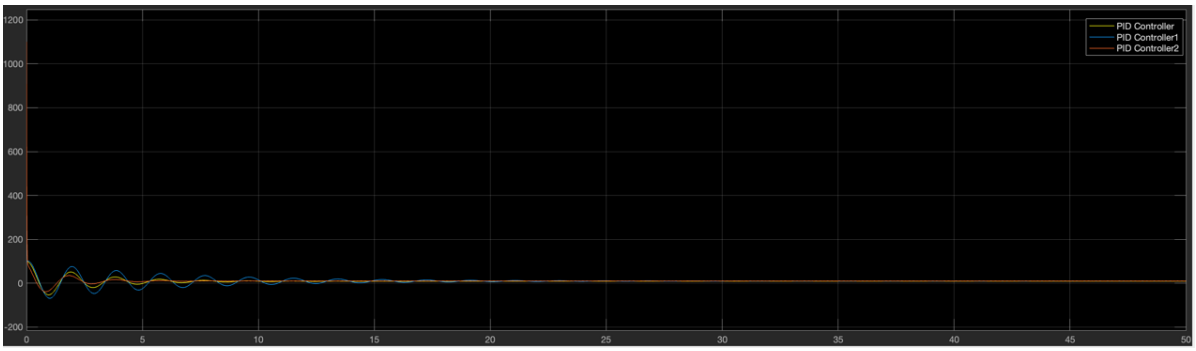
Figure 4.1: The schematic diagram of Simulink modelled direction control system

In the simulation, the parameters are set as Table 4.1.

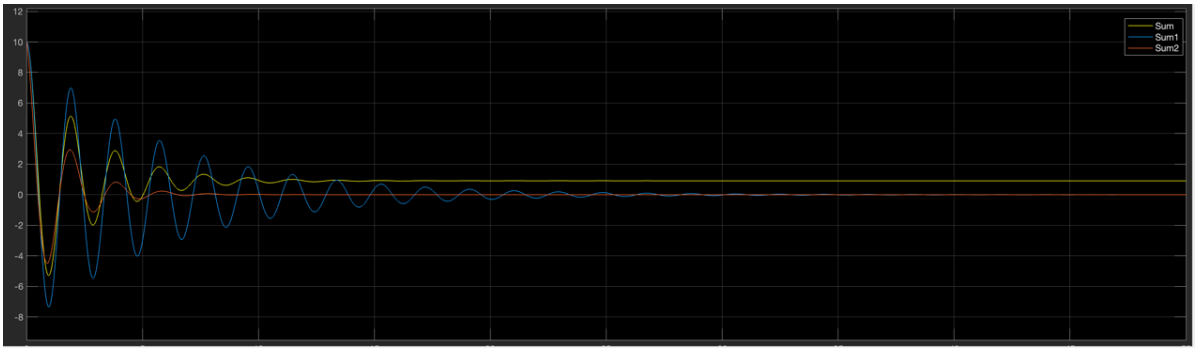
Controller Type	`kp`	`ki`	`kd`
P	10	0	0
PI	10	5	0
PID	10	5	1

Table 4.1: The different parameter settings of P, PI, PID controllers

The error and output curves are plotted and compared in Figure 4.2. The author notices that a steady-state error is present in P control system, indicated by an offset in the error curve after the curve ceases to oscillate. This leads to the conclusion that P control is unable to eliminate the error after adequate functioning time. Another immediate discovery emerges that the initial output of PID control system appears to be an impulse-like spike (visible in Figure 4.2a), which is difficult to eliminate with ordinary techniques (especially for controlled objects with large inertia and system delay). On the other hand, PI control system outperforms its counterparts for having minimal steady-state error and no uncontrollable spikes. After numerous simulations, the author concludes that PI controller is optimal for direction control of the smart robot, and the `kp` and `ki` constants are found to be 0.45 and 0.2 respectively when the control effect culminates. In the central program, the constant values are assigned by writing the sentence `PID(p = 0.45, i = 0.2)`, as the functions have been readily encapsulated as a header file that allows direct use of the defined functions in the wireless communication module program.



(a) The measured *output* response curves, different colors representing different controllers



(b) The measured *error* response curves, different colors representing different controllers

Figure 4.2: The output and error responses of the controllers in the simulation

After programming the robot, the team tests the robot's actual performance at the patio terrains. The error and output data of the robot is sampled and depicted in Figure 4.3. It can be concluded from the data that the PI controller can promptly correct the deviation from the target direction and maintain the robot on the preassigned path.

One issue regarding the direction control module is that live monitoring of performance is consid-

ered highly difficult to materialize. Should the PID parameters be monitored real-time, the wireless communication module would execute more complicated programs and, most importantly, need to operate continuously throughout both patios, which requires the robot to multitask for long periods of time. This adds to the uncertainty of the program execution, and due to time limitations on completing the design task, the team did not anticipate to achieve the objectives.

Another point to make is that however excellently the PID algorithm performs, it cannot correct excessive direction deviation due to inaccurate measurement. If the accelerometer outputs inaccurate or erroneous angle reading, the PID algorithm is not capable enough to fix such big errors.

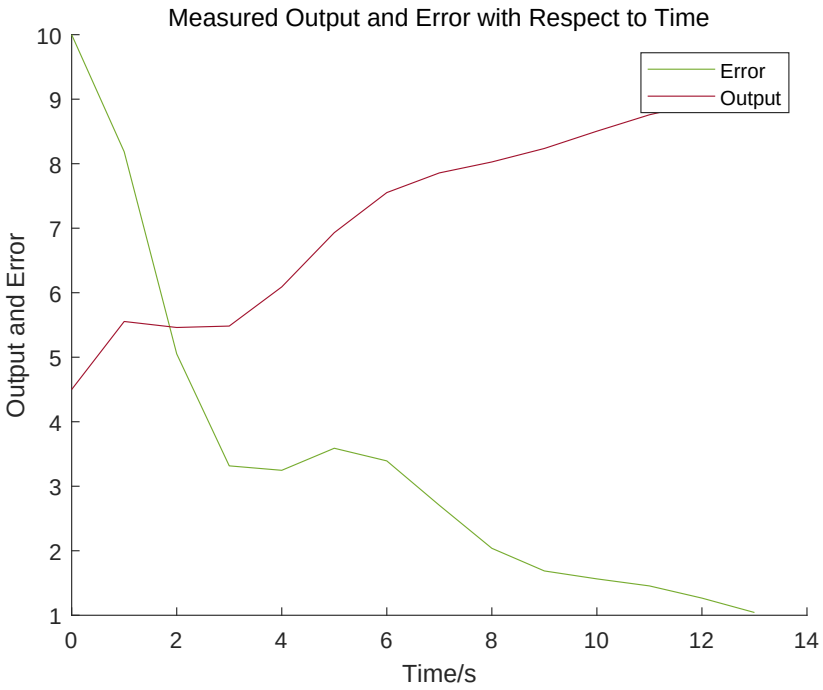


Figure 4.3: The output and error data collected from an actual experiment

4.3. Ejector Test

Although experiments have shown that the gearbox in the ejector is fully functional when the device is triggered by a digital high-level signal, there is still insufficient proof that the ejector could be able to eject the fish food (pond pellets) as expected. More evidence is demanded to further guarantee the normal functionality of the device.

Figure 4.4: The SAP granules and fish food (pond pellets) have similar size

Since the fish food granules are not provided before the demonstration, the team has chosen Super Absorbent Polymer (SAP) granules. SAP is a new functional polymer material. It combines hydrophilic groups, which can absorb a large amount of water while swelling and keep water from flowing out, such as starch grafted acrylate, grafted acrylamide, high degree of substitution crosslinked carboxymethyl cellulose, crosslinked carboxymethyl cellulose grafted acrylamide, crosslinked hydroxyethyl cellulose grafted acrylamide polymer, etc. Generally, it can absorb more than 100 times the volume of resin, and the highest water absorption rate can reach more than 1000%. It is generally used as medical materials, such as diapers, sanitary napkins, etc., and also used as plugging materials in industry [29]. Once these small balls have absorbed sufficient amount of water, they expand to a size (approximately 7mm in diameter) similar to the target fish food granules. Therefore, the team commences experiments on the device by loading in SAP granules in replacement of the actual fish food, because their similar sizes may lead to the same results.

The ejector test mainly commands the device to eject SAP granules which resemble the fish food in many aspects. If the team will track the projectile of the ejected and record relevant quantities that help evaluate the performance, such as the ejection range, recoil of the device the at moment of ejection.

After several experiments, the SAP granules can be ejected consecutively at the exact loading rate of *9 rounds per second*. The projectiles of the ejected granules cover an area as far as *8 meters away* from the device. The results are shown in Figure 4.5, where the projectiles end at the ripples on the lake surface. The long range of ejection is considered sufficient for fish food release.

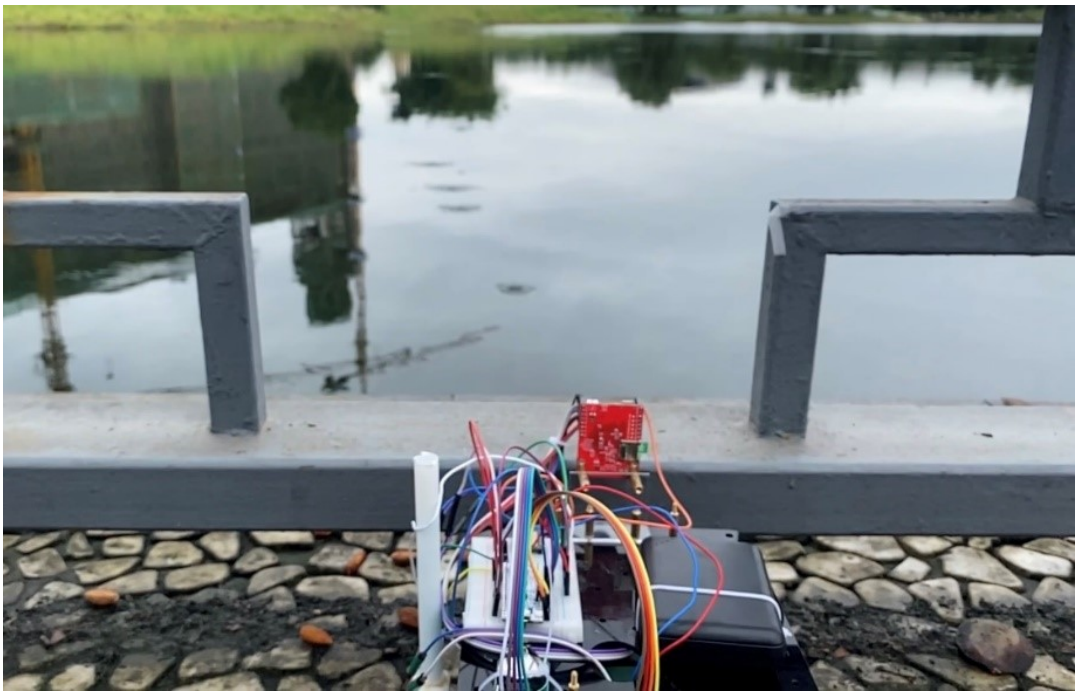


Figure 4.5: The ripples on lake surface indicate the point of impact

Despite the positive conclusion, the ejector displays numerous issues that would impair the overall stability and robustness of the system. Although the high rate of automatic loading and ejection, the operating power required to sustain the automatic ejection is excessively high, and is even estimated to be greater than the operating power of the motors. This indicates that the autoloading ejector and motors cannot function normally in simultaneity, that is, the robot cannot eject fish food (or other granules) while travelling to another site.

One ramification of the excessive operating power is the severe energy loss, mostly in kinetic and thermal energy in this design. Since the thermal energy dissipation is determined by the intrinsic properties of the motors, this source of energy loss is considered uncontrollable if better motors are not available. The kinetic energy loss is mostly contributed by the drastic recoil of the ejector while operating continuously.

Beside the severe energy loss, the recoil force still has other negative effects. While the autoloading ejector is operating continuously, the recoil force shakes skeleton of the robot. Since the internal structure and circuit wiring of the robot is not reinforced and protected by strong and resilient materials, the heavy shakes are likely to destruct the circuit arrangement, and reconstruction requires thorough examination of all components and modules. The recoil force could also damage the installation slot, causing a potential danger of midway detachment of the device.

Although the maximum range reached by the ejected granules is measured to be as far as 8 meters, the variation has been random, the horizontal spread of the granules is obvious, and the accuracy of ejection is subpar. These could be attributed to the drastic recoil when the ejector is operating at full power.

4.4. Wireless Communication

After referring to online mbed forums and supplementary materials, the author has adapted an embedded C program to realize wireless communication between the robot and the laptop. When the microcontroller executes the program, it begins to transmit the target information repeatedly after each time interval of one second, as shown in Figure 4.6. This accomplishes the required task of wireless communication.

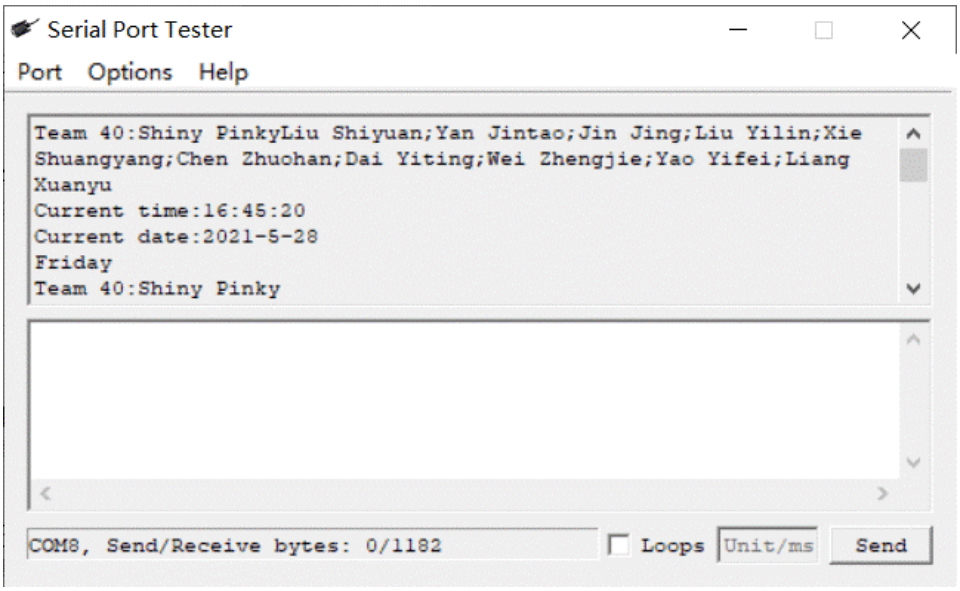


Figure 4.6: The received message on the laptop terminal user interface

Delay due to data processing and transmission is one main contributor to the system latency in the design scheme. Since the command from the central processor (built-in microcontroller on OpenMV) needs to propagate through the mbed microcontroller to reach the communication module, the processing time of received signals and the execution time of programs at the microcontroller have accumulated to a visible delay in the robot. Were the mbed board removed from the design and the wireless communication directly commanded by the OpenMV Cam, the delay would have been significantly curtailed.

The delay in time detected by DS1302, however, is difficult to manage and eliminate. In fact, the DS1302 module is not in nature a digital clock – it merely calculates the current time and keeps monitoring the data instead of acquiring the data directly from the computer – and since the first initialization of the module is conducted manually, there would be unavoidable error in the clock data due to program execution time. In final experiments, the delay was measured to be several seconds, but this delay does not impair the task completion on both patios.

4.5. Visual Recognition

The final results of the demonstration have shown that the visual module functions normally and the data processed by the designed programs effectively supports the central processors to make decisions. However, there are still minor issues that affect the performance of the robot.

Despite the team's efforts in adjusting the threshold value range for the brightness axis in the CIELAB representation of the captured images, the outcome of this consideration is not as satisfying as expected. During the final testing days when weather conditions change frequently and drastically, the visual module's operations had been consistently checked by the varying sunshine exposure level. As numerous attempts to tune the threshold values have failed during this period of time, the team could only rely on benign weather conditions to expect excellent performance of visual recognition.

When the team was testing the ultrasonic sensor as the robot approaches the preassigned site for fish food release and wireless communication, the robot some times stopped prior to the anticipated time and proceeded to the next task. After much investigation, the team concluded that the flying insects like mosquitoes must have flown across the propagating ultrasonic waves and reflected much of them back to the sensor, leading to the erroneous conclusion that the robot had entered the preassigned area. To handle this problem, the team adds a feature that resembles the *debounce* function in the touch pad design: only if the received signal remains high level in five consecutive periods of detection, the measurement is validated.

The team once adapted the line tracking function to the bridge crossing task, but was frustrated to find that in over half of the attempts, the robot ran into the supporting pillar of the bridge and was unable to redirect itself to the correct path. Therefore, this feature was removed and the robot would proceed using the initial approach to cross the bridge.

After a heavy rain, the patios' ground became muddy and moist, and some dark soils have covered the initial edges of the paths. This was tantamount to blurring the edges of the roads, and as an inevitable result the robots' performance had dwindled during the day. This external disrupting factor was removed as the campus staff cleaned the patio ground. Were the conditions remain unchanged for a longer time, the robot would have needed much improvements on the visual module, which might again necessitate the DL solution for avoiding complicated theories and algorithms.

4.6. Patio 2: Timed Advance

As is mentioned in **Patio 2 system integration**, in the backup plan, the team has three different paths to measure and preprogram into the robot, and this has greatly reduced the burden of the OpenMV and mbed microcontroller. As is shown in Figure 3.36, the first path starts from the green start line and the color chunk placed at the rightmost corner of the square, the second path includes three possible options depending on the color chunk recognized by the visual module, and the last path starts from the fish food release site and ends at the wireless communication area. These paths can be measured and a relatively accurate result can be obtained by taking the average values of the collected data. All pertinent data collected and analyzed by the author is displayed in Table 4.2.

Time/ms		T1	T2	T3	T4	T5
Path 1		9800	3500	/	/	/
Path 2	Case 1	6900	15300	7300	9600	11000
	Case 2	9800	9600	11700	9000	11000
	Case 3	6900	16000	15600	8300	11000
Path 3		31000	/	/	/	/

Table 4.2: The average measured time piece-wise travel on each linear path

At turning points, the robot is programmed to stop and rotate to the target direction before pushing forward again to minimize possible errors and measure the desired data. Likewise, the author collects multiple sets of experimental data and takes the average values to eliminate the errors. If the measured data deviates from the theoretical value, the parameters in the program should be adjusted until the actual turning angle agrees with the theoretical values. The process will also be assisted by AprilTag and PID algorithm during the actual demonstration. The data collected and an-

alyzed by the author is displayed in Table 4.3. The actual average value display slight deviation from the theoretical value, most likely as a result of subpar equipment accuracy when measuring the data.

Target Angle/°	45			90			135			180		
Actual Angle/°	41	43	40	93	87	79	126	130	113	160	173	159

Table 4.3: The inputted angle and measured actual turning angle

Note that in the table above that the data apply to both directions. As we can see, the turning angle inputted into the motor and the actual output angle have an average deviation of around 10°. This error could be the result of intrinsic inaccuracy of the angle measuring module, and manual efforts are needed to calibrate the reading and adjust the actual output angle to the desired one.

Another contributing factor to the inaccuracy of angle measurement and calculation is the bumpiness in performing turning maneuvers. If the robot experiences excessive and uncontrolled shake during a turn, the accelerometer could obtain erroneous readings of the angle data and lead to the misjudgment that the robot has completed the turn while it is not. If the robot design includes any shake reduction measure, the validity of this speculation could be evaluated.

5 Conclusion

The desired smart robot that needs to accomplish several assigned tasks was designed and implemented successfully by the end of the project. In the robot race, it completed all the tasks on the assigned patio and the performance was seamless.

The initial smart robot design scheme was presented soon after the commencement of the three-month project. In initial experiments and attempts to realize the subsystems, the team identified numerous issues regarding hardware design and programming approaches. To handle these issues, initial subsystem designs were modified or replaced with new design schemes. By the end of the second month, all subsystems could function normally when connected to the programming laptop.

The subsystems were assembled and tested on actual sites of the final demonstration in the third month. At the first trails of operating the robot to perform the assigned tasks on site, the team has encountered many difficulties, and the design scheme experienced frequent modification to improve the performance of the robot on both patios. The performance of the subsystem components is recorded and evaluated, and the modifications of design scheme are justified in detail.

The team expects an enhanced outcome if more financial support is guaranteed. The purchased components and robot parts, though functioning normally as standalone modules, appeared to be insufficiently durable for repetitive use. Besides, the limited finance could only support primitive circuit designs prone to external influences. These have added to the difficulty of proper maintenance. To further increase the overall system stability, more high-quality, well-designed and thus more costly components are needed.

References

[1] Shuzhen Wang. *High-Speed Track Vehicle Driving Systems*. Beijing: Beijing Institute of Industrial Technology Press, 1988.

[2] *Dual Full-Bridge Driver*. L298. STMicroelectronics. Jan. 2000.

[3] Wikipedia contributors. *PID controller — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=PID_controller&oldid=1027829138. [Online; accessed 17-June-2021]. 2021.

[4] GT Shrivakshan and Chandramouli Chandrasekar. "A comparison of various edge detection techniques used in image processing". In: *International Journal of Computer Science Issues (IJCSI)* 9.5 (2012), p. 269.

[5] D Poobathy and R Manicka Chezian. "Edge detection operators: Peak signal to noise ratio based comparison". In: *IJ Image, Graphics and Signal Processing* 10 (2014), pp. 55–61.

[6] Ehsan Nadernejad, Sara Sharifzadeh, and Hamid Hassanpour. "Edge detection techniques: Evaluations and comparisons". In: *Applied Mathematical Sciences* 2.31 (2008), pp. 1507–1520.

[7] TG Smith Jr et al. "Edge detection in images using Marr-Hildreth filtering techniques". In: *Journal of neuroscience methods* 26.1 (1988), pp. 75–81.

[8] OpenMV LLC. *image — machine vision*. https://docs.openmv.io/library/omv.image.html?highlight=image#module-image. 2021.

[9] Ian Goodfellow et al. *Deep learning*. Vol. 1. 2. MIT press Cambridge, 2016.

[10] Wikipedia contributors. *Convolutional neural network — Wikipedia, The Free Encyclopedia*. [Online; accessed 18-June-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Convolutional_neural_network&oldid=1026397751.

[11] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*. PMLR. 2015, pp. 448–456.

[12] Wikipedia contributors. *RGB color model — Wikipedia, The Free Encyclopedia*. [Online; accessed 18-June-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=RGB_color_model&oldid=1029017035.

[13] Wikipedia contributors. *RGB color space — Wikipedia, The Free Encyclopedia*. [Online; accessed 18-June-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=RGB_color_space&oldid=1028703159.

[14] Wikipedia contributors. *CIE 1931 color space — Wikipedia, The Free Encyclopedia*. [Online; accessed 18-June-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=CIE_1931_color_space&oldid=1027748935.

[15] Bruce Justin Lindbloom. *RGB/XYZ Matrices*. http://www.brucelindbloom.com/index.html?Eqn_RGB_XYZ_Matrix.html. 2017.

[16] Bruce Justin Lindbloom. *XYZ to Lab*. http://www.brucelindbloom.com/index.html?Eqn_XYZ_to_Lab.html. 2017.

[17] The APRIL Robotics Laboratory. *AprilTag*. https://april.eecs.umich.edu/software/apriltag. 2010.

[18] TZZQing. *AprilTag working principles and source codes*. https://blog.csdn.net/han784851198/article/details/90261197. 2019.

[19] Rob Toulson and Tim Wilmshurst. *Fast and effective embedded systems design: applying the ARM mbed*. Newnes, 2016.

[20] *Wireless Serial Port Communication Module*. HC-12. HiLetgo. Jan. 2016.

[21] Mark Hughes. *Understanding and Implementing the HC-12 Wireless Transceiver Module*. https://www.allaboutcircuits.com/projects/understanding-and-implementing-the-hc-12-wireless-transceiver-module/. 2016.

[22] *Trickle Charge Timekeeping Chip*. DS1302. Dallas Semiconductor. 2015.

[23] Thomas_Murphy. *Real Time Clock - DS1302*. https://www.instructables.com/Real-Time-Clock-DS1302/. 2016.

[24] *Precision ultrasonic sensor*. HY-SRF05. ETC.

[25] Web Contributors. *HMC5883L Angle Calibration and Formulae*. http://www.elecfans.com/dianzichangshi/20180308644594.html. 2018.

[26] *High-Speed, Low-Voltage, CMOS Analog Multiplexers/Switches*. MAX4617/MAX4618/MAX4619. Maxim Integrated Products, Inc. 2012.

[27] *CMOS Single 8-Channel Analog Multiplexer/Demultiplexer with Logic-Level Conversion*. CD405xB. Texas Instruments, Inc. 2021.

[28] Wang et al. Jianling. "Application of CD4052 as Mul-series Expander [J]". In: *Journal of Henan Mechanic and Electric Engineering College* 3 (2005).

[29] Cycoloy. *What are the airsoft gun bullets made of?* https://wenda.so.com/q/1458929655728398. 2015.

A

Programming Nuances

The content of report that has been covered above mainly concentrates on the theoretical and practical aspects of the teamwork project, while neglecting technical programming approaches that materialize design ideas of the smart robot. This chapter features the programming techniques explained in great detail. The program structure is not completely displayed but the key ideas of programming are illustrated.

A.1. Wireless Communication

The complete functionality of the timekeeping DS1302 module is defined in the following program. As the basic principle has been covered in previous content, there would be no more explanations.

Listing A.1: DS1302 complete module definition

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

DigitalOut sclk(D4);//CLK
DigitalInOut io(D5);//DAT
DigitalOut ce(D6);//RST

uint8_t tim[7]={0x30, 0x31, 0x16, 0x28, 0x05, 0x05, 0x21};
// second, minute, hour, day, month, week, year
uint8_t w[7]={0x80, 0x82, 0x84, 0x86, 0x88, 0x8a, 0x8c};
// write the address bytes
uint8_t r[7]={0x81, 0x83, 0x85, 0x87, 0x89, 0x8b, 0x8d};
// read address bytes
uint8_t second,minute,hour,week,day,month,year;

void ds1302writebyte(uint8_t dat)   // writing single byte
{
    uint8_t i,value;
    sclk = 0;
    wait_us(20);

    io.output();
    for(i=0;i<8;i++)
    {
        value=dat&0x01;
        if(value)
            io = 1;
        else
            io = 0;
        dat>>=1;
        sclk = 1;
        wait_us(20);
        sclk = 0;
        wait_us(20);
    }
}

```

```
38  void ds1302writebytes(uint8_t add, uint8_t dat) // write multiple bytes
39  {
40      ce = 0;
41      wait_us(20);
42      ce = 1;
43      wait_us(20);
44      ds1302writebyte(add);
45      ds1302writebyte(dat);
46      ce = 0;
47      wait_us(20);
48  }
49
50  uint8_t cmd_readbyte(uint8_t add){  // read commands
51      uint8_t i, value;
52      ce = 0;
53      wait_us(20);
54      ce = 1;
55      wait_us(20);
56
57      sclk = 0;
58      wait_us(20);
59
60      // write an address of one byte
61      io.output();
62      for(i=0;i<8;i++){
63          value=add&0x01;
64          if(value)
65              io = 1;
66          else
67              io = 0;
68          add>>=1;
69          sclk = 1;
70          wait_us(20);
71          if(i<7){
72              sclk = 0;
73              wait_us(20);
74          }
75      }
76
77      // read one byte
78      io.mode(PullUp);
79      io.input();
80      value = 0;
81      wait_us(20);
82      for(i=0; i<8; i++){
83          sclk = 0;
84
85          value>>=1;
86          if(io.read() == 1)
87              value|=0x80;
88          sclk = 1;
89          wait_us(20);
90      }
91
92      ce = 0;
93      wait_us(20);
94      sclk = 0;
95      wait_us(20);
96
97      return value;
98  }
99
100 void settime()
101 {
102     uint8_t i;
103     ds1302writebytes(0x8e,0x00);
104     for(i=0;i<7;i++)
105     {
106         ds1302writebytes(w[i],tim[i]);
107     }
108     ds1302writebytes(0x8e,0x80);
```

```
}

void readtime()
{
    second=cmd_readbyte(r[0]);
    minute=cmd_readbyte(r[1]);
    hour=cmd_readbyte(r[2]);
    week=cmd_readbyte(r[5]);
    day=cmd_readbyte(r[3]);
    month=cmd_readbyte(r[4]);
    year=cmd_readbyte(r[6]);
}

void DS1302_SubMain(){

    settime();
    // this line is used for the first initialization,
    // delete the first line after first initialization
    readtime();
}
```

As the program above will not be part of the main program for wireless communication, a header file is needed to extend the use of functions defined in this program to the main program. The header file makes the use of variables of type `uint8_t` applicable in external environments, and extends the functionality of the "initializer" and time acquisition functions. The header file is written as follows.

Listing A.2: DS1302 header file

```
#ifndef DS1302_H
#define DS1302_H

#include "mbed.h"

extern uint8_t second,minute,hour,week,day,month,year;
void readtime();
void DS1302_SubMain();

#endif
```

With all the functions of wireless communication and time acquisition well defined in the header files, the program executing the wireless communication can be written as follows.

Listing A.3: Wireless communication programming realization

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

float Humi, Temp;
int check;
// defines necessary variables to use

Serial pc(USBTX, USBRX); // Laptop side TX and RX
Serial hc12(D1, D0); // HC-12 side RX

int wireless_comm()
{
    while (1) {
        DS1302_SubMain();
        // initiates the clock module, acquires time data
        hc12.baud(9600);
        // sets the UART communication baud rate to 9600
        hc12.format(8, SerialBase::None, 1);
        // receives data from the serial port and sends it to laptop
        hc12.printf("Team 40:Shiny Pinky\n\r");
        // prints team name and number
        hc12.printf("Liu Shiyuan;Yan Jintao;Jin Jing;Liu Yilin;Xie Shuangyang;Chen Zhuohan;
    Dai Yiting;Wei Zhengjie;Yao Yifei;Liang Xuanyu\n\r");
        // prints team member list
        hc12.printf("Current time:%x:%x:%x \n\r", hour, minute, second);
        hc12.printf("Current date:20%x-%x-%x \n\r", year, month, day);
```

```
26          // prints time data from DS1302 defined variables
27
28          switch(week){
29              case 1: hc12.printf("Monday \n\r");
30              break;
31              case 2: hc12.printf("Tuesday \n\r");
32              break;
33              case 3: hc12.printf("Wednesday \n\r");
34              break;
35              case 4: hc12.printf("Thursday \n\r");
36              break;
37              case 5: hc12.printf("Friday \n\r");
38              break;
39              case 6: hc12.printf("Saturday \n\r");
40              break;
41              case 7: hc12.printf("Sunday \n\r");
42              break;
43              }
44          // print the day according to the obtained number case
45          thread_sleep_for(1000);
46      }
47  }
```

Lines 1-9 provide preliminaries of wireless communication module, including header files and defining necessary variables and the ports for data transmission and reception. Lines 11-47 define the main function `int wireless_comm`. The function first initiates the timekeeper and transceivers (lines 14-19), and then transmits the desired information (lines 20-43). Note that the weekday data is acquired with the digital time data. After each transmission, the program pauses for 1000ms before repeating the same process (line 45).

A.2. PID Algorithm

As is mentioned in previous content, the preliminaries of the direction control mainly include four categories: defining the data class and three control terms according to the optimal controller type. The program is written as follows.

Listing A.4: PID algorithm preliminaries

```
1  class PID:
2      _kp = _ki = _kd = _integrator = _imax = 0
3      _last_error = _last_derivative = _last_t = 0
4      _RC = 1/(2 * pi * 20)
5      def __init__(self, p=0, i=0, d=0, imax=0):
6          self._kp = float(p)      # proportional gain constant
7          self._ki = float(i)      # integral gain constant
8          self._kd = float(d)      # derivative gain constant
9          self._imax = abs(imax)   # auxiliary variable in I control
10         self._last_derivative = float('nan')   # auxiliary variable in D control
11
12 # defines new data class that receives and carries information required by the algorithm to
       ease data operation
13
14 self._last_t = tnow
15 delta_time = float(dt) / float(1000)
16 output += error * self._kp
17
18 # programs the proportional control section individually
19 # this part describes the proportional contribution to the overall function output, and seems
       like the basis for all other controls...
20
21 if abs(self._ki) > 0 and dt > 0:
22     self._integrator += (error * self._ki) * scaler * delta_time
23     if self._integrator < -self._imax: self._integrator = -self._imax
24     elif self._integrator > self._imax: self._integrator = self._imax
25     output += self._integra
26
27 # programs the intrgral control section individually
28 # this part describes the intrgral contribution to the final system output, and can be switch
       to zero if desired...
```

```
29
30 if abs(self._kd) > 0 and dt > 0:
31     if isnan(self._last_derivative):
32         derivative = 0
33         self._last_derivative = 0
34     else:
35         derivative = (error - self._last_error) / delta_time
36     derivative = self._last_derivative + \
37                             ((delta_time / (self._RC + delta_time)) * \
38                                 (derivative - self._last_derivative))
39     self._last_error = error
40     self._last_derivative = derivative
41     output += self._kd * derivative
42 output *= scaler
43
44 # programs the derivative control section individually
45 # this part describes the intrgral derivative to the final system output, and can be switch
       to zero if desired...
```

Note that lines 1-12 define the data class, lines 14-19 define the proportional (P) term of the controller, lines 21-28 define the integration (I) term of the controller, and lines 30-45 define the derivative (D) term of the controller. The definitions of three terms require respective constants **kp, ki, kd** be inputted, and since all three values are encapsulated in the data class, the controller can be activated using one single command that sets the values of constants in the data class.

A.3. Ultrasonic Distance Measurement

Based on the theory of ultrasonic measurement discussed in previous content, the ultrasonic distance measuring sensor can be executed by the program below. The main idea is to declare the hardware pins for trigger and echo signals and define the periodic measuring process of the module.

Listing A.5: Ultrasonic distance measurement

```
1  from pyb import Pin
2  import pyb
3
4  trig = Pin('P2', Pin.OUT_PP)
5  echo = Pin('P3',Pin.IN)
6  # defines two pins for trigger signal and echo signal reception
7
8  def distanceMeasure(timeout):
9    print('Distance Measure...')
10   trig.value(1)
11   pyb.udelay(100)
12   # generates rectangular pulses with period 100us
13   trig.value(0)
14   mtime = pyb.millis()
15   while ((echo.value() == 0) and (pyb.elapsed_millis(mtime) < timeout)):
16     trig.value(0);
17     # when echo turns low level, the echo signal validates
18   if (echo.value() == 1):
19     dtime = pyb.micros()
20     while ((echo.value() == 1) and (pyb.elapsed_millis(mtime) < timeout)):
21       trig.value(0)
22     # when echo turns high level, waits for low level echo
23     if (echo.value() == 0):
24       # turns low level before timeout and implies an obsticle
25       distance = pyb.elapsed_micros(dtime)
26     else:
27       print('TimeOut.');
28       return 0;
29       # indicates no obstacle or malfunctioning measurement
30     print('distance time: ', distance);
31     print(distance * 34000 / 1000000 / 2, 'cm')
32     return (distance * 34000 / 1000000 / 2)
33     # outputs distance data
34 # defines the distance measuring function as a seperate function
35
36 while (True):
```

```
37   distanceMeasure(100)
38   pyb.delay(900)
39 # sets time duration for distance measurement and delay before repeating the process of
       measurement
```

Lines 1-2 include the important library **pyb** which enables the use of Python in programming hardware components. Lines 4-6 declare the pins for trigger and echo signals. Lines 8-34 define the function to measure the distance based on the working principle of the module. The periodic measuring process is defined in lines 36-39. We can see that one period has 1000ms, where 100ms is allocated for distance measurement and the remaining 900ms is dormant state.

A.4. Turning based on accelerometer angle measurement

The key idea of making turns is to eliminate the angle difference between the current direction. The accelerometer first detects the angle difference and triggers the motor control circuit to produce velocity difference to turn the robot until the angle difference is eliminated. The process can be encapsulated as several functions as follows.

Listing A.6: Turning maneuver functions

```
1  import time
2  from pyb import UART
3  from pyb import Pin, Timer
4
5  inverse_left = False
6  # changes it to True to inverse left wheel
7  inverse_right = False
8  # changes it to True to inverse right wheel
9
10 ain1 = Pin('P0', Pin.OUT_PP)
11 ain2 = Pin('P1', Pin.OUT_PP)
12 bin1 = Pin('P2', Pin.OUT_PP)
13 bin2 = Pin('P3', Pin.OUT_PP)
14 # declares digital output pins
15
16 ain1.low()
17 ain2.low()
18 bin1.low()
19 bin2.low()
20 # initializes the outputs to low level
21
22 pwma = Pin('P7')
23 pwmb = Pin('P8')
24 pwmc = Pin('P4')
25 pwmd = Pin('P5')
26 # declares PWM output pins
27
28 tim = Timer(4, freq = 1000)
29 tim2 = Timer(2, freq = 1000)
30 # sets timers
31
32 leftp = tim.channel(1, Timer.PWM, pin = pwma)
33 rightp = tim.channel(2, Timer.PWM, pin = pwmb)
34 leftn = tim2.channel(3, Timer.PWM, pin = pwmc)
35 rightn = tim2.channel(4, Timer.PWM, pin = pwmd)
36 # initializes propulsion settings for making turns
37
38 uart = UART(1, 9600)
39 # initializes UART transmission
40
41 AngleData=[0.0]*8
42 FrameState = 0
43 # for determining the case from the numbers behind 0x
44 Bytenum = 0
45 # for determining which bit to read from the start data bit
46 CheckSum = 0
47 # for calculating the checksum
48 Angle = [0.0]*3
```

```
49
50  def DueData(inputdata):
51      global FrameState
52      global Bytenum
53      global CheckSum
54      global Angle
55      # defines global variables to allow modifications on partial scale
56      for data in inputdata:  # loop through all data
57          if FrameState == 0:   # in the case of undetermined state, do as follows
58              if data == 0x55 and Bytenum == 0:
59      # if 0x55 is at the beginning, starts to read data and increment bytenum
60                  CheckSum = data
61                  Bytenum = 1
62                  continue
63              elif data == 0x53 and Bytenum == 1:
64                  CheckSum += data
65                  FrameState = 3
66                  Bytenum = 2
67          elif FrameState == 3:
68              # angle data
69              if Bytenum < 10:
70                  AngleData[Bytenum-2] = data
71                  CheckSum += data
72                  Bytenum += 1
73              else:
74                  if data == (CheckSum & 0xff):
75                      Angle = get_angle(AngleData)
76                      return Angle
77                  CheckSum=0
78                  Bytenum=0
79                  FrameState=0
80  # segmentizes the data to read, based on data lists
81
82  def get_angle(datahex):
83      rxl = datahex[0]
84      rxh = datahex[1]
85      ryl = datahex[2]
86      ryh = datahex[3]
87      rzl = datahex[4]
88      rzh = datahex[5]
89      k_angle = 180.0
90      angle_x = (rxh << 8 | rxl) / 32768.0 * k_angle
91      angle_y = (ryh << 8 | ryl) / 32768.0 * k_angle
92      angle_z = (rzh << 8 | rzl) / 32768.0 * k_angle
93      if angle_x >= k_angle:
94          angle_x -= 2 * k_angle
95      if angle_y >= k_angle:
96          angle_y -= 2 * k_angle
97      if angle_z >=k_angle:
98          angle_z-= 2 * k_angle
99      return angle_x,angle_y,angle_z
100 # defines function that measures the angle data in three axis
101
102 def turn_angle(angle):
103     uart.writechar(0xff)
104     uart.writechar(0xaa)
105     uart.writechar(0x52)
106     time.sleep_ms(20)
107     # calibrates the module
108
109     while True:
110         datahex = uart.read(33)
111         if datahex:
112             Due = DueData(datahex)
113             if Due:
114                 break
115             pass
116     print(datahex)
117
118     ang1 = Due[2]
119     print(ang1)
```

```
120     # confirms the desired turning angle
121
122     if angle > 0:
123         leftp.pulse_width_percent(0)
124         rightp.pulse_width_percent(65)
125         leftn.pulse_width_percent(65)
126         rightn.pulse_width_percent(0)
127     else:
128         leftp.pulse_width_percent(65)
129         rightp.pulse_width_percent(0)
130         leftn.pulse_width_percent(0)
131         rightn.pulse_width_percent(65)
132     # basic turning settings in both directions
133     ang2 = 0.0
134     # reference angle for monitoring turning process
135
136     while not (abs(ang2-ang1) >= abs(angle)):
137     # when the turning has not been finished
138         while True:
139             datahex = uart.read(33)
140             if datahex:
141                 Due = DueData(datahex)
142                 if Due:
143                     break
144                 pass
145         ang2 = DueData(datahex)[2]
146
147     leftp.pulse_width_percent(0)
148     rightp.pulse_width_percent(0)
149     leftn.pulse_width_percent(0)
150     rightn.pulse_width_percent(0)
151     print("success!")
152     # concludes the turning process and sets all pins back
153 # defines the function to make a turn for a given angle
154 # sets angle > 0 for right turn and < 0 for left turn
155
156 while(True):
157     leftp.pulse_width_percent(50)
158     rightp.pulse_width_percent(50)
159     leftn.pulse_width_percent(50)
160     rightn.pulse_width_percent(50)
161 # when there is no turning command input, travel straight ahead
```

Lines 1-39 establish the motor control scheme for four PWM output lines, which is the basis of track velocity control. Lines 41-48 initialize variables to use for angle data processing. Lines 50-80 define a function **DueData(inputdata)** which segmentizes the angle data for later processing. Lines 82-100 define a function **get_angle(datahex)** which takes a hexadecimal data and returns the angle data on three axis. Lines 102-154 define the core function of making a turn - it compares the angle readings and eliminate the difference. A successful turning will be indicated in the program monitoring console. When the robot needs to travel straight ahead, it executes the command in lines 156-161.By including the file, these defined functions can be directly used in the main program of the smart robot.

A.5. Edge Detection

The edge detector is chosen to be Canny edge detector and the function can be realized with ease. The actual edge detection is programmed as follows.

Listing A.7: Canny edge detection using OpenMV MicroPython

```
1 # Edge detection - By: Jintao - Friday 4/23/2021
2 import sensor, image, time
3
4 sensor.reset()
5 sensor.set_pixformat(sensor.RGB565)
6 sensor.set_framesize(sensor.QVGA)
7 sensor.skip_frames(time = 2000)
8 # initializes the camera
```

```
9
10 clock = time.clock()
11 # initializes the clock
12
13 while(True):
14     clock.tick()
15     img = sensor.snapshot()
16     # captures live image
17     img.rotation_corr(z_rotation = 180)
18     # rotates the image as the camera is installed upside down
19     print(clock.fps())
20     # records frame rate per second
21
22     img.to_grayscale()
23     # turns image to grayscale
24     img.find_edges(image.EDGE_CANNY, threshold=(40, 80))
25     # uses Canny detector to identify edges
26     img.dilate(2)
27     # makes the white spots/pixels more obvious
```

Lines 4-8 initialize the camera for edge detection. Line 15 captures the live image, line 17 rotates the image to correct the upside-down installation of the camera, and lines 22-27 processes the image so that the edge is clearly marked.

A.6. Color Recognition

As is mentioned in previous technical explanation, despite the complexity of theories behind the color recognition algorithm, OpenMV MicroPython provides one-step solution to achieve this function. Below is an example of the straightforward use of **`find_blobs`**.

Listing A.8: Demonstraing the use of find_blobs

```
1  red = (Lr_min, Lr_max, Ar_min, Ar_max, Br_min, Br_max)
2  blue = (Lb_min, Lb_max, Ab_min, Ab_max, Bb_min, Bb_max)
3  yellow = (Ly_min, Ly_max, Ay_min, Ay_max, By_min, By_max)
4  # setting all measured LAB thresholds of different colors
5
6  img = sensor.snapshot()
7  # capturing the current image for color recognition
8
9  red_blobs = img.find_blobs([red])
10 blue_blobs = img.find_blobs([blue])
11 yellow_blobs = img.find_blobs([yellow])
12 # identifying one single color each time
13
14 color_blobs = img.find_blobs([red, blue, yellow])
15 # identifying multiple colors
```

In actual demonstration, the program section used for color recognition is written as follows. Slight modifications are needed before integrating the program into the main program.

Listing A.9: Program section for color recognition

```
1  # Color Recognition - By: Jintao - Thursday 4/22/2021
2  import sensor, image, time
3
4  sensor.reset()
5  sensor.set_pixformat(sensor.RGB565)
6  sensor.set_framesize(sensor.QQVGA)
7  sensor.skip_frames(time = 2000)
8  # initializes the visual module
9
10 clock = time.clock()
11 # starts the timer
12
13 while(True):
14     clock.tick()
15     img = sensor.snapshot()
16     # captures real-time images
```

```
17      img.rotation_corr(z_rotation=180)
18      # rotates the image as the camera is installed upside down
19      '''
20      for already captured images please use the following command:
21      img = image.Image(r"./color/yellow1.jpg", copy_to_fb = True)
22      '''
23      red = (40, 70, 44, 67, -38, -15)
24      blue = (39, 61, 2, 17, -58, -13)
25      yellow = (50, 80, -28, -13, 42, 67)
26      # sets the threshold values for color recognition
27      red_blobs = img.find_blobs([red])
28      blue_blobs = img.find_blobs([blue])
29      yellow_blobs = img.find_blobs([yellow])
30      # identifies one single color each time
```

Lines 4-11 initialize the visual module to the setting that optimizes the performance. Line 15 captures the real-time image from the camera, line 17 rotates the image as it is taken upside down due to the camera installation, lines 23-25 state the threshold values for three target colors, and lines 27-30 complete the color recognition.

A.7. AprilTag Detection

AprilTag recognition in OpenMV Cam is of great simplicity thanks to the efforts of OpenMV developers. Instead of programming each step individually, the team is now able to accomplish the task in one step using the functions defined in

`class image.apriltag`.

The program section realizing AprilTag recognition is written as follows.

Listing A.10: AprilTag recognition using OpenMV MicroPython

```
1  import sensor, image, time, math
2
3  sensor.reset()
4  sensor.set_pixformat(sensor.RGB565)
5  sensor.set_framesize(sensor.QVGA)  # would run out of memory if resolution is bigger...
6  sensor.skip_frames(time = 2000)
7  sensor.set_auto_gain(False)  # must turn this off to prevent image washout...
8  sensor.set_auto_whitebal(False)  # must turn this off to prevent image washout...
9  clock = time.clock() # initializes the clock
10
11 f_x = (2.8 / 3.984) * 160 # find_apriltags defaults to this if not set
12 f_y = (2.8 / 2.952) * 120 # find_apriltags defaults to this if not set
13 c_x = 160 * 0.5 # find_apriltags defaults to this if not set (the image.w * 0.5)
14 c_y = 120 * 0.5 # find_apriltags defaults to this if not set (the image.h * 0.5)
15
16 def rad_to_degrees(radians):
17     return (180 * radians) / math.pi
18 # converts radian to degree
19
20 while(True):
21     clock.tick() # starts the timer
22     img = sensor.snapshot() # capture the image live
23     for tag in img.find_apriltags(): # defaults to TAG36H11 without "families".
24         img.draw_rectangle(tag.rect(), color = (255, 0, 0))
25         img.draw_cross(tag.cx(), tag.cy(), color = (0, 255, 0))
26         # draws a rectangle and cross to assist recognition
27         degrees = rad_to_degrees(tag.rotation())
28         # converts the rotation to degrees
29         x = tag.x_translation()
30         y = tag.y_translation()
31         z = tag.x_translation()
32         # returns the translation in unknown units from the camera in all directions
33         # this helps to locate the tag, but the size of tag matters
34         distance = math.sqrt(math.pow(x, 2) + math.pow(y, 2) + math.pow(z, 2))
35         # calculates the distance
36         print(tag.id(),degrees,distance)
37         # outputs the distance and direction
```

As one could clearly see, there is no advanced libraries required to complete AprilTag recognition. Lines 3-9 initialize the camera. Notice the resolution is identical to that in edege detection, as the image captured needs to be as high-quality as possible. This is different from the setting in color recognition, where higher refresh rate is valued over image clarity.

Lines 11-14 further specifies the camera setting. **f_x** and **f_y** are respectively the x and y focal length of the camera and should be equal to the lens focal length in mm. **c_x** and **c_y** are respectively the image x and y center position in pixels. These parameters further help to identify the AprilTags.

Lines 16-18 define a function that convert the radian angles to degree metric. Since the recognition process produces angles in radians and the following functions desires input values in degrees, the conversion is necessary to the completion of the task.

Lines 20-37 describe the process of completing one round of AprilTag recognition. In this part, line 22 captures the live image for processing, and lines 23-37 searches, identifies and locates any tag of TAG36H11 family which is the default setting of the program.

A.8. Patio 1 Main Program

The idea of program integration is straightforward: putting the function programs together in the desired sequence while removing conflicting variables and organizing program structure. Preliminaries required for all tasks should be included to prevent errors. The main program on Patio 1 is written as follows.

Listing A.11: Patio 1 main program

```
1  # This is the main program of Patio 1 tasks
2  # presneted by Team 40 Shiny Pinky in June 2021
3
4  import sensor, image, time, math
5  import ustruct
6  import car
7  from pid import PID
8  # includes encapsulated files of direction control (PID) algorithm to enable direct use of
       defined fucntions, variables and constants
9  from acc_2 import turn_angle
10 # includes encapsulated files of accelerometer to enable direct use of defined fucntions,
       variables and constants
11 from ultrasonic import distanceMeasure
12 # includes encapsulated files of accelerometer to enable direct use of defined fucntions,
       variables and constants
13
14 theta_pid = PID(p=0.45, i=0.2)
15 # initializes the PI controller and envalues the parameters
16
17 GRAYSCALE_THRESHOLD = [(128, 255)]
18 sensor.reset()                      # resets and initializes the sensor.
19 sensor.set_pixformat(sensor.RGB565) # sets pixel format to RGB565 (or GRAYSCALE)
20 sensor.set_framesize(sensor.QVGA)   # sets frame size to QVGA (320x240)
21 sensor.skip_frames(time = 2000)     # waits for settings take effect.
22 clock = time.clock()                # creates a clock object to track the FPS.
23 # uart = UART(1, 9600)
24
25 ROIS = [
26         (10, 50, 300, 30 ,0.15),
27         (10, 100, 300, 30, 0.15),
28         (10, 150, 300, 30, 0.3),
29         (10, 200, 300, 30, 0.4)
30        ]
31        # defines four regions of interest
32
33 weight_sum = 0 # initializes weight values
34 for r in ROIS: weight_sum += r[4] # defines calculation for weigted
35
36 thresh=[(30, 79, -7, 30, 6, 23)]
37 clock.tick() # initiates timer
38
39 while(True):
40     s = 1
```

```
41      if s == 1:
42          # updates the FPS clock.
43          img = sensor.snapshot()  # takes a picture and return the image
44          img.rotation_corr(z_rotation=180) # rectify the image because it is shot upside down
45          img = img.to_grayscale() # converts to grayscale image
46          img = img.find_edges(image.EDGE_CANNY, threshold=(130, 130))
47          # one-step realization of Canny detection
48          img = img.dilate(1)
49          # makes white spots more present
50
51          centroid_sum = 0
52          # initializes the weighted distance from centroids to the central line
53          for r in ROIS: # for each centroid of ROI
54              blobs = img.find_blobs(GRAYSCALE_THRESHOLD, roi=r[0:4], merge=True)
55              # r[0:4] is roi tuple.
56              # finds all white spots and merge as one (connecting each other with lines)
57
58              # searching for lines in ROI
59              if blobs:
60                  # indentifies the bolb index with the most lines
61                  largest_blob = 0
62                  most_pixels = 0
63                  for i in range(len(blobs)):
64                  # there might be several line clusters in ROI, but identify the larges as the
     target line
65                      if blobs[i].pixels() > most_pixels:
66                          most_pixels = blobs[i].pixels()
67                          # update the value if there is any larger pixel clusters
68                          largest_blob = i
69
70                  img.draw_rectangle(blobs[largest_blob].rect(),color = 0)
71                  # draws a rectangle around the largest pixel cluster
72                  img.draw_cross(blobs[largest_blob].cx(),
73                                 blobs[largest_blob].cy(),color = 0)
74                  # marks the largest pixel cluster with a white cross
75
76                  centroid_sum += blobs[largest_blob].cx() * r[4]
77                  # accumulates the weighted distances to obtain the final result
78
79          center_pos = (centroid_sum / weight_sum) # determines the center of line
80
81          # converts the result above to an angle, using unlinear operations
82          deflection_angle = 0
83          # initializes the angle to turn for the robot
84          deflection_angle = -math.atan((center_pos-160)/60)
85          # obtains the result (in radian)
86          deflection_angle = math.degrees(deflection_angle)
87          # converts result to degrees
88
89          output = theta_pid.get_pid(deflection_angle,1)
90          # tells the robot to turn the target angle
91          print("output")
92          print(output)
93          car.run(40+output, 40-output)
94          print(40+output)
95          print(40-output)
96          # prints the process to monitor the execution
97          if clock.avg() >100000:
98              print(clock.avg())
99              d = distanceMeasure(100)
100             # where d is measured in cm
101             if d < 200:
102                 turn_angle(90)
103                 s = 2
104             # maybe we need to add a bridge detection module
105         # crosses the bridge
106
107     if s == 2:
108     # bridge at the center of the sight, identifies the bridge
109         car.run(40, 40) # drives across the bridge
110         if clock.avg() >150000:
```

```
111             d = distanceMeasure(100)
112             if d < 200:
113                 turn_angle(90)
114                 s = 1
115             # re-enter line tracking to head towards the arch
116     # passes the arch, enf of Patio 1
```

Lines 4-12 include all function programs needed on Patio 1, together with well-programmed extensions from OpenMV MicroPython. Line 14 initializes the controller as PI controller and gives value to the control constants. Lines 17-22 initializes the visual module, and lines 25-36 give the ROIs and threshold values for edge detection, and initializes the variable **weight_sum** which will be used for summing the weighted distances. Lines 40-96 complete line tracking using multiple centroid algorithm and output results for monitoring the execution process. Lines 97-116 describe the process of making two turns and crossing a bridge in between.

A.9. Patio 2 Main Program

The gist of program integration on Patio 2 is essentially the same as Patio 1 system integration. The major difference is to adapt a different OpenMV camera setting that optimizes the color recognition, mainly changing the resolution setting of images, that is

Listing A.12: A different camera setting

```
1 sensor.reset()
2 sensor.set_pixformat(sensor.RGB565)
3 sensor.set_framesize(sensor.QQVGA)
4 sensor.skip_frames(time = 2000)
5 # initializes the visual module
```

Likewise, all desired functions should be included in the sequence of appearance. One thing worth noting is the activation of autoloading ejector and wireless communication. For both actions, only one high level signal output is needed to trigger the modules. The commands are demonstrated as follows, and by directly integrating the sentences to the main program, the functions can be achieved.

Listing A.13: Triggering the ejector and wireless communication

```
1  p = pyb.Pin("P9", pyb.Pin.OUT_PP)   # declares pin for autoloading ejector
2  q = pyb.Pin("P6", pyb.Pin.OUT_PP)   # declares pin for wireless communication
3
4  # now activate the autoloading ejector
5  pyb.delay(10000)
6  p.high() # or p.value(1) to make the pin high (3.3V)
7  pyb.delay(2000)
8  p.low() # or p.value(0) to make the pin low (0V)
9  # triggers the ejector with one single high level signal output
10
11 # now activate the wireless communication module
12 pyb.delay(10000)
13 q.high() # or p.value(1) to make the pin high (3.3V)
14 # triggers HC-12 transceiver to transmit signals to laptop terminal
```

The remaining details are highly similar to Patio 1, so the author would not delve into them again.

B

Project Management

The project will be completed with the collective efforts from all team members, and the need of carefully designed teamwork plan necessitates proper project management. This section clarifies the distribution of workload in the team in initial stages of the project, attaches the bill list recording every purchased component and disbursement, and visualizes the team's current project management in a Gantt chart. In general, the project management of the team has been successful in that teamwork was well-organized and clearly recorded and the plans were flexible to modifications.

B.1. Workload Distribution

The overall workload of the whole project is divided into three classes: *algorithmic programming (AP)*, *hardware design (HD)*, *general management (GM)*. Team members were initially assigned to one single task, but some of them switched to help other members if they own work was completed. The final workload distribution is illustrated in Table B.1. The focuses of each class of work is detailed in the following paragraphs.

Table B.1: Overall task division of the project teamwork

Class	Responsibility	Member Name(s)
AP	Patio 1/2 Integration & Main Programs	Jin Jing, Liu Shiyuan, Dai Yiting
	Direction Control (PID) Algorithm	Liu Yilin
	Line Tracking	Yan Jintao
	Color & AprilTag Detection	Xie Shuangyang
	Wireless Communication	Dai Yiting
HD	System Design & Skeleton Building	Wei Zhengjie
	Ultrasonic Sensor	Liang Xuanyu
	Accelerometer	Liu Yilin
	PCB Design & Construction	Chen Zhuohan
	Autoloading Ejector	Yao Yifei
GM	Project Manager	Liu Shiyuan
	Technical Leader	Wei Zhengjie
	Meeting Recorder	Liang Xuanyu, Xie Shuangyang
	Lab Notebook	Liang Xuanyu
	Artistic Design	Xie Shuangyang
	Report Editing and Typesetting	Xie Shuangyang

Algorithmic Programming (AP) This class of workload mainly concentrates on algorithm design and program writing. Team members responsible for this category of work need to have basic knowledge on both embedded C and Python programming languages. Some modules have their own variant of programming language to enable their functions, so the programming member should research the language libraries and find the correct ways of programming certain functions. Having developed eligible program structures, the programmer runs the program to identify any error, debug and troubleshoot the program. Some parameters need to be fine-tuned to achieve optimal results, and this requires much work from the programming members.

Hardware Design (HD) This category of work mainly concentrates on the physical structure design of the robot. This includes the construction of the chassis and the motor system, the choice of hardware components needed to materialize the desired functions, and the assembly of subsystems. Although no particular programming skills are needed, members who are responsible for this kind of work should have basic knowledge on theoretical and practical analogue circuits. They need to fix any malfunctioning modules, replace broken equipment and enhance the overall stability of the hardware system design.

General Management (GM) This category includes all the miscellaneous nontechnical work, mainly project management measures and project outcome presentation. As part of the project report, the overall project management and task schedules were recorded and every team meeting was documented in a uniform format. The members also kept financial records on the purchased equipment to monitor and control the overall expense of the teamwork project. Another important aspect is the presentation of project outcomes, including a lab notebook, long report and two short videos demonstrating the functionality of the implemented robot. These members should have basic skills on document and video editing to produce high-quality presentations of the project results.

B.2. Financial Record

Since the majority of components are purchased online, the disbursements or cash-out flows are recorded to track the accumulating expense of the project. The final project bill list is provided in Table B.2. Each item and its financial cost are marked in the table for review.

Table B.2: The financial record of the team by the end of the project

Item/Component	Unit price	Number	Budget	Actual cost
LORA	51.5	2	100	103
OpenMV(4 plush 7)	359	1	350	359
MPU6050	44.8	1	50	44.8
Lithium battery	62.54	1	50	62.54
PMMA board	180	1	100	180
Chassis	83	1	100	83
mbed microcontroller	96.71	1	100	96.71
HC-SRF05	5	1	5	5
OLED screen	8	1	5	8
CD4052	1	1	5	1
Copper screw	3.45	10	30	34.5
Dupont line	10	1	10	10
Autoloading ejector	4.46	1	5	4.46
L-shaped connector	5.5	1	5	5.5
Total			915	997.51

The currency used in the financial record is CNY. From the record, it can be clearly seen that the total actual cost is controlled below the maximum supported budget. This indicates an effective budget management throughout the project span.

B.3. Project Planning

The team records every important event of the project work and checks the time planning frequently to monitor the overall progress of teamwork and make adjustments when necessary. The record is visualized as a Gantt chart as shown in Figure B.1. The interdependencies and durations of seperate tasks are clearly illustrated in the chart.

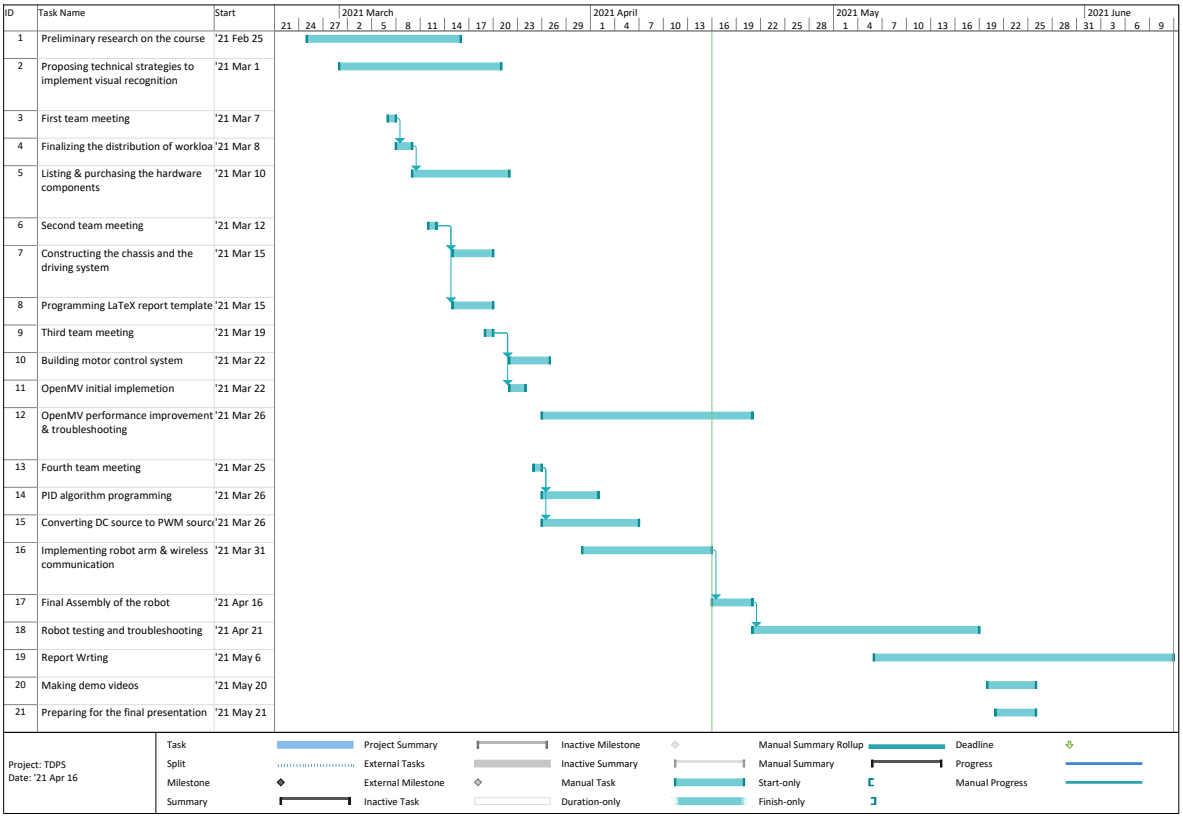


Figure B.1: The overall task planning of the team by the fruition of the project