UNIVERSITY OF TORONTO

INSTITUTE FOR AEROSPACE STUDIES

4925 Dufferin Street, Toronto, Ontario, Canada, M3H 5T6

BALL BALL U

Ball Dispensing Machine

prepared by

Team 50 – Wednesday

Chengnan Shentu (Jimmy) (1003917434)
Chuyi Hou (Sky) (1004197834)
En Xu Li (Thomas) (1004028759)

prepared for

Prof. M.R. Emami

A technical report submitted for
AER201 – Engineering Design

TA: Michael Bazzocchi

April 11, 2019

Chapter 1

Acknowledgements

The success of this project would not be possible without the help and encouragement from the following individuals and businesses.

Firstly, we would like to thank Professor M. Reza Emami for the ongoing recognition and support throughout the term. Particularly, Professor Emami has given incredibly useful and concise lectures on general engineering designs. These lectures at the beginning of the term have helped the team to come up with the alternative designs and converge to the best possible solution. In addition, Professor Emami has given introductory lecture series on the fundamentals of microcontrollers. The successful programming control of the machine would be impossible without these lecture series. Moreover, he has helped our team to debug the system and gave invaluable advice regarding the design to help the team make important design decisions, such as, changing the orientation of the wheels to improve the mobility of the machine.

Secondly, we would like to acknowledge our lovely supervisor, Michael Bazzocchi. He has been supervising the team throughout the term. He met our team once a week to check on the progress of the project and always provided constructive feedback to our machine for future improvements.

Moreover, we would like to also acknowledge two teacher assistants, Michael Bazzocchi and Houman Hakima They both gave us a solid foundation for circuitry and electromechanical system for this project. This machine would not be possible without their hard work teaching us the fundamentals.

In addition, we would like to thank friendly staffs at Myhal Fabrication Facility and the Undergraduate Aerospace Laboratory for offering us tools and equipment, as well as valuable advice. For example, most of our frame is done using laser cutting techniques with multiple iterations. This would not be possible without the continuous and rapid support from Myhal Fabrication Facility staffs.

Furthermore, we would like to acknowledge Creatron Inc. and TaoBao Corp. for providing electronic supplies, such as DC motors, stepper motors, driver boards, and sensors.

Lastly, we would like to thank Engineering Science class of 2021, our fellow classmates. The class has been helping each other with the design framing and system debugging. When many of us encountered the same type of issue, we encourage each other and tackle the problem together. Most importantly, we created a positive working environment for each other.

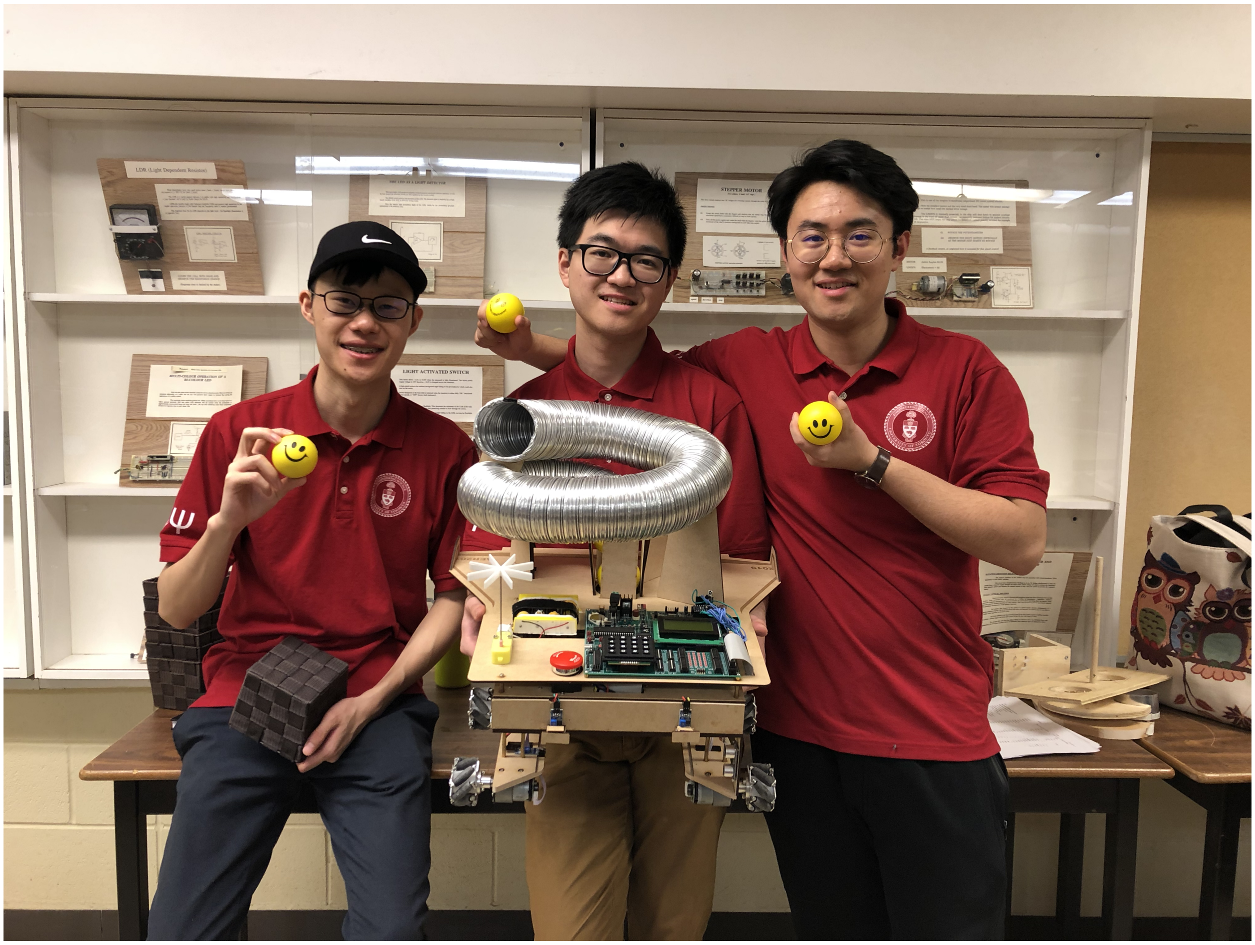


Figure 1.1: Team BALLBALLU Photo

Team 50 | AER201 | PRA0103 | TA: Michael Bazzocchi | Instructor: M. Reza Emami

Chapter 2

Abstract

An engineering R & D company is seeking for a prototype for mobile ball dispensing machines. The machine should be designed to travel along a row of canisters and dispense balls according to the requirements stated as per the RFP[1]. This machine must be fully autonomous, operating on its own with on-board power supply, and is controlled by microcontroller chips on-board.

The project is firstly divided into 3 subsystems, and each team member specializes in one of the subsystems, electromech, circuits and sensors, and microcontroller. After achieving the individual milestones before week 8 of the project, we came together to integrate our design. Our design solution, BALL BALL U, has a dimension of 50 cm x 45 cm x 48 cm and its weight is 4.52 kg, fulfilling the size constraint. During the stages of integration, several major problems have occurred. In particular, the mobility of the robot could not be easily controlled with the help of one microcontroller board. Therefore, we are proud of making the biggest design decision of changing the direction and wheels and adding another Arduino Nano with PD motor control.

From sketch to a fully functional machine, the entire project took 57 days to complete. Despite countless challenges we have faced during integration, the machine is qualified with a huge success.

Contents

1 Acknowledgements **2**

2 Abstract **4**

3 Abbreviation **8**

4 Introduction **9**
4.1 Statement of Need 9
4.2 Background 9
4.3 Motivation 9

5 Project Concept and Design Parameters **11**

6 Perspective **12**
6.1 Theory and History 12
6.2 Background Surveys 12
6.2.1 Literature Survey 12
6.2.2 Market Survey 12
6.2.3 Idea Survey 13

7 Design Overview **14**
7.1 Stakeholders 14
7.2 Objectives 14
7.3 Requirements 16
7.4 Budget 19

8 Detailed Design **20**
8.1 Standard Operating Procedure 20
8.2 Problem Division 20
8.3 Electromechanical Subsystem 20
8.3.1 Mechanical Frame 21
8.3.2 Ball Storage Unit 23
8.3.3 Ball Selection Unit 25
8.3.4 Ball Dispense Unit 26
8.3.5 Mass Budget 28
8.3.6 Mobility Unit 28
8.3.7 Circuits and Microcontroller Housing 30
8.3.8 Sensors Housing 31
8.3.9 Material Selection 31
8.4 Circuits and Sensors 32
8.4.1 Circuitry Design 33
8.4.2 Sensors 34
8.4.3 Motor Driver Boards 35

8.4.4 Power Delivering . . . 37
8.4.5 Future Improvements for this subsystem . . . 38
8.5 Microcontroller . . . 38
8.5.1 Choice of Microcontroller . . . 39
8.5.2 User Interface . . . 41
8.5.3 Permanent Log . . . 44
8.5.4 Communication between PIC and Arduino . . . 45
8.5.5 Mobility . . . 46
8.5.6 Canister Detection . . . 46
8.5.7 Ball Dispensing . . . 47
8.5.8 Pin Assignments . . . 48

9 Integration 50
9.1 Mobility . . . 50
9.1.1 4WD Calibration with 2 PWM Pins from PIC . . . 50
9.1.2 Additional Arduino Nano to Control 4 DC Motors . . . 51
9.1.3 Change Direction of the Motors . . . 51
9.1.4 PID Motor Control . . . 51
9.2 Power Source . . . 51
9.2.1 Voltage Regulator . . . 51
9.2.2 Battery Supply . . . 52
9.2.3 Debugging . . . 52

10 Project Execution 53
10.1 Initial Gantt Chart . . . 53
10.2 Final Gantt Chart . . . 56

11 Conclusions 60
11.1 Bottlenecks . . . 60
11.2 Future Improvements . . . 60

A Datasheets 62
A.1 DC Motor with Encoder Data Sheet . . . 62
A.2 US Sensor HC-SR04 Data Sheet . . . 64
A.3 IR Receiver VS1838 Data Sheet . . . 67
A.4 L298N H-Bridge Driver Board Data Sheet . . . 71
A.5 Nema 42 Stepper Motor Data Sheet . . . 75
A.6 A4988 Stepper Driver Board Data Sheet . . . 77
A.7 IR Proximity Sensor Data Sheet . . . 84

B PIC Microcontroller Main Program 87

C PIC Microcontroller I2C Program 112

D PIC Microcontroller LCD Program 114

E PIC Microcontroller Configuration Bits 117

F Arduino Microcontroller Main Program 120

G PC Interface Main Code 130

H PC Interface Data Structure 132

I Laser Cut Files 134

Chapter 3

Abbreviation

all abbreviations spelled out in alphabetical order

4WD Four-wheel drive

CCW Counterclockwise

CW Clockwise

DC Direct Current

I2C Inter-Integrated Circuit

IR Infrared

MDF Medium Density Fibreboard

MISS Make it super simple

MSB Most Significant Bit

PD Motor Control Proportional-derivative Motor Control

PIC PIC18F4620

PWM Pulse Width Modulate

RFP Request for proposal

RTC Real Time Clock

R&D Research and Development

SD Card Secure Digital Card

UI User Interface

US Ultrasonic

Chapter 4

Introduction

As a second year design team, we have always been passionate to engineering design problems. This section of the report will introduce the design challenge with the background of the project. Our motivation of the design project will also be discussed.

4.1 Statement of Need

An engineering R & D company wants to develop a functional prototype for fully automatic mobile ball dispensing machines. This prototype will need to have as many functionalities, which are specified by the requirements in the RFP, as the machine is required. It is mainly expected to travel along a row of canisters and dispense balls according to the conditions specified [1].

Robots have often been suggested by researches to be suitable for performing dull,dirty and dangerous work in place of human [2]. The case introduced in the RFP is relatively low-skilled, tedious, and time-consuming. In this case, autonomous robots could help reduce human workload [2]. Thus, it allows human workers to focus more on higher level tasks such as decision making.

The objective of our team is to design and build from sketch, and finally develop a fully autonomous and efficient mobile machine that can store and dispense balls based on the conditions already given.

4.2 Background

This project requests second year Engineering Science students to design from sketch, and finally come up with a prototype that should meet the requirements. Due to the nature of this design project, there could be various design solutions that will meet the objectives. Student teams should use their own decision making skills to come up the best solutions they could ever thought. To come up with alternatives, teams would need to do numerous surveys to gather information and idea related to this project.

4.3 Motivation

As the robotic industry developing rapidly, it is important for student engineers, like us, to acquire knowledge and experience related to this field. As the future engineers of the world, we were motivated to go through a difficult journey of developing our very first task-oriented autonomous mobile machine, as our first step into the field of robotics.

Moreover, we were so enthusiastic about this project that we worked day by day planning and developing the machine. It was quite a precious experience for us, leaving no regret that we tried our best. Also, it is what we wanted to accomplish to leave AER201, its instructor, Prof. Emami, and TAs, with unique memories.

Chapter 5

Project Concept and Design Parameters

This section will focus on the how the machine is designed to complete assigned task while meeting the requirements, plus the machine's physical appearances.

The most important part of this design is the machine's mobility while it is designed to go over the canisters. It is hard for a 4WD machine to drive in a perfect straight line. In addition, the space between the wheels/motors, which allows the pass of canisters, is limited. We had to add a functionality which allows the machine to adjust its current position, so that it will avoid any collisions with canisters as it is moving forward. Hence, we chose a special wheel type, mecanum wheels. This wheel type allows the machine to drive in parallel translation directions (i.e. forward, backward, left, and right). Although, it is not perfectly parallel, which is understandable due to manufacture and powering asymmetric existing in the machine, it is good enough for the machine to drive in a way which does not hit any canisters.

Secondly, the ball dispensing mechanism is also essential. The ability of moving in omnidirections acquired from mecanum wheels can improve the ball dispensing accuracy. The machine will move closer to a canister as ball dispensing process starts.

Moreover, canister detection (with its opening and its distance from the start line) is indispensable. This function is achieved by using only one US sensor which is placed at the front of the machine not pointing forward but side way instead. The information gained from the US sensor can tell the machine that a canister is under it. In addition to that, the distance recorded can actually tell the opening of the canister. Based on our testing trials, we found that a reasonable distance but more 18cm than means that the detected canister is opening to the left, while a distance less or equal to 18cm means that the detected canister is opening to the right.

Last but not least, ball detection plus the opening information allows the machine to make accurate dispenses. Detecting yellow balls inside canisters is accomplished by using two digital IR sensors. One IR sensor is placed on left bottom and the other is on the right bottom of the machine. These two IR sensors will scan through the canister as it is passing through. The contrast will generate the detection signals. Based on simple algorithm, the signals can be processed, with the help of the opening information, to determine whether there is a ball inside or not.

The dimension of the design is 50 cm x 45 cm x 48 cm and its weight is 4.52 kg.

Chapter 6

Perspective

6.1 Theory and History

P(Power) = VI Mobile machine requires on-board power supply. Thus, determining power needed for the entire system is important, as well as the choice of on-board power supply.

US sensor distance calculation: Convert timer to distance: timer/58.82 Distance detected is back and forth, so we need to divide the distance by 2 to get the actual distance from the ultrasonic to the object.

6.2 Background Surveys

This section will introduce the major background researches we did during the course of this project. The followings are literature surveys, market surveys, and idea surveys in additional to the ones mentioned in the proposal [2].

6.2.1 Literature Survey

PID Controller uses a feedback loop structure to control any variable such as velocity and temperature. Examples of application include cruise control and indoor temperature maintain. For instance, the cruise control takes the distance from the current vehicle to the one at the front as the input value. It tries to calculate the optimal output of gasoline to the vehicle in order for maintaining the relative distance. It is one of the most accurate and reliable controllers used in industry [3]. This controller type can be very beneficial for our design, since we do need a controller to monitor the current status of the machine giving a set point to the machine and let itself decided how to reach that point.

The 3 parameters of PID controller are kp, ki, and kd, standing for proportional coefficient, integration coefficient, and derivative coefficient. PID Auto Tuning library has been found in Arduino; however, Arduino Nano has not enough memory for setting up 4 PID control with auto turnings [4]. Therefore, our team has followed the protocol online for determining these three parameters [5].

6.2.2 Market Survey

Currently in the market, some existing products have similar functionality with the design requested for this project. For instance, the golf ball dispenser is like a vending machine dispense golf balls to users. The mechanism behind dispensing the golf ball is somewhat similar to the design of the ball dispensing machine described in the RFP [1]. What they have in common is the part where both machines have to select a user-defined number of balls from a pool of balls. These balls need to be separated from the rest by some mechanism. This mechanism has to ensure that the balls don't get stuck or get jammed. Similarly, a gumball machine also has a similar function where it selects one gumball from a box of gumballs. These kinds of mechanisms usually involve with a stepper motor or servo motor to do so.

6.2.3 Idea Survey

For the nature of this project, mobility of the machine is a top focus since it will affect the entire design to a great extend. For example, ball dispensing mechanism can vary a lot if we choose to go over the canister instead of driving around the row of canisters. Thus, we wanted to put more attention to the selection of the paths we wanted the machine to take, along with the choices of wheels. Since our team values the concept of MISS, we wanted to design the machine to go over the canisters, which is our preferred alternative. Meanwhile, we also want our machine to operate in various conditions such as terrain conditions and canisters positions, etc. Thus, we were looking for some special wheels that allow the machine to perform specially.

Mecanum Wheel

A Mecanum wheel comprises a circular surrounded with a plurality of peripheral flanges. A set of rollers are mounted to the tab portions bent at an angle of 45° from said pate []. The greatest benefit from choosing this type of wheel is that it enables the machine to drive in any directions as shown in figure below 6.1

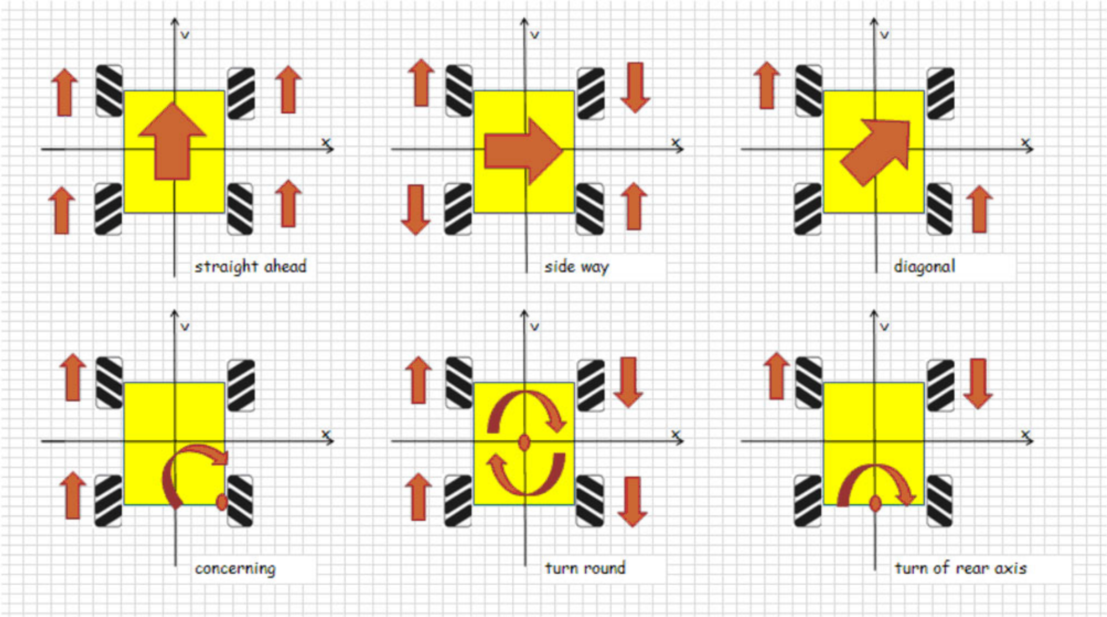


Figure 6.1: Driving Mecanum Wheels in Omnidirections

Chapter 7

Design Overview

The design solution is expected to travel along a line of canisters, trying to recognize the presence of canisters with directions of openings. Meanwhile, the distances from the start line of each canister must be tracked in order to determine if the ball is required to be dispensed. According to the RFP, the machine only dispenses a ball if it's a empty canister with a greater than 30cm distance from the previous one [1]. The stakeholders and the detailed design objectives along with requirements are summarized in the following sections.

7.1 Stakeholders

Engineering R&D Company gives RFP for the ball dispensing machine that the design team needs to develop.

AER201 Course Instructor: Prof. M.R. Emami wrote the RFP and is responsible for guiding the design team going through the entire process.

Design Team is the main active stakeholder who are responsible for every stage throughout this project.

7.2 Objectives

The following objectives stated in the RFP [] are valued by our team:

- Usability/Accessibility
  - User can load/unload balls conveniently.
  - Setting up the machine before each operation should be done simply and must not exceed 2 min.
  - The LCD UI should be straight forward, can be easily used by the user without any confusion.
  - Balls should not be stuck leading to a emergency stop during the operation.
- Manufacturability
  - Components and materials are easy to obtain.
  - Building phase of the machine should be easy and quick.
- Accuracy
  - The machine should contain a RTC telling the user more time details of the operations, such as when is the operation been done, and what time is it right now.
  - Distance of each canister from the start line should be recorded correctly within 10cm.

  - Dispensing only one ball at a time if applicable.
  - Emptiness/fullness of canisters should be detected correctly.
  - Obtaining correct information about the openings of the canisters, so that the machine can make accurate dispenses.
  - The machine should notify the user using any means of signal (e.g. sound, light, motion, etc) whenever it detects a new canister in range.

- Safety
  - Canisters must not be significantly moved or stroked by the machine during operations.
  - The machine must be equipped with an emergency stop button which allows the machine to stop immediately after the button is pressed.
  - The machine must not act dangerously or hazardously to any of the surroundings.

- Compactness
  - The entire prototype must fit within an envelope of 50cm*50cm*50cm during operations.
  - The weight of the machine should be less than 8 kg.

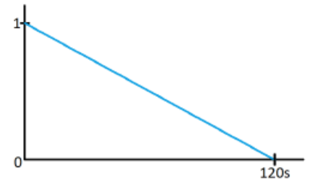
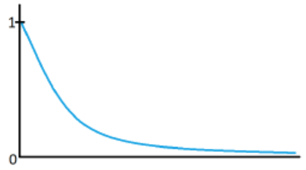
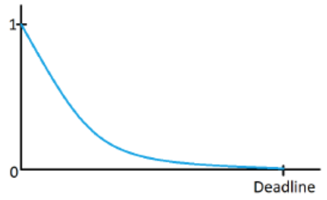
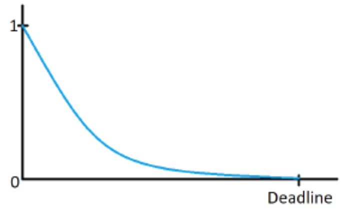
- Efficiency
  - The machine should work efficiently taking as less time as possible for each operation. The operation time is limited to less than 3 min.
  - The machine should consume as few power as possible
  - Ball dispensing mechanism should be done efficiently in terms of time and power.

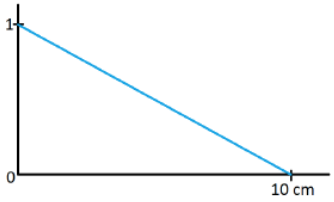
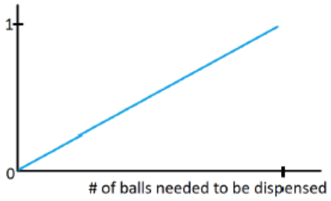
- Cost
  - The total cost of building the machine must not exceed 230.00 CAD.

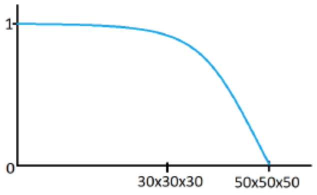
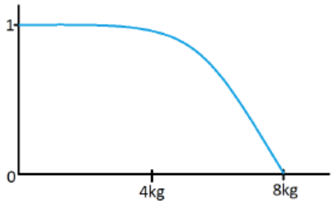
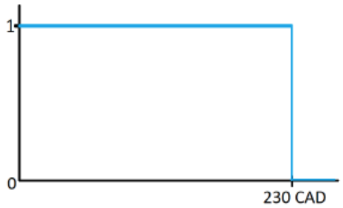
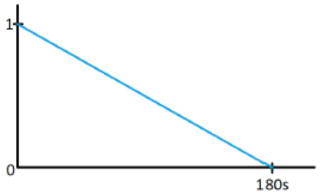
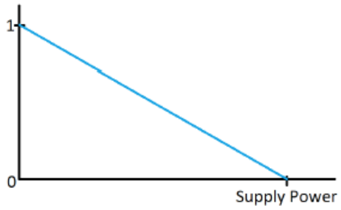
- Durability
  - The machine should be able to run for multiple operations with little maintenance or repair.
  - The machine can withstand heavier load of balls.
  - Materials such as paper and tape must not be used in the machine for construction purposes.

7.3 Requirements

Table 7.1: Requirements

Objective	**Parameter**	**Unit**	**Constraint**	**Utility Function**	**Justification**
Usability	Preparation time prior to operation	Seconds	Must be less than 120 s		Less prepare time would make the operation much easier.
	User could easily understand the use of LCD along with keypad	Qualitative	N/A	Binary “Yes” means high utility “No” means low utility	Easy understanding of the LCD display helps the user to operate the machine.
	Number of buttons pressed when requesting for information	Number of buttons			Fewer number of buttons that user needs to press, easier the operation it takes.
Manufacturability	Number of days between ordering row material and arrival	Number of days			To minimize the time cost of the materials for building the machine.
	Number of days to build the functional machine	Number of days			To minimize the time cost of building and assembling the machine
	Number of days to rebuild the machine	Number of days			To minimize the number of days for rebuilding the machine

Accuracy	Error in distance measured for each canister	Centimeters	The error must be less than 10 cm		Less errors would result in a better distance determination.
	Number of times of accurate ball dispensing actions (include ball/detection canister)	Number of times	At least 3 accurate dispensing per run		More accurate operations would increase the score linearly.
	Difference between clock in robot with real world clock	seconds	The error must be less than 2 seconds		Less errors in the operation time determination would indicate a more reliable and accurate machine.
Safety	Response time of emergency stop button	seconds	Must terminate as soon as the button is pressed		The response time of emergency stop button must be instant.
	Machine looks rigid with as little sharp edges as possible	Qualitative			Machines with sharp edges would hurt users.
	Machine's circuitry connections should have no exposed conductive wires	Qualitative			Exposed conductive wires have a high chance of shorting the circuits.

Compactness	Dimension of physical envelope of the machine during operation	Cubic centimetre	Must be less than 50*50*50		Smaller dimension would make the machine more portable.
	Weight of the machine	kilogram	Must be less than 8 kg		The lighter the machine, the easier for the user to carry.
Cost	Total cost of the machine	CAD	Must be less than $230 CAD		Lower costs would make the machine more marketable to consumers.
Efficiency	Operation time	seconds	Must be less than 180 s		Faster operation is always preferred due to time costs.
	Power consumption	watts			Power consumption should be minimized, wasting no electricity.

7.4 Budget

Table 7.2: Budget Table

Budget					
			1 RMB = 0.1943 CAD		
ITEM	**Supplier**	**Per Unit Cost CAD**	**Quantity**	**COST IN CAD**	**COST IN RMB**
Mini Balance DC Motor with Encoder	TaoBao	8.14	4	32.56	
Mini Balance Mecanum Wheels	TaoBao	10.75	4	43	
Mini Balance Motor Support	TaoBao	0.46	4	1.84	
Arduino Nano Board	TaoBao	2.1574	1	2.1574	12.88
PIC DevBugger Board	Project Kit	55	1	55	
LCD Display + Keypad	Project Kit	8	1	8	
Real Time Clock Chip + Coin Battery	Project Kit	4	1	4	
L298N Driver Board	TaoBao	0.75375	2	1.5075	4.5
A4988 Stepper Driver Board	TaoBao	0.641525	1	0.641525	3.83
Stepper Motor Screw Rod with Support	TaoBao	2.76375	1	2.76375	16.5
Nema 42 Stepper Motor with Support	TaoBao	4.929525	1	4.929525	29.43
US Sensor HC-SR04	TaoBao	0.293125	1	0.293125	1.75
IR Receiver + Remote Control Pack	TaoBao	0.6365	1	0.6365	3.8
IR Sensor EK1254x5C	TaoBao	0.373525	4	1.4941	2.23
3.7V 18650 Battery 4 pack	Amazon	15.99	1	15.99	
Emergency Switch	Creatron	6.25	1	6.25	
Frame - Laser cut in 1/8 in MDF	Myhal	1.33	4	5.32	
Frame - Time cost of laser cutting	Myhal	5	1	5	
Screw and Nut (M2 M3 M4 Assortment Kit 1080pc)	Amazon	29.95	0.1	2.995	
Brass Standoff (M2 M3 M4 Aassorment Kit 360 pc)	Amazon	26.99	0.1	2.699	
Flexible Aluminum Duct 3 inch x8 foot	Home Depot	8.67	1	8.67	
3D Printed Battery Pack	Myhal	3.96	1	3.96	
			Total CAD	**209.71**	
			Constraint	**230.00**	

Chapter 8

Detailed Design

8.1 Standard Operating Procedure

For each operation, the machine should be placed along the row of canisters roughly centred at the same line. After positioning the machine, make sure the emergency stop button is not pressed down. LCD will light up showing "Press * to start" with real time clock underneath the text. Operation will start after a long press on the "*" button. During the operation, the machine will complete its dispensing task fully autonomously according to the requirements stated in the RFP. It will adjust its horizontal position if it detects a canister which is too far away from the centre line. After the machine dispense all 10 balls, or detected 10 canisters, or went 4 meters, the machine will first move forward a short distance and then shift to its right and run back to the start line. In total of five previous operations will be recorded on the PIC through EEPROM.

8.2 Problem Division

Problem was divided into three subsystems. In particular, Jimmy is responsible for the electromechanical part of the project where he has designed and built the structural frame of the machine and installed sensors and actuators onto the robot. Sky worked on choosing the sensors, designing and building electrical circuits to connect all the sensors with the microcontroller. Thomas designed the UI of the machine and programmed the microcontroller to process input signals from the sensors and output appropriate actions accordingly. Three subsystems would have to work together to accomplish the final machine.

In this chapter, we will discuss each subsystem in detail at the final stage of the design, also including the changes from what we planned from the proposal.

8.3 Electromechanical Subsystem

The electromechanical subsystem refers to the combination of electrical and mechanical structures that provide the prototype with required functionalities. Specifically, the electromechanical subsystem is responsible for the storage and dispensing of the balls, mobility of the prototype, and the housing for all circuits, sensor, and microcontroller components. We applied the design principle of MISS (make it super simple) to ensure reliability and robustness. In this section, the detailed design of each electromechanical components are explained, along with supporting materials, and an overview of the design process.

The machine is divided into the following units: Mechanical Frame, Ball storage unit, Ball Selection Unit, Ball Dispense Unit, Mobility Unit, and Housing for Circuits and Microcontroller. Their problem assessment, detailed design, changes made from proposal, justifications, and potential improvements are explained for each functional unit.

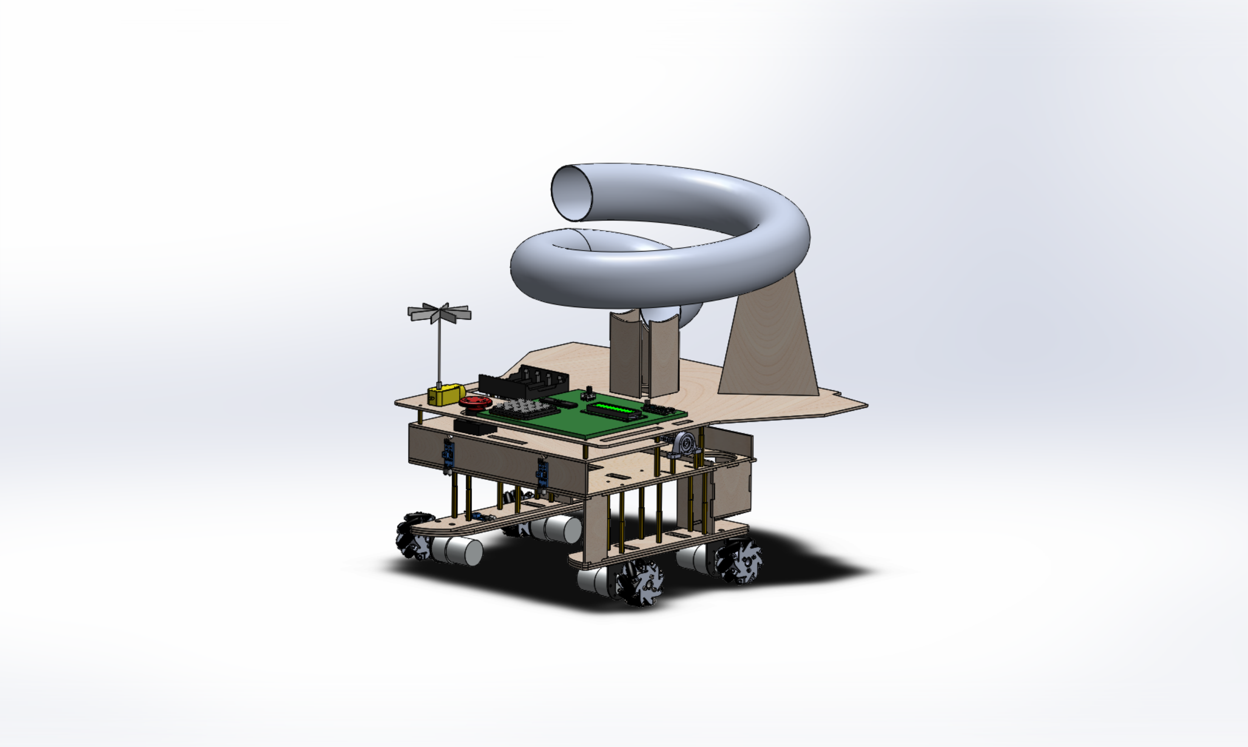


Figure 8.1: Overview of Final Design

8.3.1 Mechanical Frame

Assessment of the Problem

The main electromechanical objectives of the frame are as follows:

- Provide strength to support functional units, especially for Mecanum wheels which have high precision requirements
- Ensure clearance for passing over canisters
- Provide clearance in front to avoid collisions when approaching canisters
- Provide openings for wires and accessibility to the Circuits Unit
- Maximize Strength-to-Weight ratio
- Remain Stable during Motion
- Provide easily accessible space for the housing of sensory units for installing and calibrating

Detailed Design

The frame is consisted of three plates. The first one is a 30x30 cm main plate that supports the Ball Selection Unit, the Ball Storage Unit, and Circuits Unit from below. There are two circular openings with a 63 millimeter diameter for balls to drop through the plate, and proceed to the Ball Dispense Unit. Two square openings are present for the placements of wires. One big center opening ensures accessibility to the Circuits Unit from below. There are also multiple holes allocated on the plate for the use of screws. A dent is designed right below the ball storage unit to prevent balls from uncontrolled rolling before they are dispensed.

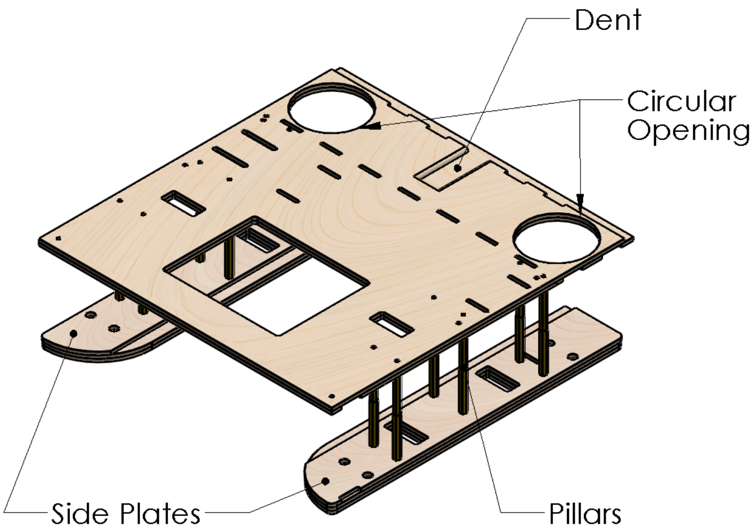


Figure 8.2: Frame Design

Two side plates are located 80 millimeters below the main plate. There are precise holes for the alignments of the four wheels besides holding them in place. Two cuts are made to the forward-facing inner corners of these plates to avoid collision when approaching canisters. There is a 180 millimeter distance between the two plates, which gives a 30 millimeter clearance on each side when passing over a canister. Openings are also made for sensors, Ball Dispensing Units, and wires connecting these components. These two plates are connected to the main plate by 12 sets of brass pillars, each consists of two 40 millimeter M3 standoffs. These pillars provides the frame with enough vertical strength, as well as resistance for rotation during motion.

Clearance for Height

To provide the clearance needed for passing over the canisters, the following height calculation is made to verify the design.

$$49mm + 3 * 3.175mm + 80mm = 138.525mm$$

with height contribution from moto, side plate, and pillars respectively. This clearance is safe for canisters which have a 135 millimeter height maximum.

Changes from Proposal

The middle plate as shown in figure 8.3 was designed to provide more flexibility to the placement of sensors. However, the bottom plate is sufficient for sensors.Thus, the middle plate is removed to further simply the structure.

More openings are added through the process to hold various functional components mentioned above, and make the Circuits Unit above more accessible.

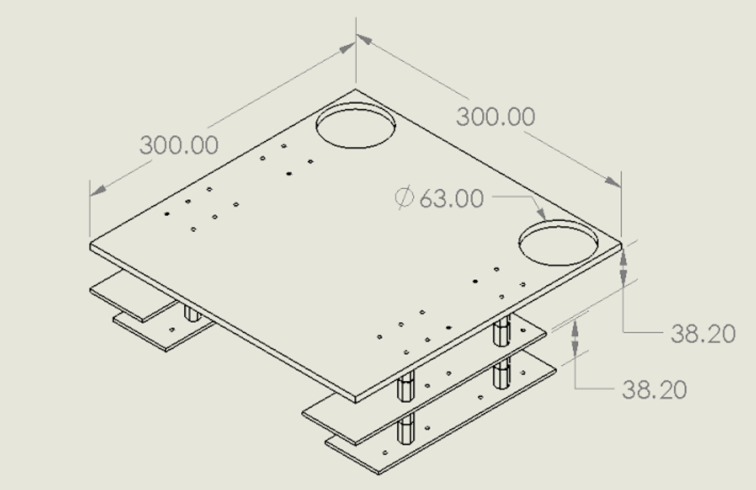


Figure 8.3: Frame Design from Proposal

8.3.2 Ball Storage Unit

Assessment of the Problem

The main objectives of the Ball Storage Unit are as follows:

- Provide space for 20 balls
- Prevent balls from falling from the machine when moving
- Reliably feed balls to the Ball Selection Unit
- Prevent balls from jamming
- Keep weight low
- Allow accessible loading and unloading of the balls

Detailed Design

An extendable 3 inch diameter aluminum duct with a maximum length of 8 feet (2438 millimeter) is used as the Ball Storage Unit along with a few pieces for structural support.The output end is located above the Ball Selection Unit with a clearance of 60 millimeters. This clearance allows for easy access for unloading balls, and the other end allows for loading.The all-around tube is able to securely store the balls, and its flexibility allows for gently curved path, which avoids balls from jamming within the storage unit.

Length Verification

To store 20 balls in the machine, the tube must fit 19 balls while 1 is in the clearance space. The minimum length required for 19 balls with 63 millimeter diameter is $19 * 63mm = 1197mm$, which is well under the maximum length such tube can extend to.

Changes from Proposal

The design has changed completely since the proposal. This is due to the fact that the balls can easily get jammed in a rigid container like the one shown in figure 8.5. Preventing jamming without greatly modifying the design would require adding an actuator or adding some rapid motion to the system, which would add greatly to the complexity of the system. To align with the MISS design principle, the original design is replaced with the tube.

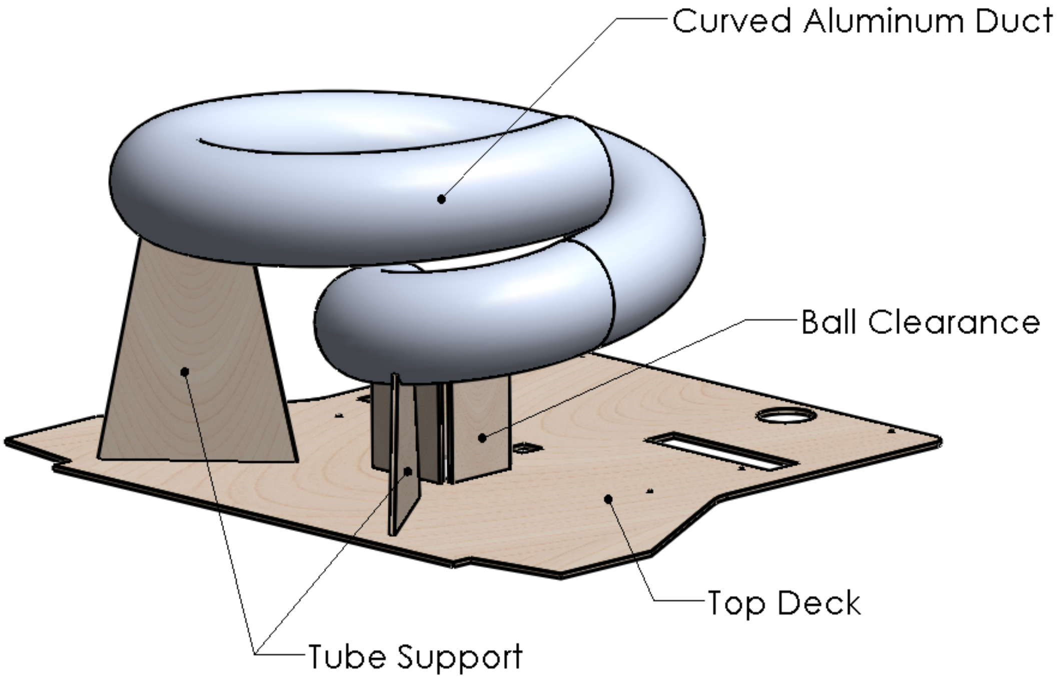


Figure 8.4: Ball Storage Unit Detailed Design

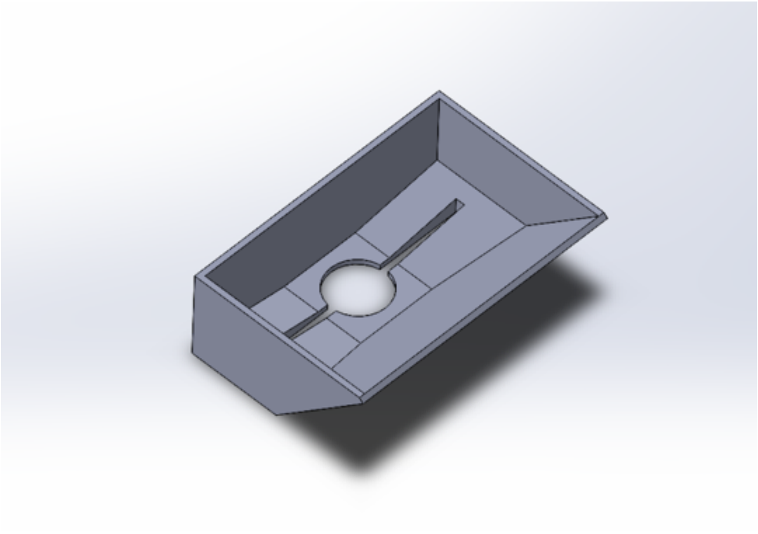


Figure 8.5: Ball Storage Unit Design from Proposal

Potential Improvements

As an off-the-shelf item, the Ball Storage Unit has a great potential for improvements. One of the most important factor is the occupied volume. There is a lot of extra space unused because the tube's diameter is a lot larger than the balls. A narrower tube can be made as a replacement unit, and will reduce the space occupancy significantly.

8.3.3 Ball Selection Unit

Assessment of the Problem

The main objectives of the Ball Selection Unit are as follows:

- Receive one ball from the Ball Storage Unit at a time
- Move that ball towards either left or right part of the Ball Dispensing Unit
- Avoid jamming
- Prevent uncontrolled movement of the taken ball
- Prevent other balls from entering this unit

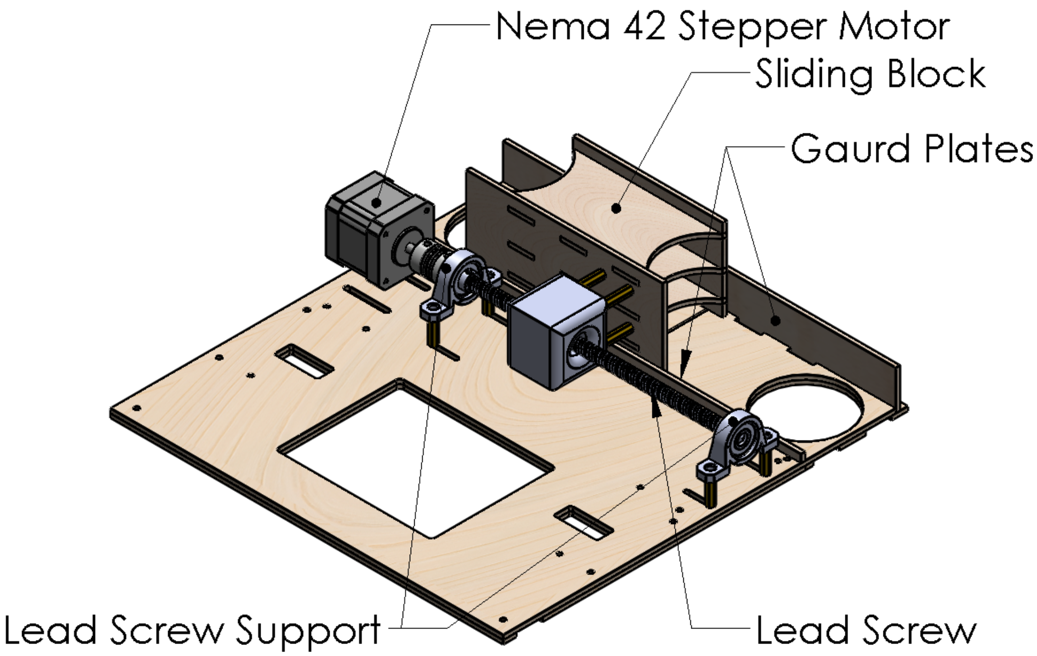


Figure 8.6: Ball Selection Unit Detailed Design

Detailed Design

The Ball Selection Unit is a block that slides left and right between the two circular openings on the main plate. If the unit is instructed to push one ball to the right, it would first shift all the way to the left, letting one ball drop down on the dent on the main plate. Then, it would start to shift back and push the ball to the right. Other balls would be blocked above the unit by either the ball or the sliding block. After the ball is pushed to the opening, the sliding block will return to its default position, which is the centre, and wait for the next run.

The motion of this unit is powered by a Nema 42 stepper motor and a lead screw, which is widely used as the method of actuation for 3-D printers. Such method of actuation has the advantage of being very precise, and a stepper motor is easy to control with a stepper driver board installed.

Changes from Proposal

The concept of the unit remains unchanged. The proposed rack and pinion system is exchanged to a lead screw unit for easier and more precise control. A few other details are also adjusted to accommodate the change of the Ball Storage Unit. Since the aluminum duct significantly reduces jamming, the slide for shaking balls is removed.

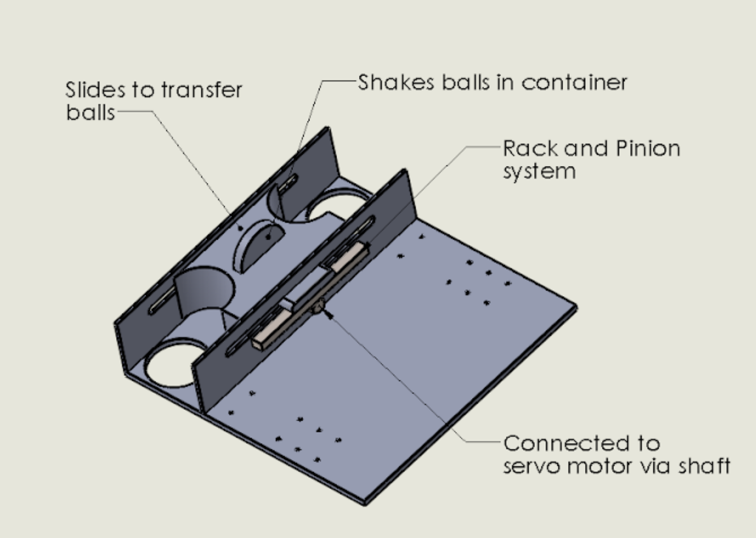


Figure 8.7: Ball Selection Unit Design from Proposal

Potential Improvements

The Ball Selecting Unit demonstrated reliable performance during public demo. However, a few rollers can be added to the sliding block to reduce its friction with structures around. Reduced friction can further enhance the performance of the unit, and also reduce noise.

8.3.4 Ball Dispense Unit

Assessment of Problem

The main objectives of the Ball Dispense Unit is as follows:

- Receive one ball from the Ball Selection Unit
- Transfer the ball down towards the opening of the canister
- Let the ball to be fully inside the canisters
- Consistent performance

Detailed Design

The Ball Dispensing Unit has two sub units on both sides of the machine towards the back. They are essentially two "boxes" with tilted bottom plates to guide the balls towards the canisters. They are fitted between the main plate and the side plates of the mechanical frame through designed openings. This adds to the overall strength of the structure, which increases reliability.

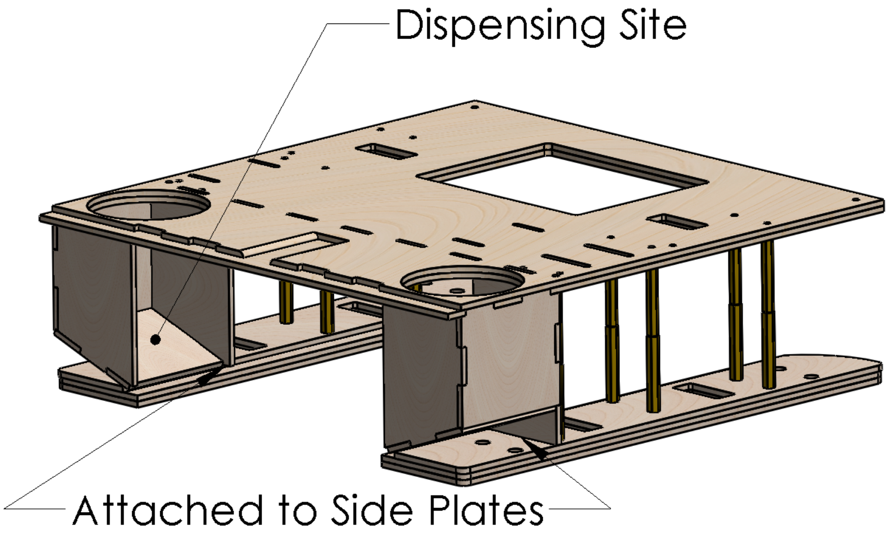


Figure 8.8: Ball Dispense Unit Detailed Design

Changes from Proposal

While the main idea behind the unit remained unchanged, the structure is made with MDF board instead of metal tubes. Laser cut MDF provides greater flexibility over the shape. Purchased metal tubes only have limited degrees of freedom, and generally occupies larger volume compared to the current solution.

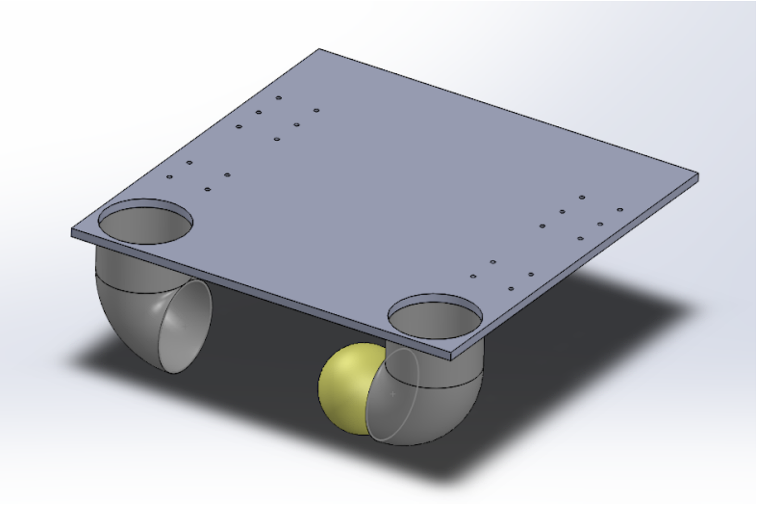


Figure 8.9: Ball Dispense Unit Design from Proposal

8.3.5 Mass Budget

The overall mass of the machine is constrained to be below 8 kilogram. The mass calculation is shown in Table 8.1. The total mass is expected to be 4.5 kilogram.

Table 8.1: Mass Budget Calculation

Components	Individual Weight (g)	Amount	Combined Weight (g)
12V DC Motor	145	4	580
Mecanum Wheels	87.5	4	350
PIC Devbugger Board	295	1	295
Aluminum Duct	360	1	260
Stepper Motor	250	1	250
Battery Unit	200	1	200
Lead Screw - 230mm	180	1	180
DC TT Motor	30	1	30
Emergency Stop Button	25	1	25
MDF Boards	-	-	1250*
Standoffs	-	-	450*
Sensors and other Electrical Components	-	-	450*
Screws and Nuts	-	-	200*
Total			4520

*Include multiple components

8.3.6 Mobility Unit

Assessment of the Problem

The main objectives we expect from the mobility unit is as follows:

- Forward and Backward motion in straight lines
- Able to shift to the left or right without significant tilt
- Able to support the machine during operation

Detailed Design

The Mobility Unit is composed of four Mecanum wheels each powered by a 12 V DC motor.They are strictly aligned to ensure expected performance.

Torque Calculation

As calculated in the Mass Budget section, the machine has a weight of 4.520 kilogram. This means each wheel experience a normal force of 11.0853 N. Taking the kinetic coefficient of friction as 0.5, a friction force of 5.504 N, which require a torque of 1.83 N/cm. The purchased DC motor have a torque of 9.8 N/cm (Appendix A.1). The selection of motor leaves a very big safety margin, which is necessary for omni-wheels since part of the force generated by the wheels get cancel out during operation.

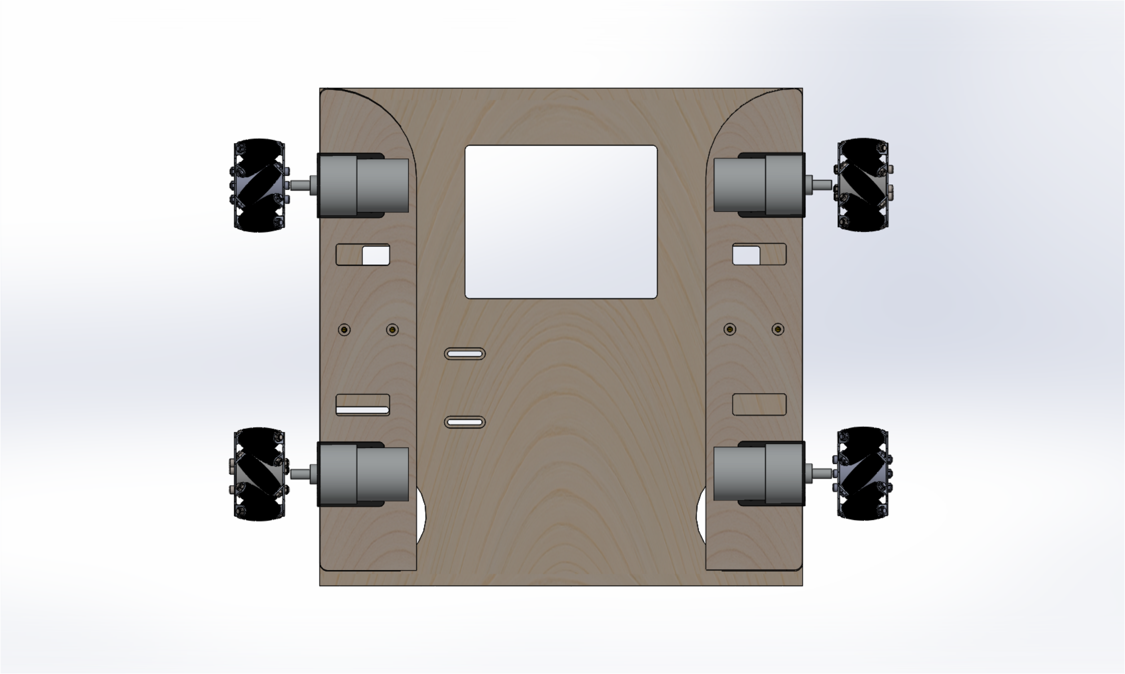


Figure 8.10: Bottom View of the Mobility Unit

Changes from Proposal

The wheels are originally designed in a different orientation to maximize compactness. However, that orientation is found to be ineffective during the integration process, so the new orientation is used for the machine. More details regarding the integration process around the Mobility Unit can be found in section 9.2.

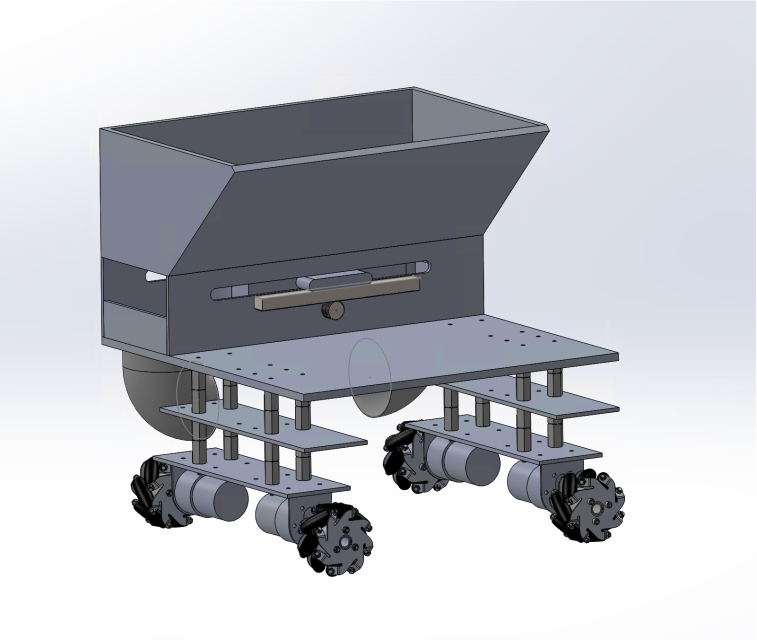


Figure 8.11: Wheel Orientation from Proposal

8.3.7 Circuits and Microcontroller Housing

As shown in figure 8.12 and figure 8.13, most of the circuits components are integrated on to the deck above the main plate, and the microcontroller component is integrated on the top deck for easy access.

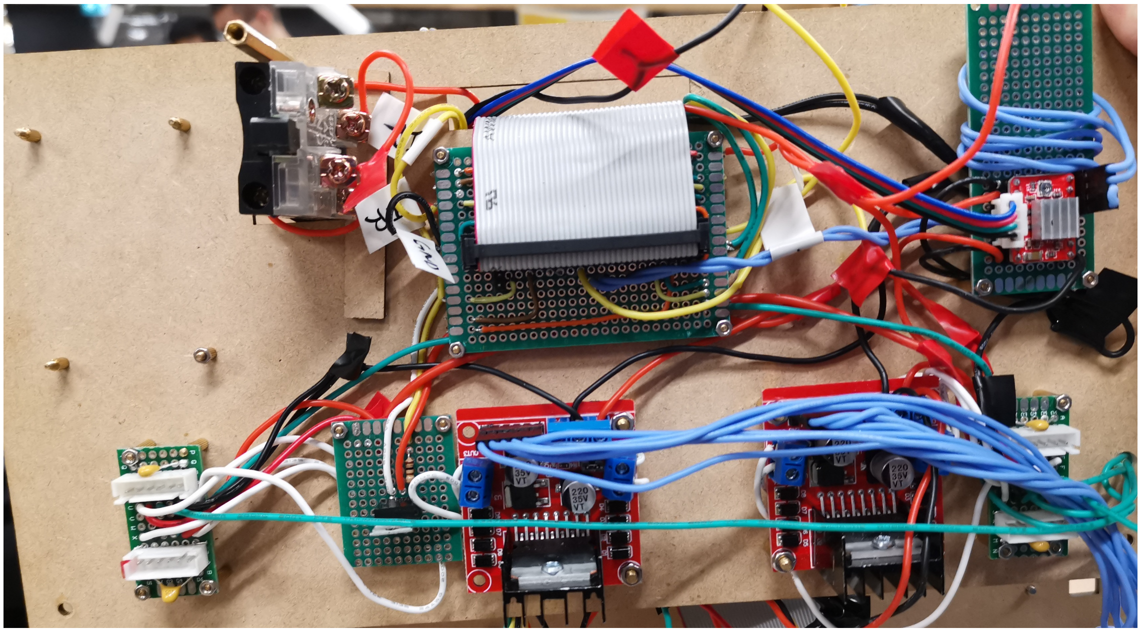


Figure 8.12: Circuit Housing

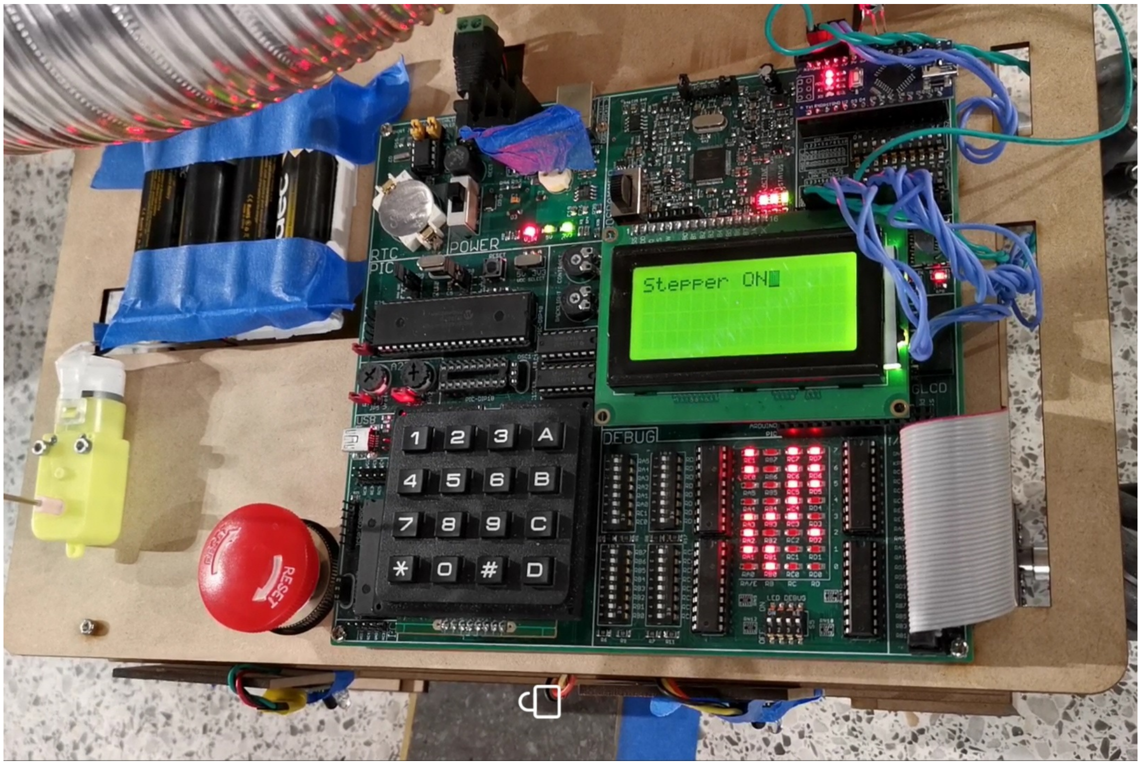


Figure 8.13: Microcontroller Housing

8.3.8 Sensors Housing

There are 4 IR sensors and 1 US sensor in total on this machine, and they are located at different parts of the machine to achieve their functionalities. Figure 8.14 demonstrate their specific locations on the machine.

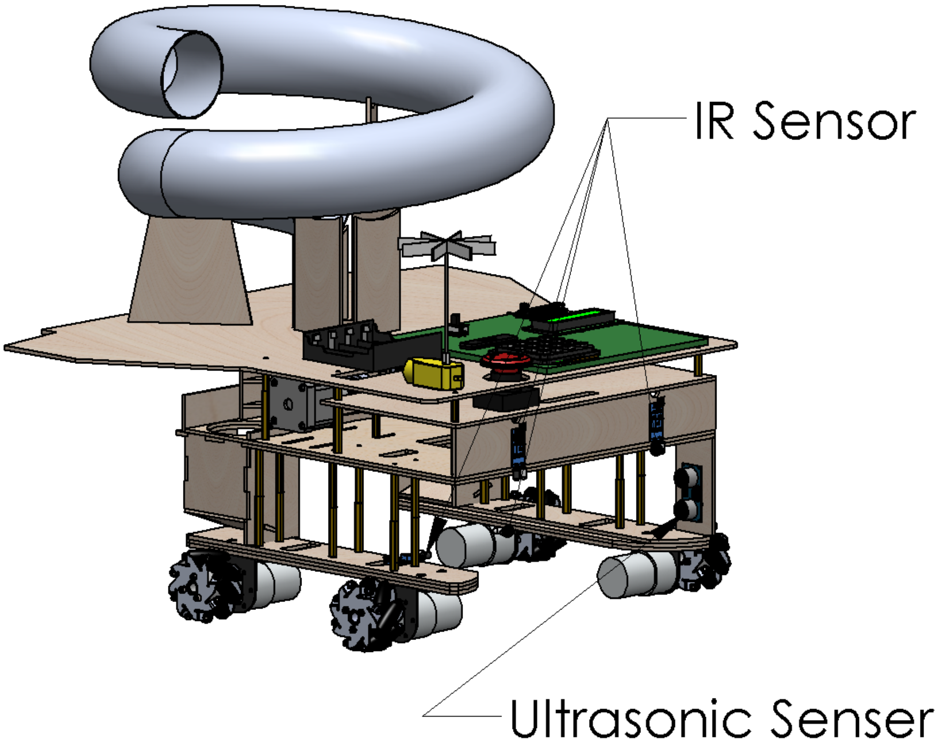


Figure 8.14: Sensor Housing on the Electromechanical Frame

8.3.9 Material Selection

All the plates in the electromechanical system are laser cut with 1/8 inch MDF (Medium Density Fibreboard). Firstly, MDF is produced with recycled wood, which reduces robot's ecological footprint. Secondly, MDF is an economic choice in terms of availability and cost. Last but least, MDF have the advantages of uniform property and more resistant to heat and humidity compared to wood.

8.4 Circuits and Sensors

Assessment of the Problems Circuits connect all electronic components together, from separate circuit boards, sensors, and actuators to a whole circuitry, as well as delivering sufficient power to ensure that the machine works sustainably at least during its operating time. Sensors are used to grant the machine its sensibility which enables it to detect canisters and balls. In addition, circuits also need to provide passages for signal transmissions, both sending signals from microcontroller end and giving feedback from sensors to microcontroller. Basically, the above describes the responsibility along with the problem that the circuit member needs to solve.

In the following subsections, a complete circuit schematic is shown below, and every major circuit components will be described in detail. The solution to the problems will come along within the subsections.

8.4.1 Circuitry Design

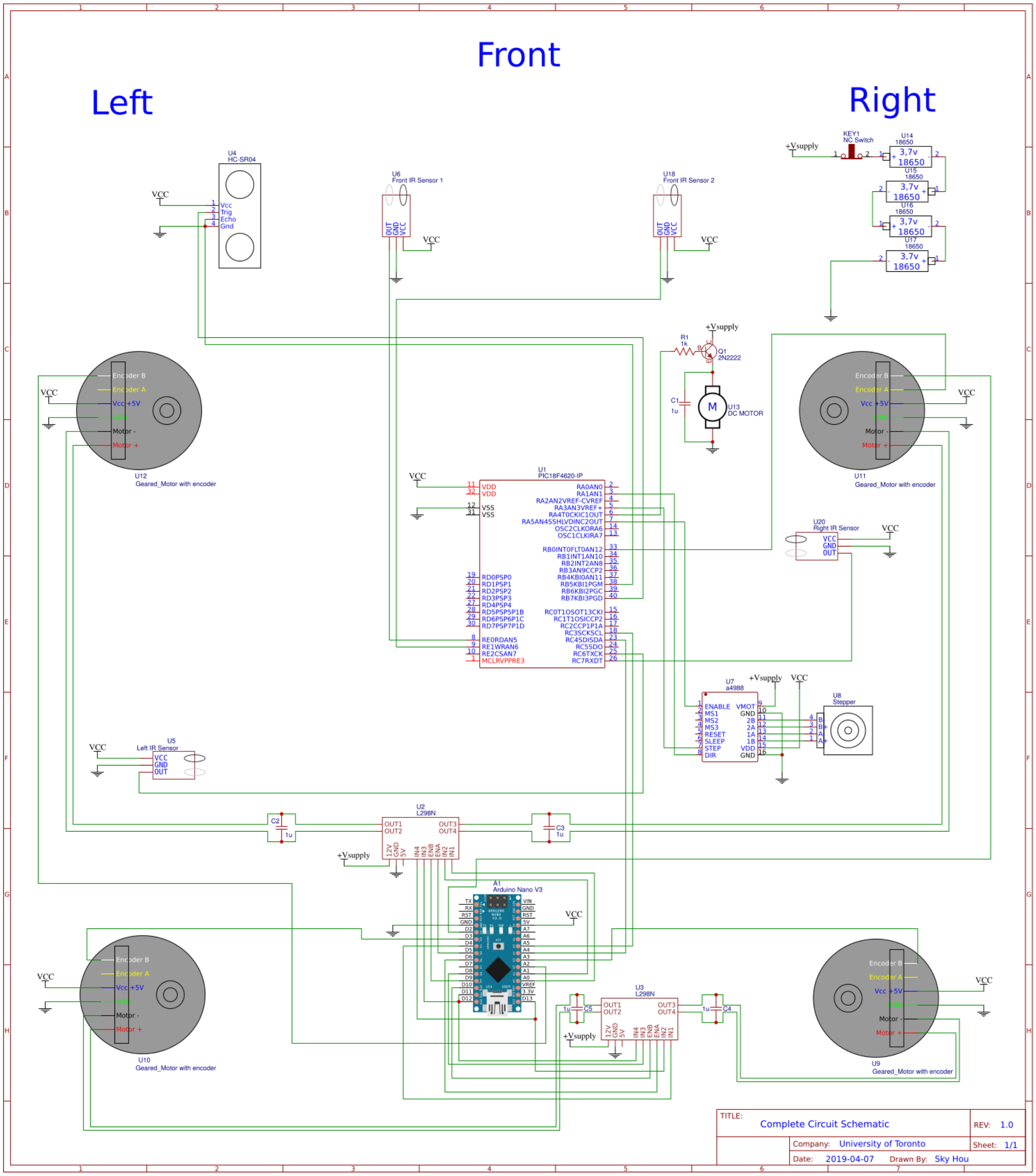


Figure 8.15: complete circuit schematic

The above circuit schematic shows all the connections of the circuits of the machine. All V_{cc} symbols are representing the same point, as well as ground and V_{supply} symbols. In addition, the components arrangement of this schematic approximately matches the placements of each components inside the machine. This illustration can help distinguish IR sensors and motors, also assist the visual understanding of the entire circuitry. One power supply supplies entire circuitry including the microcontrollers, PIC and Arduino. An emergency stop button is placed right after the positive side of the power supply. All grounds of each components are connected together. V_{cc} is supplied directly from the PIC board.

Components controlled by each microcontroller is listed below:

- PIC: IR sensors, Ultrasonic sensor, stepper motor signal, one encoder
- Arduino: Motors, encoders

Note: Pin assignments will be shown in microcontroller section.

8.4.2 Sensors

Ultrasonic Sensor (HC-SR04) One is placed near the front of the machine and is used to detect canisters including a distance value plus their opening directions (left/right). The one we used requires inputs from PIC (Trig) sending waves out, and gives feedback to PIC (Echo) by receiving the reflected waves. We programmed the US sensor so that it will be triggered after each echo. The Distance is proportional to the pulse width of the signal and can be calculated using equation[]. Fortunately, noises were not an issue for the US sensor, thus, there would not be signal filtering for this part.

Infrared Sensors (EK1254x5C) One is placed near the front bottom of the machine and another one near the middle bottom of the machine. This positioning synergies with the ball dispensing position well since it allows the machine to scan the entire canister before it dispenses balls. There is one IR emitter and one IR receiver on the sensor. Basically, it works similar to the US sensor for proximity purposes. Their job is simply detecting yellow balls inside canisters. Feedback signal will be generated if it senses a contrast inside the brown canister. In addition to these two, there are two more placed at the front of the machine both pointing downwards. These two additional IR sensors are responsible for avoiding collisions with the canister since the machine cannot drive in a perfect straight line. If either of these IR detects a closed-up canister, the machine will shift horizontally left or right accordingly. Noise cancelling is done on the microcontroller end as it is not the noise created in IR itself but the detecting conditions.

Hall Effect Encoder Each DC motor has two encoder channels, A and B. From reading both channels, one could tell the direction of the rotation of the motor from the phase shift between two signals and compute the number of turns from the number of pulses [6]. Due to the insufficient pins that Arduino Nano provides, we have decided to take only one channel from each motor, ignoring the direction of rotation because the machine will mostly be running in one direction. There are a couple of reasons for using encoders:

- Distance computation: refer to the equation(distance = encoder count/encoder resolution*pi*wheel diameter)
- PD motor control computes PWM output values from comparing the four encoder inputs (see section something) for detailed illustration of PD motor control)

Changes from Proposal We used only one ultrasonic sensor whereas the proposal said that we wanted to implement two. The reason behind this is that we could actually use just one US sensor to detect the openings of the canisters simply based on the distance acquired. Then, we realized that two US sensor are redundant. Thus, using only one US sensor is a better alternative.

8.4.3 Motor Driver Boards

L298N H-Bridge Driver Board Two are used for controlling four motors, the mobility of the machine. There are six signal input pins, which can control two DC motors separately (three pins for each motor), on each board. Out of the three pins, two are responsible for directions of rotation plus one responsible for PWM control. These driver boards are directly supplied by the power supply.

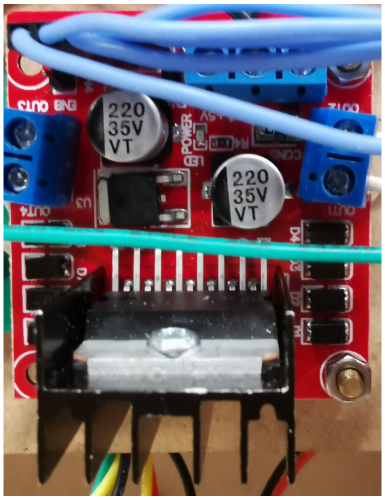


Figure 8.16: L298N H-Bridge Driver Board

A4988 Stepper Driver Board Only one is required for our machine. This special driver board allows us to minimize the pin needed for a stepper motor, as well as making the microcontroller end less complicate. It required at least two input pins, direction and step, to run a stepper. But we also added another pin which tells the board to enable/disable the stepper, so that the stepper would not ”hold” its position when it is unnecessary. This can save some power as the machine operates.

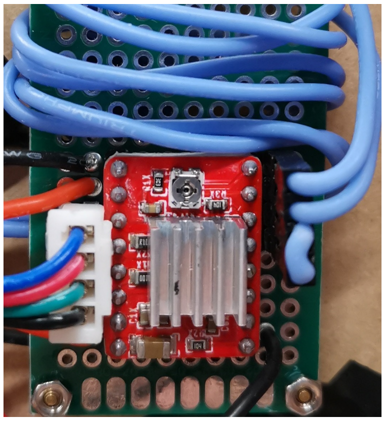


Figure 8.17: A4988 Stepper Motor Driver Board

NPN Transistor One is used as a switch to control a DC motor in one direction rotation. When signal from PIC flowing to the base of this NPN transistor, it enables current flowing from power supply to the DC motor. Thus, whenever the machine detects a canister, the DC motor will rotate for a short moment sending out a visible signal to the user.

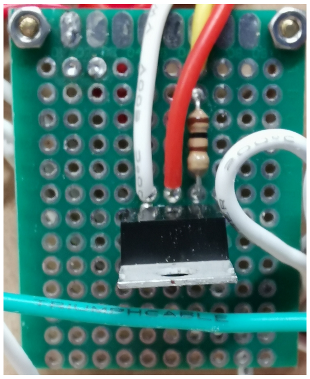


Figure 8.18: NPN Transistor DC Motor Controller

Changes from Proposal For our final design, we added the A4988 Stepper Driver Board to simplify both circuitry and microcontroller ends. It allows us to make the stepper programming a lot easier as well as less pins are needed to be soldered. In addition, we chose the NPN Transistor controlled motor to be our open circuit, which we did not mention in the proposal, as per the requirement of the course. More importantly, we ended up using all four encoders on the motors instead of just one as stated in our proposal. The machine driving straightly using PD control would not be possible without any of the encoders.

8.4.4 Power Delivering

Table 8.2: Power Estimation

Components	**Power Required (P=VI)**
PIC Board and Arduino	16V x 500mA = 6W [ref]
Motor 1	4.32W [Appendix]
Motor 2	4.32W [Appendix]
Motor 3	4.32W [Appendix]
Motor 4	4.32W [Appendix]
Encoder 1	0.75W (estimated)
Encoder 2	0.75W (estimated)
Encoder 3	0.75W (estimated)
Encoder 4	0.75W (estimated)
Motor Driver 1	16V x 22mA = 0.352W [Appendix]
Motor Driver 2	16V x 22mA = 0.352W [Appendix]
Stepper Motor	16V x 1.2A = 19.2W [Appendix]
Stepper Motor Driver Board	5V x 1.2A = 6W [Appendix]
Ultrasonic Sensor	5V x 15mA = 0.075W [Appendix]
Flag Indicator Motor	16V x 0.2A = 3.2W [Appendix]
IR sensor 1	5V x 43mA = 0.215W [Appendix]
IR sensor 2	5V x 43mA = 0.215W [Appendix]
IR sensor 3	5V x 43mA = 0.215W [Appendix]
IR sensor 4	5V x 43mA = 0.215W [Appendix]
Total Power	56.319W
Total Current	3.52A
Total Power without Stepper	31.119W
Total Current without Stepper	1.94A

For our final design, the power required to run every components on machine at the same time is similar compare to the power calculation performed above in the table 8.2. However, the machine was designed so that the stepper will not consume power when the machine is driving, using a simple stepper board enable pin.

We tested our machine using a DC power supply which can tell the current as the machine operates. We found that the machine needs less than 2A to work with the stepper disabled. Also, from the data sheet, we know that the stepper also needs around 1.2A to work. Hence, our power supply choice is very important in terms of delivering power and having a higher capacity.

18650 Type Batteries are the chosen power supply for the final design. We used four of these on the machine. Each one has 3.7v regularly and 4.2v when fully charged. They can support maximum 2.6A of current and each one has a capacity of 2600mAh.

All components except V_{cc} are directly supplied by the power supply which is around 16.8V when fully charged (normally tested at 16V). The LM338 voltage regulator on the PIC board is enough to handle this power supply. Although, an unregulated voltage would cause the motors, which are responsible for mobility, to perform inconsistently, we used PD control to encounter this problem, and it was quite successful.

8.4.5 Future Improvements for this subsystem

Overall circuit did a decent job on soldering since there was few circuit bugs throughout the course of the project. However, the circuit planning and the wire managing need improvements.

Circuit Planning should be more clear and well-thought. This can prevent the problem of circuit components over-heating themselves. More examples can be, do I really need a voltage regulator to regulate the input voltage for the entire system, or do I soldered the bus connector in the right way?

Wire managing The labelling is well done. However, the wire length as well as the well organization is pulled up poorly. The improvements can be done by thorough discussion with electromech member to confirm more details on the wire length and organize the wire path prior to housing the boards.

8.5 Microcontroller

The microcontroller system of the project is responsible for handling logical programming control of the machine. The control unit takes signals from input sensors and devices, makes decisions according to the programmed code, and changes the states of the output components, such as, motors and LCD displays. The main program has been divided into the following subroutines: user interface, mobility, canister recognition, ball dispensing, RTC (Real Time Clock), permanent log, and remote control. The following flowchart summarizes the logical flow of the program, connecting each subroutine together to perform the task.

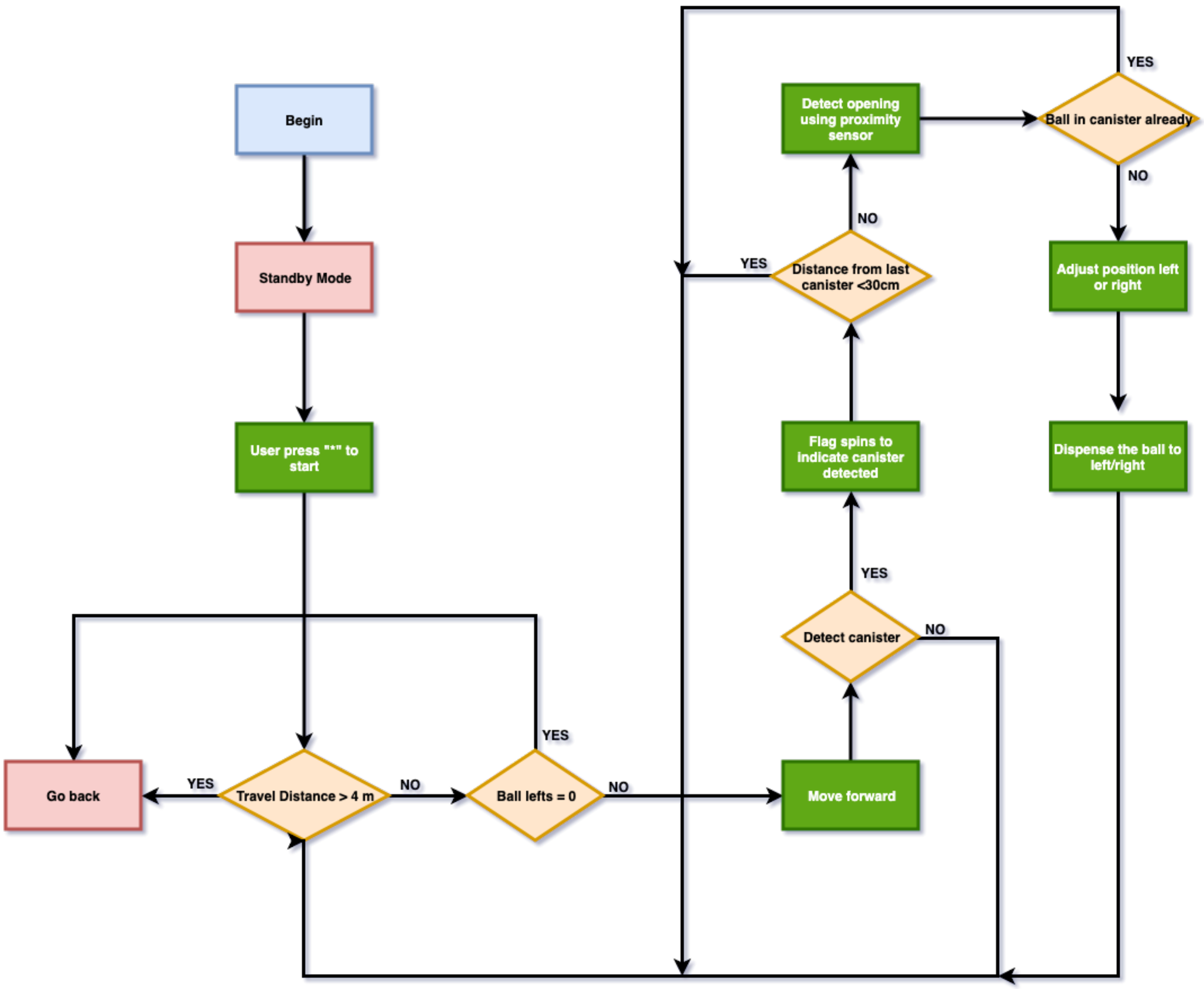

Begin
Standby Mode
User press "*" to start
Go back
YES
Travel Distance > 4 m
NO
YES
Ball lefts = 0
NO
Move forward
Detect canister
YES
NO
Flag spins to indicate canister detected
Distance from last canister <30cm
YES
NO
Detect opening using proximity sensor
Ball in canister already
YES
NO
Adjust position left or right
Dispense the ball to left/right

Figure 8.19: Logic Flow of the Programming Code

8.5.1 Choice of Microcontroller

This project uses the PIC DevBugger Development board with two microcontrollers, PIC18F4620 and Arduino Nano. The chosen board suits the needs of this project well with its pre-designed modules, such as keypad, LCD display, and RTC. For example, the LCD display has already been connected to PORT D of the PIC18F4620 chip. In additional, another important feature of this development board is its debug module, which contains a matrix of LEDs representing the state of the I/O ports (LED on when the pin is high and off when the pin is low). The switches besides LEDs could also simulate the input and output values for quick debugging. The board is powered by the female barrel jack using a DC supply between 8 to 15 volts [7].

The PIC18F4620 chip is the main control unit of this project. It is programmed with C language using MPLAB X IDE via PICkit 3. The PIC chip is responsible for the following tasks:

- User Interface: collect user inputs from keypad and output results on LCD display
- Permanent Log: EEPROM stores the operating log of previous four tasks
- RTC: communicate with DS1307 via I2C to acquire current time

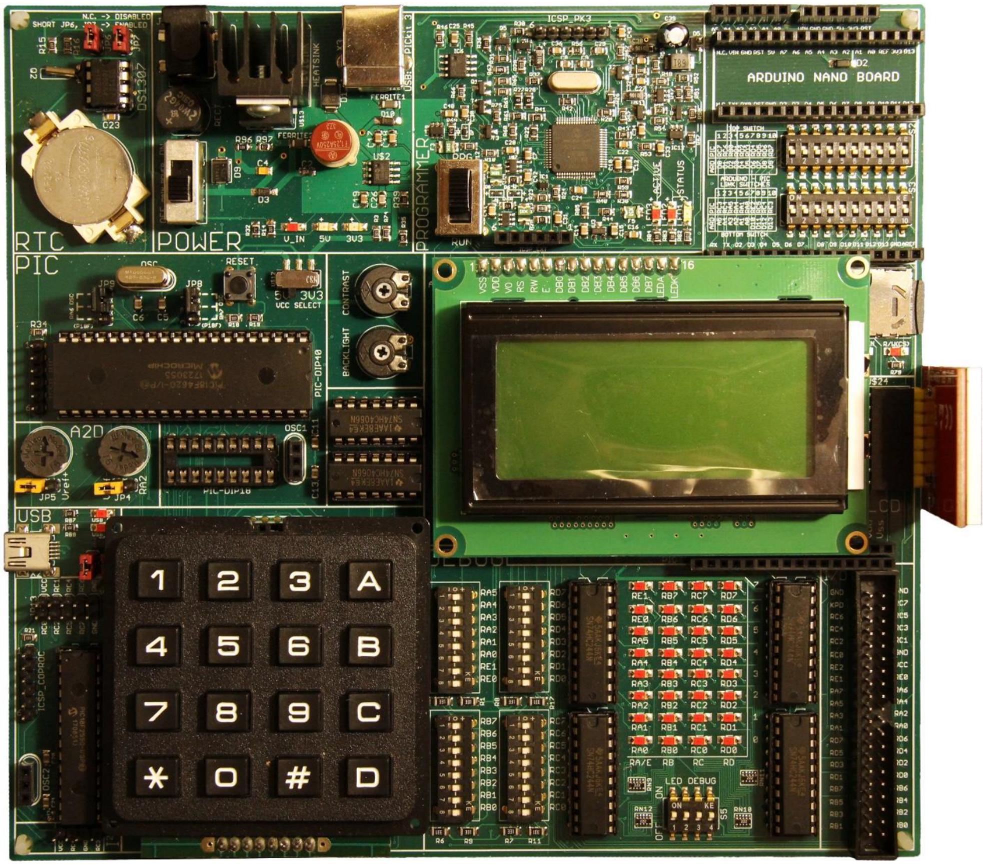


Figure 8.20: PIC DevBugger Development Board Top View[7]

- Control Arduino: send commands to Arduino to turn the DC motors (wheels) on or off
- Control Sensor: read signals from US and IR sensors to determine the orientation of the canister and the presence of ball
- Ball Dispensing: drive stepper motor to dispense the ball to left or right according to the orientation of canister

Due to the insufficient PWM pins from PIC18F4620, the project uses an additional microcontroller, Arduino Nano to control the mobility of the machine. The Arduino Nano board is programmed in C language using the Arduino IDE. The PIC and Arduino communicate using I2C where PIC is the master device and Arduino is the slave. The following list summarizes the tasks operated by Arduino Nano:

- Encoder Reading: use interrupt to keep reading encoder values and use for PD output computation
- PD DC Motor Control: Arduino is in a loop of computing the best PWM duty cycle for the wheels from the encoder reading as the feedback input
- Remote Control: Arduino receives the signal via IR receiver sensor to start and stop the machine
- PIC Communication: Arduino sends the distance data from encoder to the PIC whenever PIC requests the information

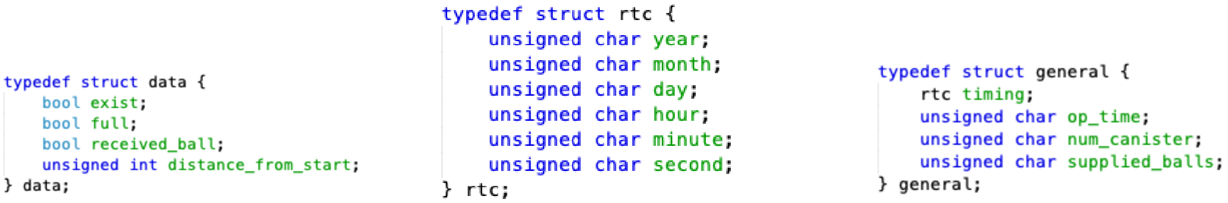
Changes from Proposal According to the initial proposal, the machine will only be equipped with the PIC microcontroller to perform all the tasks. During the integration phase of the project, our team realized the mobility of the machine could not be fully controlled unless each of the 4 DC motors is assigned a PWM output pin. This could not be possibly done without the help from an additional microcontroller after consulting with Professor Emami. In addition, the choice of using the DevBugger Development board allows the team to add an Arduino Nano board to the design with a low cost due to its pre-designed Arduino module. Challenges faced while adding the new Arduino board will be discussed in detail in section 9.1.2.

8.5.2 User Interface

The user interface (UI) of the machine consists of the data structure created for data storage, LCD display, remote control operation, and the PC interface. We have applied the principles of design for usability to ensure the UI is easily understandable to the general public.

Data Structure

In order to store all of the relevant data during the operation and return them back to user once the task is over, three data structures are created for this project (figure 8.21).



```
typedef struct data {
    bool exist;
    bool full;
    bool received_ball;
    unsigned int distance_from_start;
} data;
```

```
typedef struct rtc {
    unsigned char year;
    unsigned char month;
    unsigned char day;
    unsigned char hour;
    unsigned char minute;
    unsigned char second;
} rtc;
```

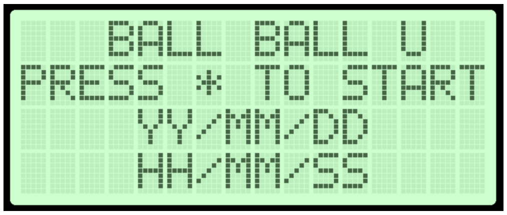
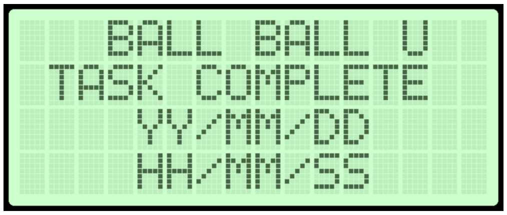
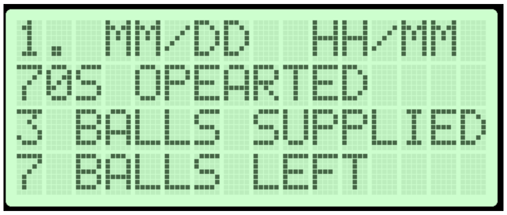
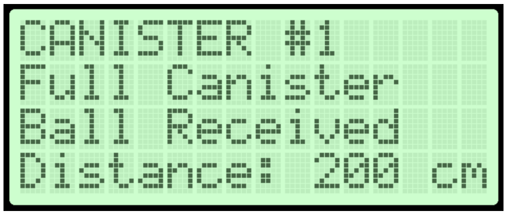
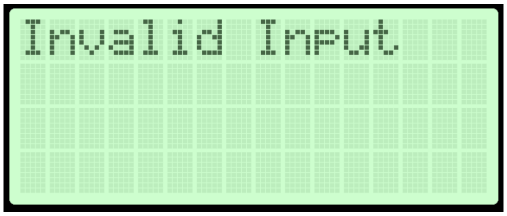
```
typedef struct general {
    rtc timing;
    unsigned char op_time;
    unsigned char num_canister;
    unsigned char supplied_balls;
} general;
```

Figure 8.21: Data Structure

LCD Display

LCD display is used for delivering operation information to the user before and after the task. In particular, the LCD displays the real time clock once the machine is turned on and the user could choose to start the operation by pressing ’*’ on the keypad. Furthermore, the user could retrieve the operation information by pressing ’A’ on the same page. The detailed illustration of the LCD display is summarized in the following table.

Table 8.3: LCD Display

Main Menu This is displayed when the machine is turned on. Real time clock is displayed on the screen. While in this state, the machine could be retrieved data about the previous operations if the user presses ’A’ on the keypad. Meanwhile, the operation could be started if the user presses and holds ’*’ key.	 BALL BALL U PRESS * TO START YY/MM/DD HH/MM/SS
Task Completion Message This is similar to the display before the operation but with a task completion message. The machine is now ready to provide the information about the last run as well as all of the previous operations.	 BALL BALL U TASK COMPLETE YY/MM/DD HH/MM/SS
General Operation Information This screen displays the month, date, hour, and minute that the task takes place in order to distinguish it from other operations. In addition, other relevant information, such as, the operating duration and the number of balls supplied by the machine and left in stock will also be provided to the user.	 1. MM/DD HH/MM 70S OPEARTED 3 BALLS SUPPLIED 7 BALLS LEFT
Detailed Canister Information Each visited canister will have its own screen for displaying detailed information as required in RFP [1]. The user could access these pages by pressing the corresponding number key while in the **General Operation Information Page**. (e.g. press ’1’ for canister #1).	 CANISTER #1 Full Canister Ball Received Distance: 200 cm
Unrecognized Input Error Message If the user inputs something that the machine does not recognize during data retrieval, it will quickly display the error message screen and switch back the most recent display and ask the user to input the request again.	 Invalid Input

Remote Control

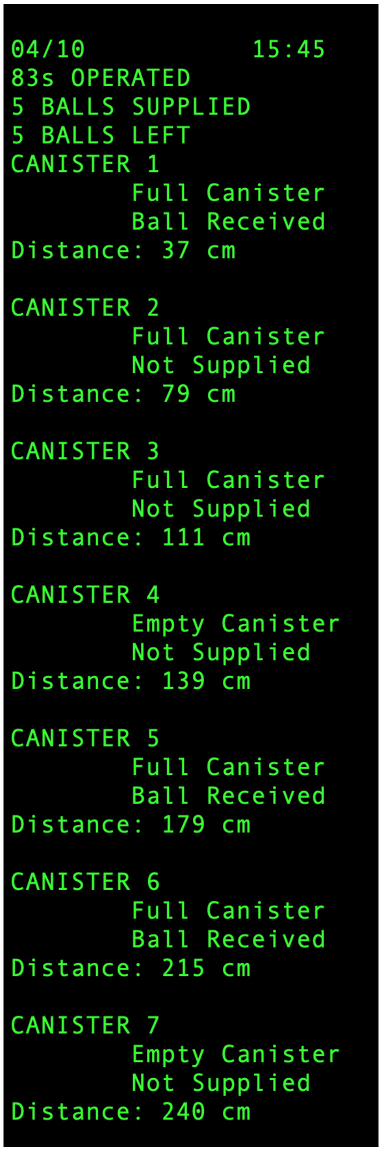
The machine is equipped with a remote control where the user could start the operation by pressing the "play/pause" button and stop the task by pressing the same button again. In addition, the user could offload balls by pressing '»|' or '|«' key to drive the stepper rightwards or leftwards respectively.

Changes from Proposal This is an additional feature to the initial proposed design. Adding this feature allows the user to have access to the control of the machine when he/she is nearby. Most importantly, this is designed for safety as the user could terminate the operation by pressing the "play/pause" button.

Suggested Future Improvements The remote control could be extended to a variety of functionality. Currently, remote control only supports starting and stopping the operation by turning on and turning off DC motors and offloading balls. However, the design could be improved by adding remote control during the autonomous operation. If anything unexpected happens during the operation, the manual inputs could take over the control and drive the robot. Furthermore, the range of the remote control could also be improved to maximize the convenience of using this functionality. According to the data sheet of the IR receiver, the range could be up to 15 m; however, various testing results show that the remote control is only reliable within 4 metres, and the controller has to be in front of the IR receiver in order for the device to receive the signal. This is possibly caused by interference with other sensors. These issues could be addressed by having an IR receiver module with an ideal position.

PC Interface

The machine's operation log could also be accessible when connected to a PC. This feature is achieved by reading EEPROM of PIC via MPLAB X IDE in hexadecimal and the encoded data will be processed by a Python program (full code attached in section G). Then the user could read or print out the log sheet which contains all information which has a similar template to the one provided by the LCD display on the PIC board.



```
04/10           15:45
83s OPERATED
5 BALLS SUPPLIED
5 BALLS LEFT
CANISTER 1
        Full Canister
        Ball Received
Distance: 37 cm

CANISTER 2
        Full Canister
        Not Supplied
Distance: 79 cm

CANISTER 3
        Full Canister
        Not Supplied
Distance: 111 cm

CANISTER 4
        Empty Canister
        Not Supplied
Distance: 139 cm

CANISTER 5
        Full Canister
        Ball Received
Distance: 179 cm

CANISTER 6
        Full Canister
        Ball Received
Distance: 215 cm

CANISTER 7
        Empty Canister
        Not Supplied
Distance: 240 cm
```

Figure 8.22: Sample log sheet from PC interface

Changes from Proposal The newest version of the machine is equipped with PC interface, which has not been introduced in the proposal. Adding this feature has applied the principles of design for usability and accessibility. The user could view the log sheet on a PC and is free to share and store it anywhere. In addition, this feature also makes the machine more accessible in a way that the operation data on PC could be processed with the help of "text-to-speech" feature.

Suggested Improvements The PC Interface could be improved by having a designated application for the machine. The usability of the robot will be significantly improved if the user could have an PC application for viewing its operation information.

8.5.3 Permanent Log

The PIC contains 1024 bytes of EEPROM and part of it will be used for storing permanent operation logs. It is designed for remembering five operations in total and the earliest operation log will be overwritten once the new task is complete. The following table illustrates how EEPROM is designed for storing all relevant information.

Table 8.4: EEPROM Data Interpretation

Bits	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
General Information	Operated Month							
	Operated Date							
	Operated Hour							
	Operated Minute							
	Duration in Seconds							
	Number of Supplied Balls							
Canister #1 Information	Unused				Distance MSB	Received Ball	Full	Exist
	Distance Lower Bits							
Canister #n Information	...				...	...	...	...
	...							

Changes from Proposal Having a permanent log storage is an additional feature from the proposed design. Since the PIC has its own EEPROM, this feature could be added with no extra cost. User could benefit from this feature because it allows users to compare the results of the previous operations

Suggested Future Improvements The permanent log feature could be improved by writing these operation information to a portable storage unit, such as a flash drive or SD card. In this way, data could not only be permanently stored but also transferred to be loaded to other devices for viewing or processing. Moreover, EEPROM of the microcontroller is limited. In particular, the PIC has 1024 bytes available. Once all of the space are filled, the old data has to be erased in order for the new data to be added. On the other hand, users could choose to replace a new portable data drive once the old one is full to avoid any data overwritten if the log is written to a SD card.

8.5.4 Communication between PIC and Arduino

Arduino acts as a slave device to the PIC via I2C communication and controls the four DC motors. To start off, the communication could be divided into 2 parts: PIC sending data to Arduino and PIC requesting data from Arduino. The following table summarizes the encoding commands:

Table 8.5: Summary of encoded commands received by Arduino from PIC

Encoded Commands Sent from PIC	**Functionality**
0	Stop All 4 DC Motors
1	Start All 4 DC Motors Running Forward
2	Start All 4 DC Motors Running Backward
3	Shift Right
4	Shift Left
5	Terminate All Motors Immediately

In addition, PIC will request information from the Arduino at the beginning and ending of the operation. In particular, Arduino is capable of receiving remote-control signals, which are required to be sent to the main controller, PIC for further processing. For instance, the machine is able to off-load balls by remote control via an IR receiver on Arduino. Then the stepper motor that is controlled by the PIC will rotate to perform this task when the signal is transferred from Arduino via I2C. The following table lists all of the decoding commands sent by Arduino:

Table 8.6: List of Decoded commands sent from Arduino to PIC

Decoded Commands Sent from Arduino	Functionality
2	Move Stepper Motor to Right Side (Ball Off-loading)
3	Move Stepper Motor to Left Side (Ball Off-loading)
253	Remote Start Operation
254	Task Complete

Changes from Proposal The I2C communication between PIC and Arduino is introduced after the implementation of the Arduino to control PWM outputs. As described in the proposal, the only device that uses I2C is the real time clock. However, one must be careful of using the Arduino since it shares the I2C bus with other slave devices like the real time clock. One has a high chance of burning the real time clock chip when finish uploading the code with the PIC board power on. Therefore, the user must turn off the PIC board while uploading the code in order to prevent any damages to the real time clock module of the board.

8.5.5 Mobility

Setup As mentioned before, Arduino Nano is responsible for controlling the mobility of the machine. It controls four DC motors with 12 output pins including 4 PWM outputs and 8 digital pins. When the Arduino is initialized, it will set up the four encoder pins as external interrupts and ready to count the pulses. PD computations are initialized with proportional coefficient 0.06 and derivative coefficient 0.2. The above coefficients are obtained from numerous testings introduced in PID Tuning Technical Support. Meanwhile, the four DC motors are in idle states where the forward signals are set but PWM enables are null. Once the Arduino receives the flag from the PIC via I2C, it will starting running the main loop.

Main Loop The duties of the main loop in the Arduino program are summarized as the following: The target encoder counts are increased by a fixed number in the loop. Adding a larger number each loop will make the motors turn in a larger speed. PD motor controls will then collect input information from the four encoders and compute the optimal PWM output values to enable the four DC motors by matching all encoder counts to the target encoder values mentioned above. In this way, the four DC motors will ensure the four wheels are running at the same speed; therefore, the machine will travel in a straight line. At the same time, the I2C interrupts are running in background to receive the signal of disabling and enabling DC motors from the PIC.

Suggested Future Improvements The mobility of the robot could be improved by attaching all 8 channels of the 4 encoders to the microcontroller for more accurate and precise reading. The algorithm of the current design turns on the encoder interrupt when the wheel moves forward and turn off when it moves backwards. This will generate some error in the reading due to the process time of the code. This error could be reduced by having both channel A and B attached, so the microcontroller would be able to tell if the motor is running in a CW or CCW direction. A more accurate reading of the encoder will definitely improve the mobility of the machine because the input of PD motor control highly depends on the reading of encoders.

8.5.6 Canister Detection

As mentioned earlier, the canister detection routine is performed by the PIC. When the machine travels along the row of canisters, the program keeps tracking the distance value acquired from the US sensor. The

US sensor is installed at the front left of the machine facing rightwards. If there is object detected within a range of 30 cm, the machine will notify the user that there is a canister in range. Meanwhile, the arrival distance from the start line will be recorded. Then the sensor will grab 2 more distance values and take the median to determine the direction of opening of the canister. Then the 2 IR proximity contrast sensors will be activated to see if there is a ball in the canister. The machine will be stopped if the canister is empty and the distance from the previous canister is greater than 30 cm. Otherwise, the robot will continue to move forward while recording the information to its operation data.

Suggested Future Improvements (US Sensor) The US sensor may not output a correct distance if the machine is slightly tilted. This may eventually affect the determination of the direction of opening. This issue could be resolved by adding more US sensors. For instance, one choice is to add another US sensor right after the current one. In this way, we could compare the distances given by these two sensors to determine if the canister is angled. If it is recognized to be angled, a new model should be applied to determine its direction of opening.

Suggested Future Improvements (IR Sensor) In addition, the sensor to determine if there is a ball could also be problematic in different light settings. Since the sensor we chose to detect balls is a digital IR sensor, which outputs high voltage if nothing detected and low voltage if something is in range. This contrast sensor would be able to detect a ball in range but not the canister if it is appropriately calibrated. In this design, we used a accumulator to count how many times that the sensor detects something in range while scanning through one particular canister. If the accumulator is relatively large, it is determined to contain a ball in canister; otherwise, it is empty. The accumulator of the sensor between the case of ball in canister and no ball in canister could be ambiguous due to the light setting of the surroundings because the floor could reflect light which has a significant interference with the IR sensor. This issue could be resolved by replacing it with an analog IR sensor. Instead of giving a binary answer of yes or no, the analog IR sensor will output the distance to the detected object. Then, two models could be set up using supervised learning algorithms, and the machine could be trained to classify if the canister contains a ball or not. Particularly, the case with a ball would have a distribution like a mountain valley because the middle of the ball will be the closest to the sensor with the smallest distance. On the other hand, the case without a ball would have a uniform "mountain hill" distribution where the detected object is the base of the canister.

8.5.7 Ball Dispensing

Ball dispensing action will be triggered by the PIC sending signals to the Arduino, making a stop at the canister by turning off the 4 DC motors. Then the PIC will determine if the machine should be shifted horizontally to move closer to the canister depending on the position reported by the US sensor. The position of the canister will be categorized into the following 6 groups and appropriate actions will be taken correspondingly.

Table 8.7: Left/Right Shift in Ball Dispensing

	Distance Reported from US Sensor	**Actions**
Facing Left	17.5cm <Distance <25cm	N/A
	25cm <Distance <27.5cm	Shift Right ONCE
	27.5cm <Distance <30cm	Shift Right TWICE
Facing Right	13cm <Distance <17.5cm	N/A
	10.5cm <Distance <13cm	Shift Left ONCE
	0cm <Distance <10.5cm	Shift Left TWICE

Changes from Proposal Despite the decision of changing the orientation of the wheels from the one described in the proposal, the programming codes do not need to be changed significantly thanks to the PD motor control algorithm.

Suggested Future Improvements The shift motion of the machine highly depends on the status of the battery supply and the floor condition; therefore, the performance of the horizontal shift is not 100% reliable. In particular, the biggest issue is that the horizontal movement shifts the machine at a different distance that is not constantly the same during each run. For instance, during the first run of the public demonstration, the machine accidentally hits the canister while shifting itself towards one way in order to dispense the ball. As a future improvement, we could try to use high torque DC motors with no-slipping Mecanum wheels. In this way, we could try to ensure all 4 wheels are rotating the same number of turns with PD motor control. As a result, the left and right shift motion will be more reliable.

8.5.8 Pin Assignments

PIC Pin Assignments

Table 8.8: PIC Pin Assignment Diagram

Pin	**Function**	**Pin**	**Function**
GND	/	**GND**	/
KPD	Connected to RC7	**RC7**	KPD disable
RC6	IR signal left	**RC5**	IR signal right
RC4	RTC (internal)	**RC3**	RTC (internal)
RC2	N.C.	**RC1**	N.C.
RC0	N.C.	**GND**	/
RE2	N.C.	**VCC**	/
RE1	IR FRONT	**RE0**	IR FRONT
RA7	PIC (internal)	**RA6**	PIC (internal)
RA5	Stepper Enable	**RA4**	Flag signal (canister detected)
RA3	Stepper Step	**RA2**	N.C.
RA1	Stepper Dir	**RA0**	N.C.
RD7	LCD (internal)	**RD6**	LCD (internal)
RD5	LCD (internal)	**RD4**	LCD (internal)
RD3	LCD (internal)	**RD2**	LCD (internal)
RD1	N.C.	**RD0**	N.C.
RB7	Keypad | side US Trig	**RB6**	Keypad
RB5	Keypad | side US Echo	**RB4**	Keypad
RB3	N.C.	**RB2**	N.C.
RB1	Keypad Interrupt	**RB0**	ENCODER

Arduino Pin Assignments

Table 8.9: Arduino Nano Pin Assignment Diagram

Pin	Function	Pin	Function
TX1	N.C.	**VIN**	N.C.
RX0	N.C.	**GND**	/
RST	N.C.	**RST**	N.C.
GND	/	**5V**	/
D2	Encoder Front Right	**A7**	N.C.
D3	Encoder Back Left	**A6**	N.C.
D4	Back Right Motor +	**A5**	I2C
D5	Front Right PWM Enable	**A4**	12C
D6	Back Right PWM Enable	**A3**	Encoder Back Right
D7	IR Receiver Input	**A2**	Encoder Front Left
D8	Back Left Motor +	**A1**	Front Right Motor -
D9	Front Left PWM Enable	**A0**	Front Right Motor+
D10	Back Left PWM Enable	**REF**	N.C.
D11	Back Left Motor -	**3V3**	N.C.
D12	Front Left Motor +	**D13**	Back Right Motor -

Chapter 9

Integration

The integration of the 3 subsystems have started since week 8 of the project. The stages of the integration could be categorized as the following two stages: physical integration and functionality integration.

The Physical Integration includes the following tasks and took the team 5 days to complete:

- All circuit boards are attached stably onto the structure.
- The US sensor and IR sensors are screwed into the appropriate positions.
- The PIC board is installed on the top board for the user to press keypad and view the LCD display.
- The battery holder is placed on the top board for easier
- Wires are braided to take less space and avoid any interference.

The Functionality Integration is much more complicated and involves calibration that may change the physical structure of the robot. Therefore, the team went through countless cycles of small modification to the physical structure, such as the position of the sensors and adding additional sensors to ensure the full functionality of the machine. During the two stages of integration, the team faced several major challenges and the solutions to each challenges will be described below.

In the sections, we will also discuss the problems that came up during and after the integration as well as how we solve these issues.

9.1 Mobility

The mobility of the machine has been improving with various modifications during integration. The process of the mobility improvement includes the initial planning of having 2 PWM pins for controlling 4 DC motors on PIC, additional Arduino Nano board to control 4 motors with 4 PWM pins, and the final design using PD motor feedback loop.

9.1.1 4WD Calibration with 2 PWM Pins from PIC

After completing the physical integration, we started to calibrate the 4 motors. Initially, we thought that, since the diagonal wheels rotates in the same direction, diagonal motors shared the same PWM. We have mistakenly relied too much on the fact that they are "exactly same" motor with same manufacture companies. However, the fact is that motors with same PWM value will not have the same output in real life.

9.1.2 Additional Arduino Nano to Control 4 DC Motors

The mobility of the design introduced in the proposal is extremely limited due to the share of PWM pins between 2 DC motors. Due to the insufficient PWM pins from the PIC, the Arduino Nano was added to the design, controlling four DC motors with 4 PWM outputs and 8 digital outputs to allow the machine to move in any direction. However, this approach requires a very consistent voltage from the power supply, whereas, the reality is always not ideal. This means that if we set a fixed PWM value or a range, the motors will definitely perform differently when powered from an inconsistent power supply (e.g. different output voltage, voltage changes as the operation is in progress). Meanwhile, the voltage regulator was removed in the design due to its overheating issue (details discussed in section 9.2). As a result, the PWM outputs would not be consistent because they highly depend on the status of the battery supply. This would be very problematic if the team chose to use a single PWM duty cycle for each of the four DC motors throughout the operation. Thus, a feedback loop is needed to constantly changing the PWM duty cycles in order for the machine to perform basic mobility functionality.

9.1.3 Change Direction of the Motors

After week 12 functionality evaluation, our machine still could not move in a straight line. We spent almost two weeks to calibrate the 4WD. Prof. Emami suggested us to switch the direction of the motors so that the machine will drive in its primary direction (i.e. in the direction as a normal car would drive). We had to make this major change at that stage since we only had two weeks left for debugging and repeatablility testing. We had to make our robot drive straight in order to proceed the further testings. This forced us to give up our compactness feature (i.e. we must make the machine bigger to allow it to go over canisters). On the other hand, we could add the extendability feature.

Like we always did, we rebuilt this machine very quickly in just one day. The direction of the wheels are changed, as well as the upper deck design is changed along with the ball storage unit installed. As what we expected, we successfully calibrated the motors in just one day using PD control.

9.1.4 PID Motor Control

PD Motor Control Based on the failure from previous approaches, we have self-taught PID motor control. The method could allow the machine to make its own adjusting, telling itself how much its PWM values should change instead of we giving it a fixed range. In order to make this happen, we introduced four encoders to be the input, target encoder counts to be the set point, and adjusted PWM values to be the output.

9.2 Power Source

It is very important to have a reliable power supply for this machine. The power supply ensure that the machine could draw enough current from the source to perform appropriate actions without compromising the rated voltage.

9.2.1 Voltage Regulator

We added a LM338 to regulate the voltage for the entire circuitry, which means that all the input voltage for motors and PIC is at around 12V. We thought it could act as a protection as well as allowing the machine to work under a consistent condition. However, as we were doing further motor calibrations, the voltage regulator started to overheat itself leading to system shut downs. We were not sure about this problem. It might be that the LM338 is broken, needed a bigger heat sink, or the entire system require a lot of current. After we consulted with Prof. Emami, the current flowing through the system is at a safe value, 1A, yet, this voltage regulator still heat up very quickly. Thus, we decided to remove it. Due to the fact that the PIC board is able to handle the power supply (around 16V), we were safe to remove the voltage regulator we soldered before.

9.2.2 Battery Supply

A fully charged battery supply is a critical component to ensure a bug-free machine. Several unexpected behaviours observed during the integration phase:

- PIC resets unexpectedly without pressing any keys.
- Motors are not running with high-duty-cycle PWM outputs.
- Tasks could only perform properly if the microcontroller is connected to the PC with USB.

The above problems that we faced were all caused by a battery supply with low power and a non-functioning battery charger. As a result, these challenges were overcome with the help of a more advanced battery charger that shows the battery power.

9.2.3 Debugging

The following table summarizes the major problems with the project during integration and the appropriate solutions for solving them.

Table 9.1: Summary of Major Bugs with Solutions

Bugs	**Solutions**
Cannot detect openings correctly	Adjusting the boundary distance for distinguish left and right
Will not stop at the right place	Adjust distance needed to travel after detected a canister
Return right after 10 canisters are detected	Fix the logic
Front IR sensor detects balls inside canister, which it is not supposed to do so	Adjust its position, re-calibrate
PIC reset by itself	Power is not enough
Remote Control	Switch the IR receive pin to another one
RTC Works randomly	The RTC chip is broken
H-Bridge over heating	Bus was inserted reversely
Voltage regulator over heated	Remove the voltage regulator

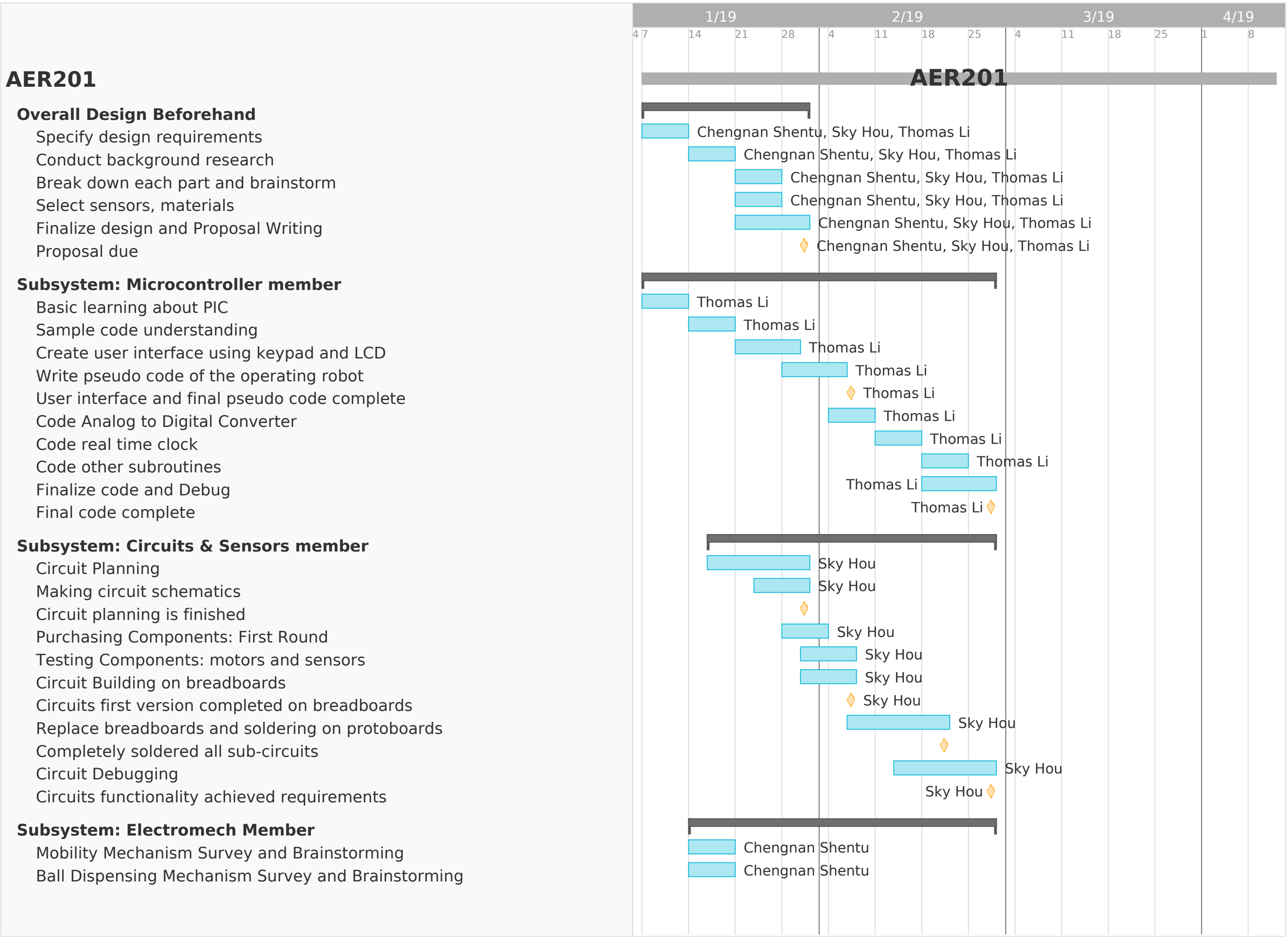
Chapter 10

Project Execution

In this section, we will present our project management technique throughout this project timeline. The first approach we took at the beginning is to create a Gantt Chart which outlines all the tasks and milestones. This Gantt Chart acts as the planned guide to the team while the project is in progress.

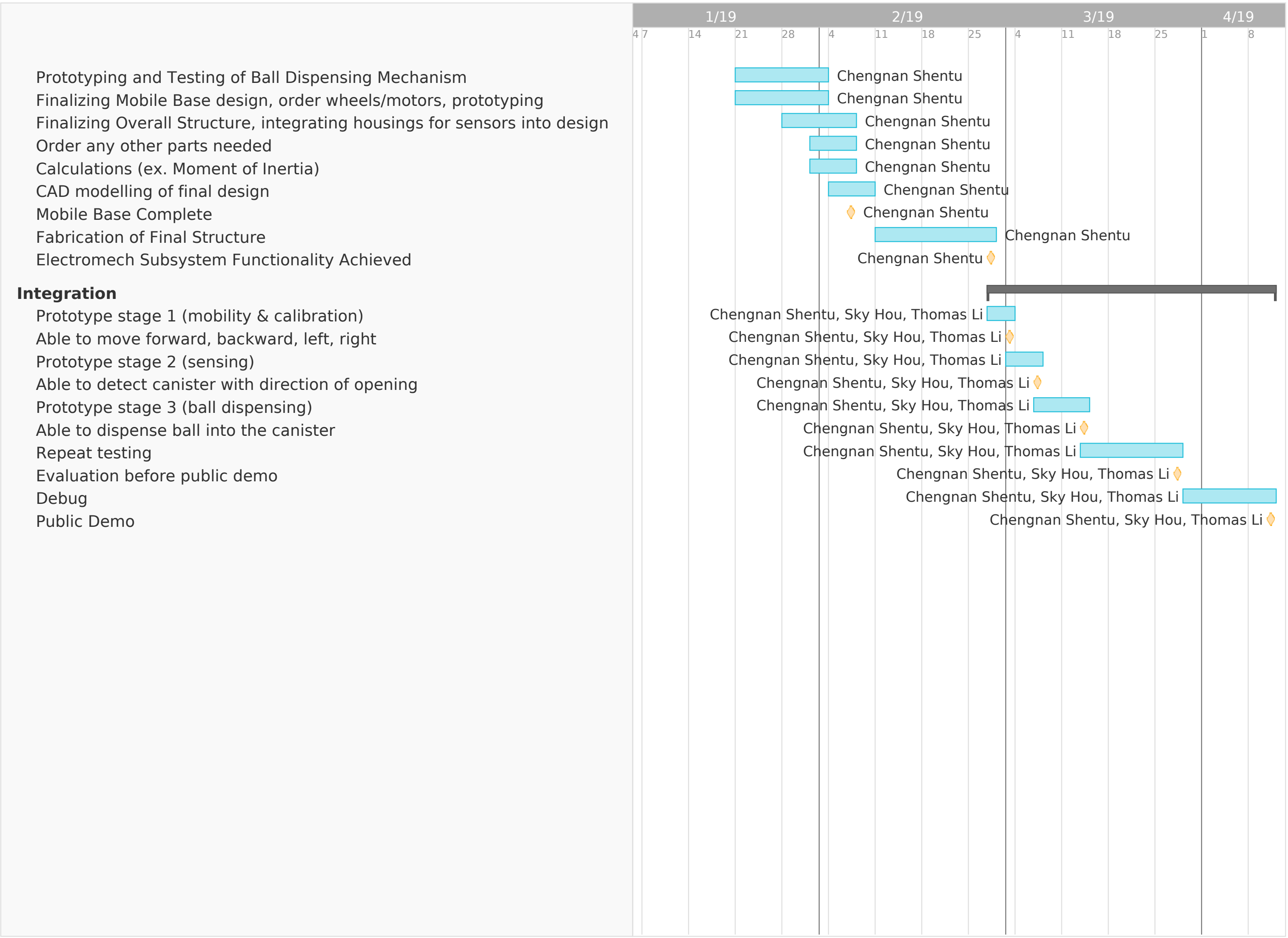
10.1 Initial Gantt Chart

teamgantt
Created with Free Edition



AER201

Overall Design Beforehand	
Specify design requirements	Chengnan Shentu, Sky Hou, Thomas Li
Conduct background research	Chengnan Shentu, Sky Hou, Thomas Li
Break down each part and brainstorm	Chengnan Shentu, Sky Hou, Thomas Li
Select sensors, materials	Chengnan Shentu, Sky Hou, Thomas Li
Finalize design and Proposal Writing	Chengnan Shentu, Sky Hou, Thomas Li
Proposal due	Chengnan Shentu, Sky Hou, Thomas Li
Subsystem: Microcontroller member	
Basic learning about PIC	Thomas Li
Sample code understanding	Thomas Li
Create user interface using keypad and LCD	Thomas Li
Write pseudo code of the operating robot	Thomas Li
User interface and final pseudo code complete	Thomas Li
Code Analog to Digital Converter	Thomas Li
Code real time clock	Thomas Li
Code other subroutines	Thomas Li
Finalize code and Debug	Thomas Li
Final code complete	Thomas Li
Subsystem: Circuits & Sensors member	
Circuit Planning	Sky Hou
Making circuit schematics	Sky Hou
Circuit planning is finished	
Purchasing Components: First Round	Sky Hou
Testing Components: motors and sensors	Sky Hou
Circuit Building on breadboards	Sky Hou
Circuits first version completed on breadboards	Sky Hou
Replace breadboards and soldering on protoboards	Sky Hou
Completely soldered all sub-circuits	
Circuit Debugging	Sky Hou
Circuits functionality achieved requirements	Sky Hou
Subsystem: Electromech Member	
Mobility Mechanism Survey and Brainstorming	Chengnan Shentu
Ball Dispensing Mechanism Survey and Brainstorming	Chengnan Shentu



Prototyping and Testing of Ball Dispensing Mechanism
Finalizing Mobile Base design, order wheels/motors, prototyping
Finalizing Overall Structure, integrating housings for sensors into design
Order any other parts needed
Calculations (ex. Moment of Inertia)
CAD modelling of final design
Mobile Base Complete
Fabrication of Final Structure
Electromech Subsystem Functionality Achieved

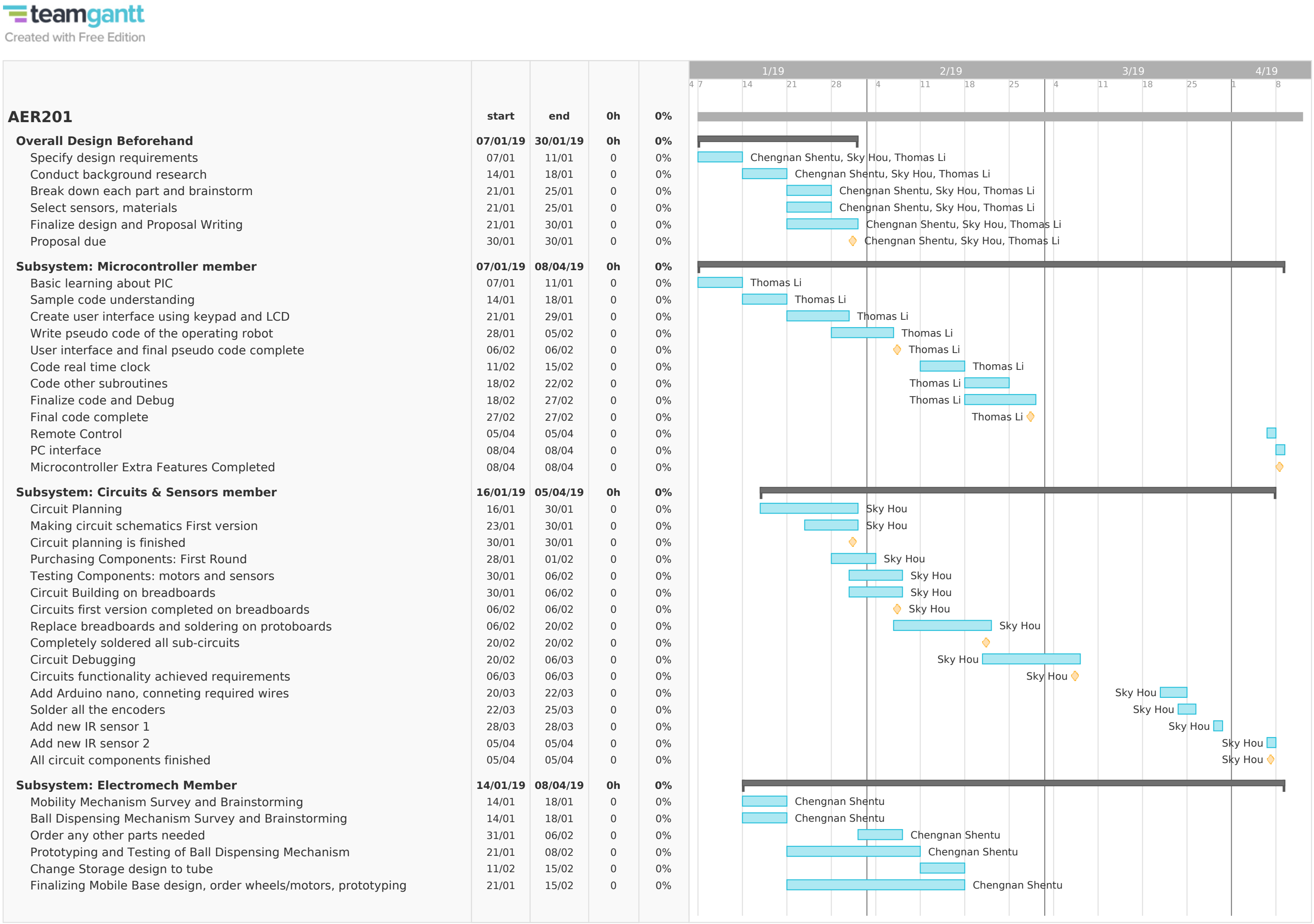
Integration

Prototype stage 1 (mobility & calibration)
Able to move forward, backward, left, right
Prototype stage 2 (sensing)
Able to detect canister with direction of opening
Prototype stage 3 (ball dispensing)
Able to dispense ball into the canister
Repeat testing
Evaluation before public demo
Debug
Public Demo

1/19
2/19
3/19
4/19
4
7
14
21
28
4
11
18
25
4
11
18
25
1
8
Chengnan Shentu
Chengnan Shentu
Chengnan Shentu
Chengnan Shentu
Chengnan Shentu
Chengnan Shentu
Chengnan Shentu
Chengnan Shentu
Chengnan Shentu
Chengnan Shentu, Sky Hou, Thomas Li
Chengnan Shentu, Sky Hou, Thomas Li
Chengnan Shentu, Sky Hou, Thomas Li
Chengnan Shentu, Sky Hou, Thomas Li
Chengnan Shentu, Sky Hou, Thomas Li
Chengnan Shentu, Sky Hou, Thomas Li
Chengnan Shentu, Sky Hou, Thomas Li
Chengnan Shentu, Sky Hou, Thomas Li
Chengnan Shentu, Sky Hou, Thomas Li
Chengnan Shentu, Sky Hou, Thomas Li

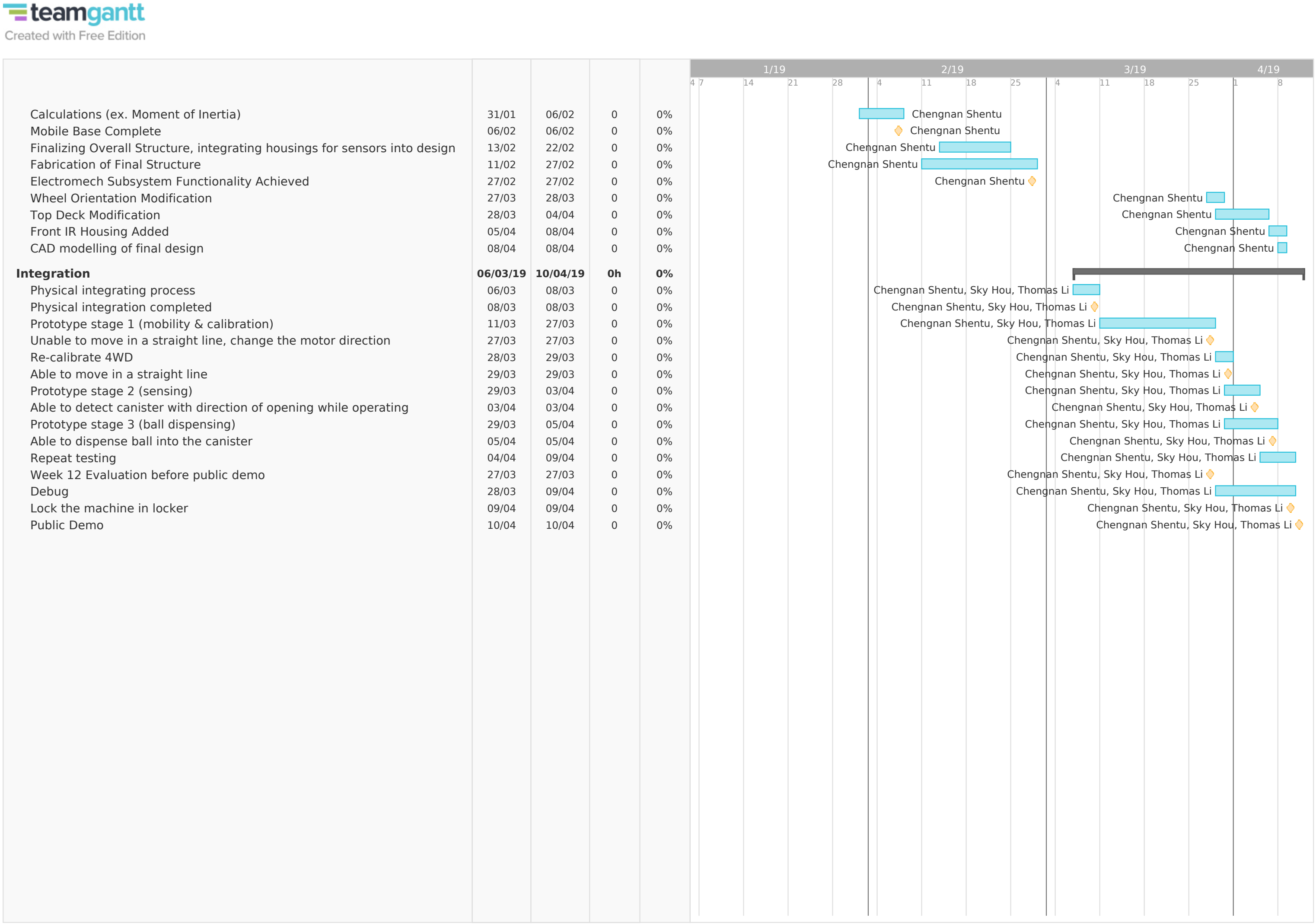
The above Gantt chart illustrates our team's initial planning for this project, from the design conceptualization to final prototype. The entire project is divided into three subsystems in the first half of the project planning. Each member would finish their subsystem on date and begin the integration at the start of the second half, while leaving about two weeks to debug (aiming for full functionality) and another two weeks to perform repeatablility testings. If the team met the milestones on time, the machine should have a great performance on the Demo day.

10.2 Final Gantt Chart



AER201	start	end	0h	0%
Overall Design Beforehand	**07/01/19**	**30/01/19**	**0h**	**0%**
Specify design requirements	07/01	11/01	0	0%
Conduct background research	14/01	18/01	0	0%
Break down each part and brainstorm	21/01	25/01	0	0%
Select sensors, materials	21/01	25/01	0	0%
Finalize design and Proposal Writing	21/01	30/01	0	0%
Proposal due	30/01	30/01	0	0%
Subsystem: Microcontroller member	**07/01/19**	**08/04/19**	**0h**	**0%**
Basic learning about PIC	07/01	11/01	0	0%
Sample code understanding	14/01	18/01	0	0%
Create user interface using keypad and LCD	21/01	29/01	0	0%
Write pseudo code of the operating robot	28/01	05/02	0	0%
User interface and final pseudo code complete	06/02	06/02	0	0%
Code real time clock	11/02	15/02	0	0%
Code other subroutines	18/02	22/02	0	0%
Finalize code and Debug	18/02	27/02	0	0%
Final code complete	27/02	27/02	0	0%
Remote Control	05/04	05/04	0	0%
PC interface	08/04	08/04	0	0%
Microcontroller Extra Features Completed	08/04	08/04	0	0%
Subsystem: Circuits & Sensors member	**16/01/19**	**05/04/19**	**0h**	**0%**
Circuit Planning	16/01	30/01	0	0%
Making circuit schematics First version	23/01	30/01	0	0%
Circuit planning is finished	30/01	30/01	0	0%
Purchasing Components: First Round	28/01	01/02	0	0%
Testing Components: motors and sensors	30/01	06/02	0	0%
Circuit Building on breadboards	30/01	06/02	0	0%
Circuits first version completed on breadboards	06/02	06/02	0	0%
Replace breadboards and soldering on protoboards	06/02	20/02	0	0%
Completely soldered all sub-circuits	20/02	20/02	0	0%
Circuit Debugging	20/02	06/03	0	0%
Circuits functionality achieved requirements	06/03	06/03	0	0%
Add Arduino nano, conneting required wires	20/03	22/03	0	0%
Solder all the encoders	22/03	25/03	0	0%
Add new IR sensor 1	28/03	28/03	0	0%
Add new IR sensor 2	05/04	05/04	0	0%
All circuit components finished	05/04	05/04	0	0%
Subsystem: Electromech Member	**14/01/19**	**08/04/19**	**0h**	**0%**
Mobility Mechanism Survey and Brainstorming	14/01	18/01	0	0%
Ball Dispensing Mechanism Survey and Brainstorming	14/01	18/01	0	0%
Order any other parts needed	31/01	06/02	0	0%
Prototyping and Testing of Ball Dispensing Mechanism	21/01	08/02	0	0%
Change Storage design to tube	11/02	15/02	0	0%
Finalizing Mobile Base design, order wheels/motors, prototyping	21/01	15/02	0	0%

teamgantt
Created with Free Edition



Calculations (ex. Moment of Inertia)	31/01	06/02	0	0%
Mobile Base Complete	06/02	06/02	0	0%
Finalizing Overall Structure, integrating housings for sensors into design	13/02	22/02	0	0%
Fabrication of Final Structure	11/02	27/02	0	0%
Electromech Subsystem Functionality Achieved	27/02	27/02	0	0%
Wheel Orientation Modification	27/03	28/03	0	0%
Top Deck Modification	28/03	04/04	0	0%
Front IR Housing Added	05/04	08/04	0	0%
CAD modelling of final design	08/04	08/04	0	0%
Integration	**06/03/19**	**10/04/19**	**0h**	**0%**
Physical integrating process	06/03	08/03	0	0%
Physical integration completed	08/03	08/03	0	0%
Prototype stage 1 (mobility & calibration)	11/03	27/03	0	0%
Unable to move in a straight line, change the motor direction	27/03	27/03	0	0%
Re-calibrate 4WD	28/03	29/03	0	0%
Able to move in a straight line	29/03	29/03	0	0%
Prototype stage 2 (sensing)	29/03	03/04	0	0%
Able to detect canister with direction of opening while operating	03/04	03/04	0	0%
Prototype stage 3 (ball dispensing)	29/03	05/04	0	0%
Able to dispense ball into the canister	05/04	05/04	0	0%
Repeat testing	04/04	09/04	0	0%
Week 12 Evaluation before public demo	27/03	27/03	0	0%
Debug	28/03	09/04	0	0%
Lock the machine in locker	09/04	09/04	0	0%
Public Demo	10/04	10/04	0	0%

1/19
2/19
3/19
4/19
4 7 14 21 28 4 11 18 25 4 11 18 25 1 8
Chengnan Shentu
Chengnan Shentu
Chengnan Shentu
Chengnan Shentu
Chengnan Shentu
Chengnan Shentu
Chengnan Shentu
Chengnan Shentu
Chengnan Shentu
Chengnan Shentu, Sky Hou, Thomas Li
Chengnan Shentu, Sky Hou, Thomas Li
Chengnan Shentu, Sky Hou, Thomas Li
Chengnan Shentu, Sky Hou, Thomas Li
Chengnan Shentu, Sky Hou, Thomas Li
Chengnan Shentu, Sky Hou, Thomas Li
Chengnan Shentu, Sky Hou, Thomas Li
Chengnan Shentu, Sky Hou, Thomas Li
Chengnan Shentu, Sky Hou, Thomas Li
Chengnan Shentu, Sky Hou, Thomas Li
Chengnan Shentu, Sky Hou, Thomas Li
Chengnan Shentu, Sky Hou, Thomas Li
Chengnan Shentu, Sky Hou, Thomas Li
Chengnan Shentu, Sky Hou, Thomas Li
Chengnan Shentu, Sky Hou, Thomas Li

Change From the Plan As the above Gantt chart shows, our executions were not exactly match what we had planned at the first stage of this project. The Gantt chart shows clearly that we spent a lot of time on calibrating motors making the machine drive in a straight line. What was worse is that we still could not make it happen. Thus, this led to a project delay since driving straight should be on the critical path of this project. We worked so hard to make up the wasted time later in the progress. Every member worked together to make important decisions such as switching the direction of the wheels and adding two IR sensors at the front of the machine. There two changes could be considered as a huge change during the last two weeks of the project, and that situation probably should never happen in the real world industry. Fortunately, we made it happen. It was all about our hard works and brave decisions.

Chapter 11

Conclusions

This report outlines the detailed design and integration of the machine, Ball Ball U. The project is firstly divided into three subsystems, electromech, circuits and sensors, and microcontroller sections. Three members of the design team focused on developing the design solution from three different perspectives at the beginning 7 weeks of the project.

11.1 Bottlenecks

At the end of this report, we want to point out the bottlenecks our team had been through. As the report mentioned earlier, we spent almost two weeks on calibrating the 4WD system using only manual adjusting. Even though, we started to embrace the power of PID control, we were still unprepared to make our initial design drive straight. Due to the time issue, we had to give up the initial design and move onto a new wheel configuration. Fortunately, we were able to pull this huge change up based on our previous experience with our machine, as well as the design feature which is really simple to be manufactured. Thus, we would say that making the machine driving straight is our biggest bottleneck during this project.

11.2 Future Improvements

Based on the machine's performance and our understanding of our machine, we would like to point out a few improvements which could potential make the machine better in terms of operation as well as user interface ?.

Operation Improvements If we had more time, we would want to make the driving more straight. Calibrations do needs time. Also, we should probably make our sensors detecting balls more reliable since there are sometimes that the sensors (IR and US) could make mis-detections. We probably should think about whether we should calibrate the existing ones or using better proximity sensors. User Interface User Interface could be improved by making a designated application on the PC for operating this machine. Meanwhile, this would make the robot much more marketable as computer application is very popular to the general public. In this way, people could also transfer the operating data much more easily with the help of PC. The permanent log could also be stored in the PC instead of the EEPROM of the microcontroller. This will also be much more reliable as it will not erase the previous operations once the storage is full.

In Conclusion Overall, our machine is really promising in terms of its performance on the Demo day. We were proud of ourselves that we actually pulled this up. Great team works have been done, and unforgettable friendships have been built up. "An Almost professional team" is said from Prof.Emami; we always came with problems to him, asked the right questions at the right time as well as making the significant decision which led to the reborn of our machine.

Bibliography

[1] M. Emami, *Multidisciplinary Engineering Design from theory to practice.* University of Toronto, 2019.

[2] E. X. Li, C. Hou, and C. Shentu, *Ball Ball U.*

[3] “What is a pid controller?” [Online]. Available: https://www.omega.com/prodinfo/pid-controllers.html

[4] “Arduino pid autotune.” [Online]. Available: https://playground.arduino.cc/Code/PIDAutotuneLibrary/

[5] [Online]. Available: http://support.motioneng.com/downloads-notes/tuning/pid_overshoot.htm

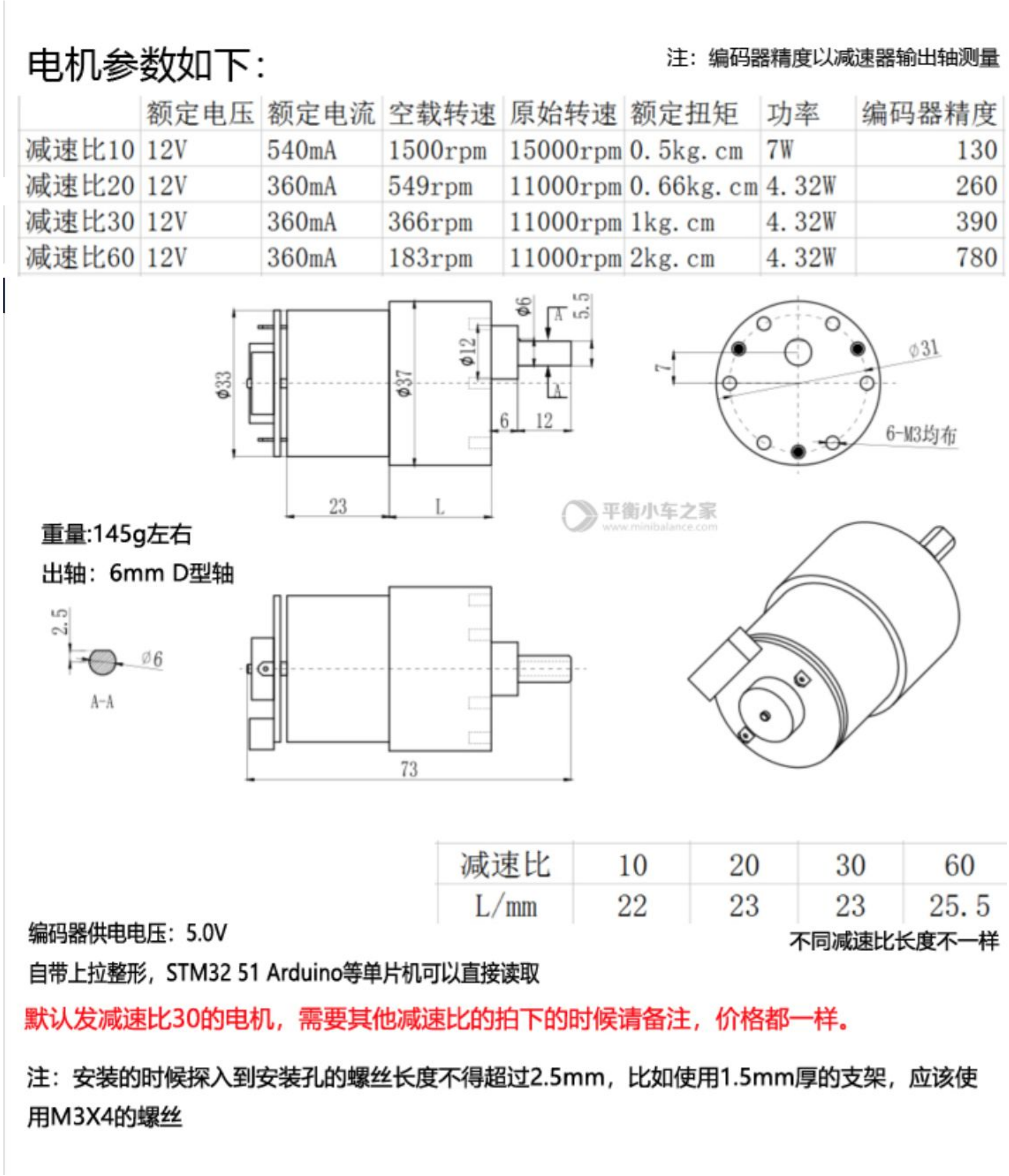
[6] “Encoder measurements: How-to guide,” national instruments. [Online]. Available: http://www.ni.com/tutorial/7109/en/

[7] *DevBugger User Manual*, Personal Mechatronics Lab, 2019.

Appendix A

Datasheets

A.1 DC Motor with Encoder Data Sheet



电机参数如下：

注：编码器精度以减速器输出轴测量

	额定电压	额定电流	空载转速	原始转速	额定扭矩	功率	编码器精度
减速比10	12V	540mA	1500rpm	15000rpm	0.5kg.cm	7W	130
减速比20	12V	360mA	549rpm	11000rpm	0.66kg.cm	4.32W	260
减速比30	12V	360mA	366rpm	11000rpm	1kg.cm	4.32W	390
减速比60	12V	360mA	183rpm	11000rpm	2kg.cm	4.32W	780

重量:145g左右

出轴：6mm D型轴

减速比	10	20	30	60
L/mm	22	23	23	25.5

不同减速比长度不一样

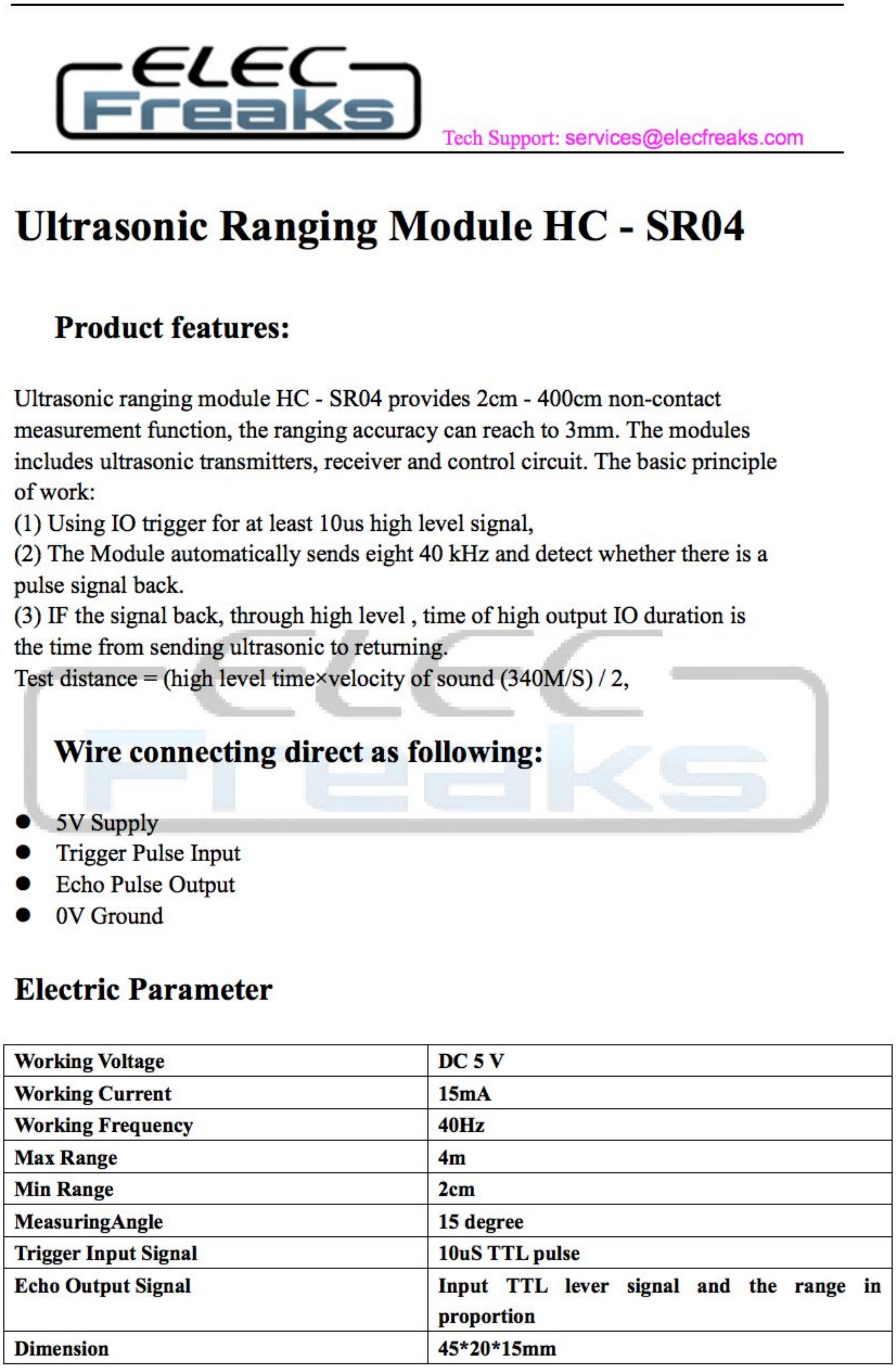
编码器供电电压：5.0V

自带上拉整形，STM32 51 Arduino等单片机可以直接读取

默认发减速比30的电机，需要其他减速比的拍下的时候请备注，价格都一样。

注：安装的时候探入到安装孔的螺丝长度不得超过2.5mm，比如使用1.5mm厚的支架，应该使用M3X4的螺丝

A.2 US Sensor HC-SR04 Data Sheet



Ultrasonic Ranging Module HC - SR04

Product features:

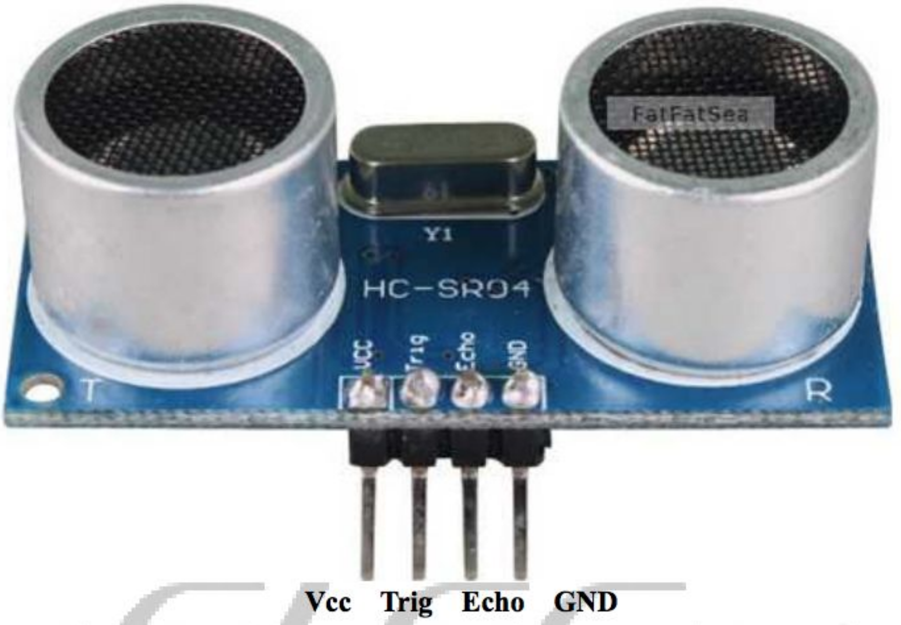
Ultrasonic ranging module HC - SR04 provides 2cm - 400cm non-contact measurement function, the ranging accuracy can reach to 3mm. The modules includes ultrasonic transmitters, receiver and control circuit. The basic principle of work:
(1) Using IO trigger for at least 10us high level signal,
(2) The Module automatically sends eight 40 kHz and detect whether there is a pulse signal back.
(3) IF the signal back, through high level , time of high output IO duration is the time from sending ultrasonic to returning.
Test distance = (high level time×velocity of sound (340M/S) / 2,

Wire connecting direct as following:

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

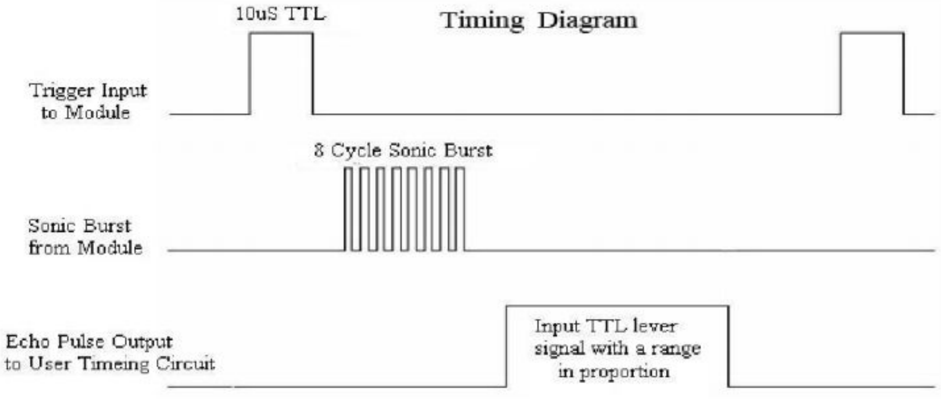
Electric Parameter

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



Timing diagram

The Timing diagram is shown below. You only need to supply a short 10uS pulse to the trigger input to start the ranging, and then the module will send out an 8 cycle burst of ultrasound at 40 kHz and raise its echo. The Echo is a distance object that is pulse width and the range in proportion .You can calculate the range through the time interval between sending trigger signal and receiving echo signal. Formula: uS / 58 = centimeters or uS / 148 =inch; or: the range = high level time * velocity (340M/S) / 2; we suggest to use over 60ms measurement cycle, in order to prevent trigger signal to the echo signal.


10uS TTL
Timing Diagram
Trigger Input to Module
8 Cycle Sonic Burst
Sonic Burst from Module
Echo Pulse Output to User Timeing Circuit
Input TTL lever signal with a range in proportion

A.3 IR Receiver VS1838 Data Sheet

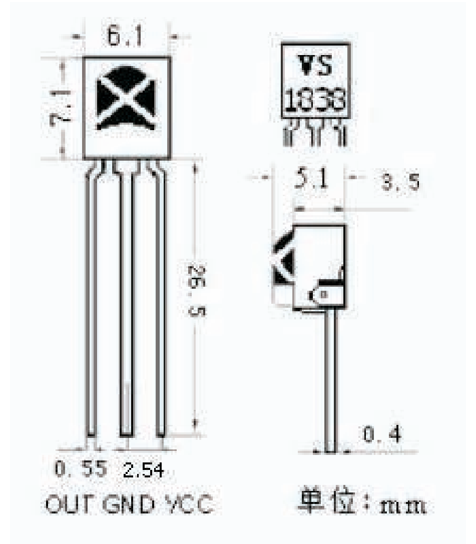
Infrared Receiver Module 红外线接收器

型号：VS1838B

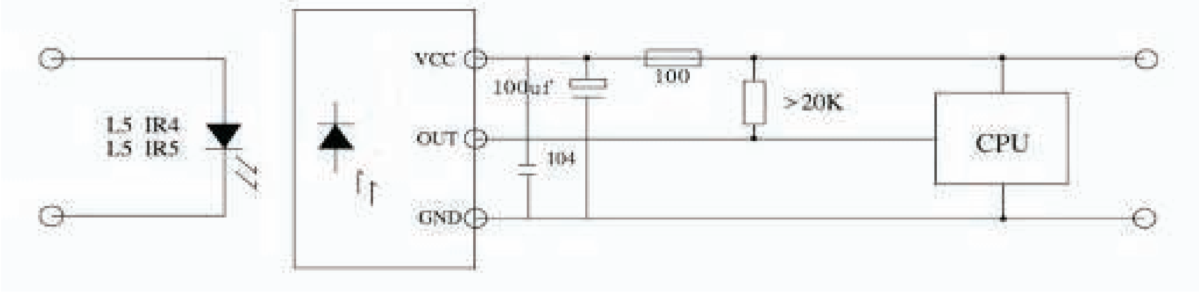
1. 特性
   - ●小型设计；
   - ●内置专用 IC；
   - ●宽角度及长距离接收；
   - ●抗干扰能力强；
   - ●能抵御环境光线干扰；
   - ●低电压工作；

2. 应用：
   - ■视听器材(音响，电视，录影机，碟机)
   - ■家用电器 （冷器机，电风扇，电灯）
   - ■其它无线电器遥控产品；

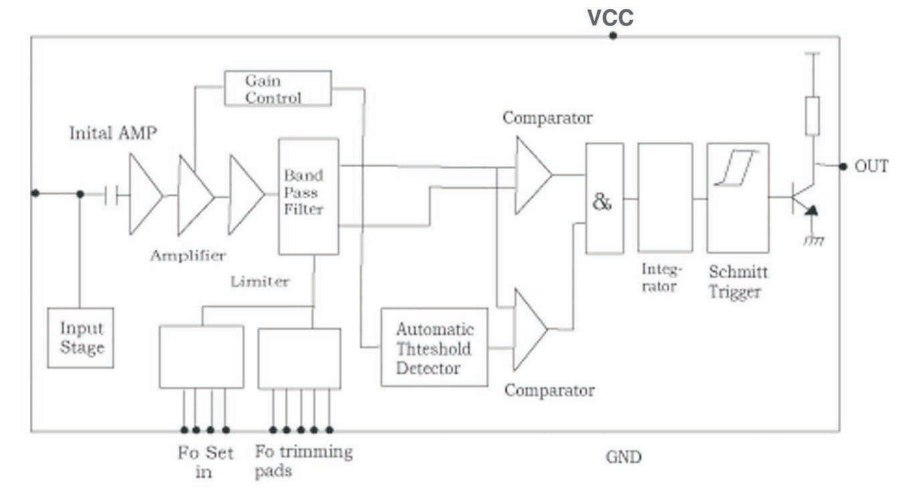
外型尺寸及引脚排列图



3. 应用电路图：



4. 原理图：



Infrared Receiver Module 红外线接收器

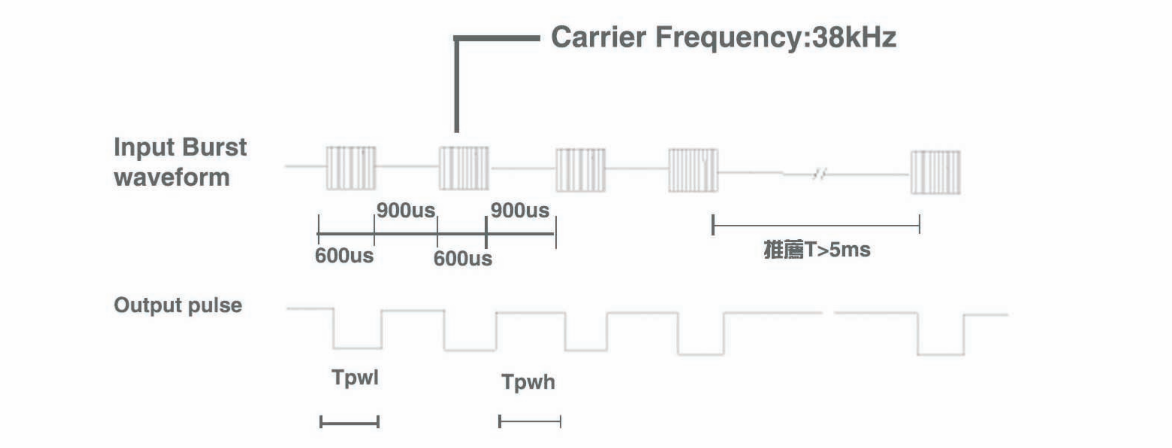
型号: VS1838B

光电参数(T=25℃　Vcc=5v　f_0=38KHZ)

参数	符号	测试条件	Min	Typ	Mnx	单位
工作电压	Vcc		2.7		5.5	V
接收距离	L	L5IR=300MA (测试信号)	18	20		M
载波频率	f_0		38K			HZ
接收角度	01/2	距离衰减 1/2		+/-45		Deg
BMP 宽度	F_{BW}	-3Db andwidth	2	3.3	5	kHz
静态电流	Icc	无信号输入时	----	0.4	1.5	mA
低电平输出	V_{OL}	Vin=0V Vcc=5V		0.2	0.4	V
高电平输出	V_{OH}	Vcc=5V	4.5			V
输出脉冲宽　度	T_{PWL}	Vin=500μVp-p※	500	600	700	μs
	T_{PWH}	Vin=50mVp-p※	500	600	700	μs

※光轴上测试,以宽度为 600/900μs 为发射脉冲,在 5CM 之接收范围内,取 50 次接收脉冲之平均值

5.测试波型:



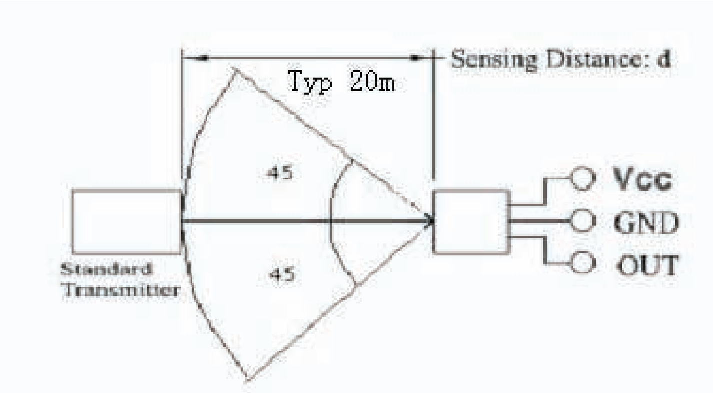
6.极限参数:

项目	符号	规格	单位
供应电压	Vcc	6.0	v
工作温度	Topr	-20-85	℃
储存温度	Tstg	-40-125	℃
焊接温度	Tsol	240	℃

Infrared Receiver Module 红外线接收器

型号：VS1838B

7.接收角度：



8.推荐使用条件：

项目	符号	Min	Typ	Mnx	单位
工作电压	Vcc	2.7	-----	5.5	V
输入频率	FM		38		kHz
工作温度	Topr	-20	25	80	℃

9.使用注意

1）在无任何外加压力及影响品质的环境下储存及使用；

2）在无污染性气体或海风（含盐份）的环境下储存及使用；

3）在低湿度环境下储存及使用；

4）在规定的条件下焊接引线管脚，焊接后，请勿施加外力；

5）请勿清洗本产品，使用前，请先用静电带将作业员及电烙铁连接落地线；

A.4 L298N H-Bridge Driver Board Data Sheet

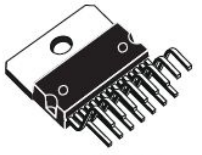
L298

DUAL FULL-BRIDGE DRIVER

- OPERATING SUPPLY VOLTAGE UP TO 46 V
- TOTAL DC CURRENT UP TO 4 A
- LOW SATURATION VOLTAGE
- OVERTEMPERATURE PROTECTION
- LOGICAL "0" INPUT VOLTAGE UP TO 1.5 V (HIGH NOISE IMMUNITY)

DESCRIPTION

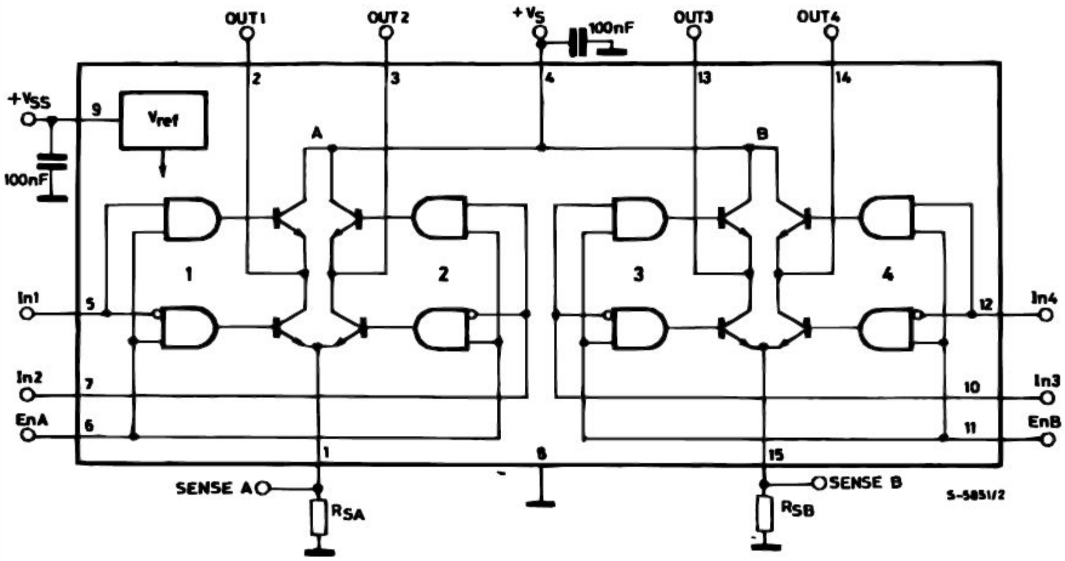
The L298 is an integrated monolithic circuit in a 15-lead Multiwatt and PowerSO20 packages. It is a high voltage, high current dual full-bridge driver designed to accept standard TTL logic levels and drive inductive loads such as relays, solenoids, DC and stepping motors. Two enable inputs are provided to enable or disable the device independently of the input signals. The emitters of the lower transistors of each bridge are connected together and the corresponding external terminal can be used for the connection of an external sensing resistor. An additional supply input is provided so that the logic works at a lower voltage.


Multiwatt15

PowerSO20

ORDERING NUMBERS : L298N (Multiwatt Vert.)
L298HN (Multiwatt Horiz.)
L298P (PowerSO20)

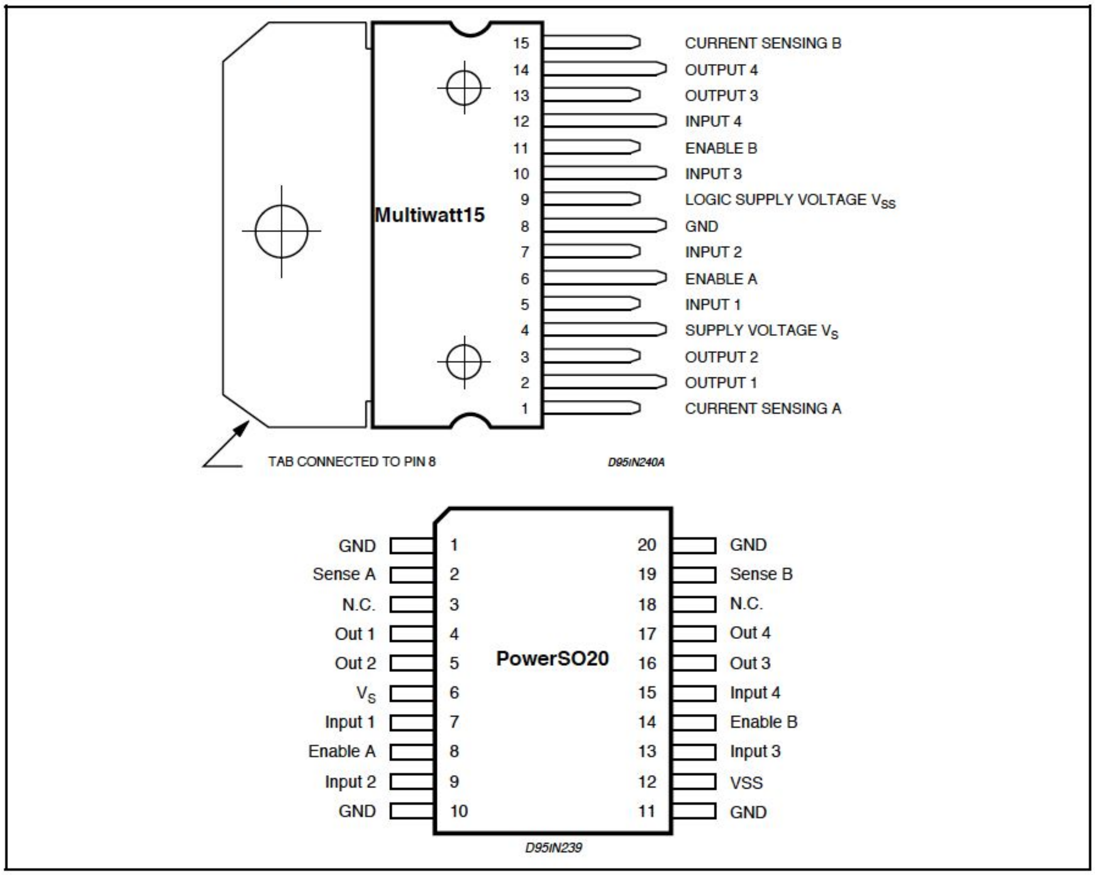
BLOCK DIAGRAM



ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_S	Power Supply	50	V
V_{SS}	Logic Supply Voltage	7	V
V_I,V_{en}	Input and Enable Voltage	–0.3 to 7	V
I_O	Peak Output Current (each Channel) – Non Repetitive (t = 100µs) –Repetitive (80% on –20% off; t_{on} = 10ms) –DC Operation	 3 2.5 2	 A A A
V_{sens}	Sensing Voltage	–1 to 2.3	V
P_{tot}	Total Power Dissipation (T_{case} = 75°C)	25	W
T_{op}	Junction Operating Temperature	–25 to 130	°C
T_{stg}, T_j	Storage and Junction Temperature	–40 to 150	°C

PIN CONNECTIONS (top view)



THERMAL DATA

Symbol	Parameter		PowerSO20	Multiwatt15	Unit
$R_{th\ j\text{-}case}$	Thermal Resistance Junction-case	Max.	–	3	°C/W
$R_{th\ j\text{-}amb}$	Thermal Resistance Junction-ambient	Max.	13 (*)	35	°C/W

PIN FUNCTIONS (refer to the block diagram)

MW.15	PowerSO	Name	Function
1;15	2;19	Sense A; Sense B	Between this pin and ground is connected the sense resistor to control the current of the load.
2;3	4;5	Out 1; Out 2	Outputs of the Bridge A; the current that flows through the load connected between these two pins is monitored at pin 1.
4	6	V_S	Supply Voltage for the Power Output Stages. A non-inductive 100nF capacitor must be connected between this pin and ground.
5;7	7;9	Input 1; Input 2	TTL Compatible Inputs of the Bridge A.
6;11	8;14	Enable A; Enable B	TTL Compatible Enable Input: the L state disables the bridge A (enable A) and/or the bridge B (enable B).
8	1,10,11,20	GND	Ground.
9	12	VSS	Supply Voltage for the Logic Blocks. A100nF capacitor must be connected between this pin and ground.
10; 12	13;15	Input 3; Input 4	TTL Compatible Inputs of the Bridge B.
13; 14	16;17	Out 3; Out 4	Outputs of the Bridge B. The current that flows through the load connected between these two pins is monitored at pin 15.
–	3;18	N.C.	Not Connected

ELECTRICAL CHARACTERISTICS (V_S = 42V; V_{SS} = 5V, T_j = 25°C; unless otherwise specified)

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
V_S	Supply Voltage (pin 4)	Operative Condition	V_{IH} +2.5		46	V
V_{SS}	Logic Supply Voltage (pin 9)		4.5	5	7	V
I_S	Quiescent Supply Current (pin 4)	V_{en} = H; I_L = 0 V_i = L V_i = H		13 50	22 70	mA mA
		V_{en} = L V_i = X			4	mA
I_{SS}	Quiescent Current from V_{SS} (pin 9)	V_{en} = H; I_L = 0 V_i = L V_i = H		24 7	36 12	mA mA
		V_{en} = L V_i = X			6	mA
V_{iL}	Input Low Voltage (pins 5, 7, 10, 12)		–0.3		1.5	V
V_{iH}	Input High Voltage (pins 5, 7, 10, 12)		2.3		VSS	V
I_{iL}	Low Voltage Input Current (pins 5, 7, 10, 12)	V_i = L			–10	µA
I_{iH}	High Voltage Input Current (pins 5, 7, 10, 12)	Vi = H ≤ V_{SS} –0.6V		30	100	µA
V_{en} = L	Enable Low Voltage (pins 6, 11)		–0.3		1.5	V
V_{en} = H	Enable High Voltage (pins 6, 11)		2.3		V_{SS}	V
I_{en} = L	Low Voltage Enable Current (pins 6, 11)	V_{en} = L			–10	µA
I_{en} = H	High Voltage Enable Current (pins 6, 11)	V_{en} = H ≤ V_{SS} –0.6V		30	100	µA
$V_{CEsat\,(H)}$	Source Saturation Voltage	I_L = 1A I_L = 2A	0.95	1.35 2	1.7 2.7	V V
$V_{CEsat\,(L)}$	Sink Saturation Voltage	I_L = 1A (5) I_L = 2A (5)	0.85	1.2 1.7	1.6 2.3	V V
V_{CEsat}	Total Drop	I_L = 1A (5) I_L = 2A (5)	1.80		3.2 4.9	V V
V_{sens}	Sensing Voltage (pins 1, 15)		–1 (1)		2	V

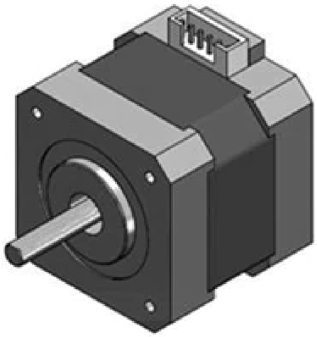
A.5 Nema 42 Stepper Motor Data Sheet

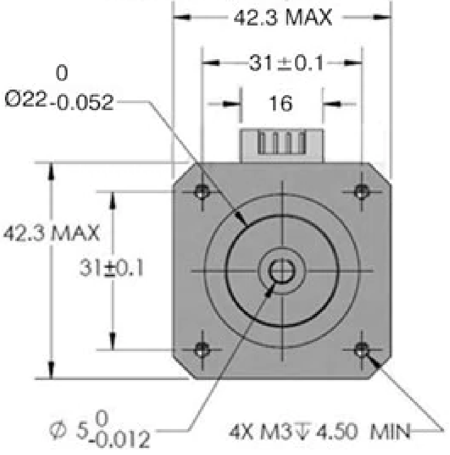
General Spec

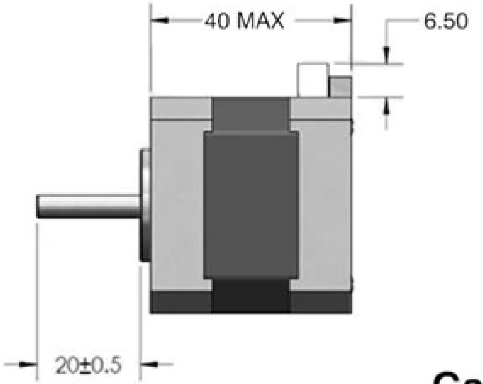
Step Angle	1.8°
Number of Phase	2
Insulation Resistance	100MΩ/min(500V DC)
Insulation Class	Class 8
Rotor Inertia	57g2/cm
Mass	0.25kg

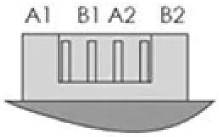
Electrical Spec

Rated Voltage	2V
Rated Current	1.2A
Resistance per Phase	1.7Ω±10±%
Inductance per Phase	4.5mH±20%
Holding Torque	400mN*m
Detent Torque	15mN*m









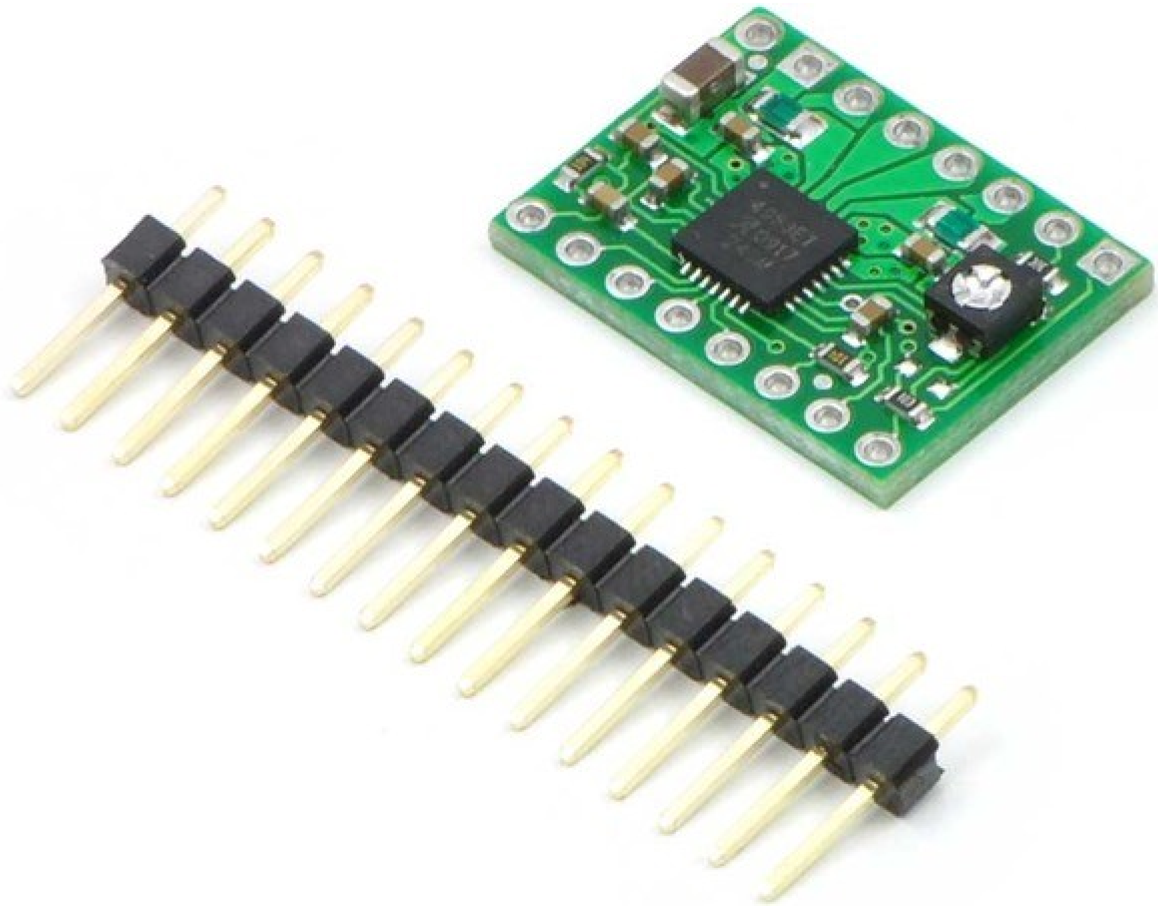
Cable Wiring

A1	Green
A2	Gray
B1	Yellow
B2	Red

A.6 A4988 Stepper Driver Board Data Sheet

RB-Pol-176

Pololu 8-35V 2A Single Bipolar Stepper Motor Driver A4988



A4988

Stepper Motor Driver Carrier

The A4988 stepper motor driver carrier is a breakout board for Allegro's easy-to-use A4988 microstepping bipolar stepper motor driver and is a drop-in replacement for the A4983 stepper motor driver carrier. The driver features adjustable current limiting, overcurrent protection, and five different microstep resolutions. It operates from 8 – 35 V and can deliver up to 2 A per coil.

Note: This board is a drop-in replacement for the original A4983 stepper motor driver carrier. The newer A4988 offers overcurrent protection and has an internal 100k pull-down on the MS1 microstep selection pin, but it is otherwise virtually identical to the A4983.

Description

Overview

This product is a carrier board or breakout board for Allegro's A4988 DMOS Microstepping Driver with Translator and Overcurrent Protection; we therefore recommend careful reading of the A4988 datasheet (380k pdf) before using this product. This stepper motor driver lets you control one bipolar stepper motor at up to 2 A output current per coil (see the Power Dissipation Considerations section below for more information). Here are some of the driver's key features:

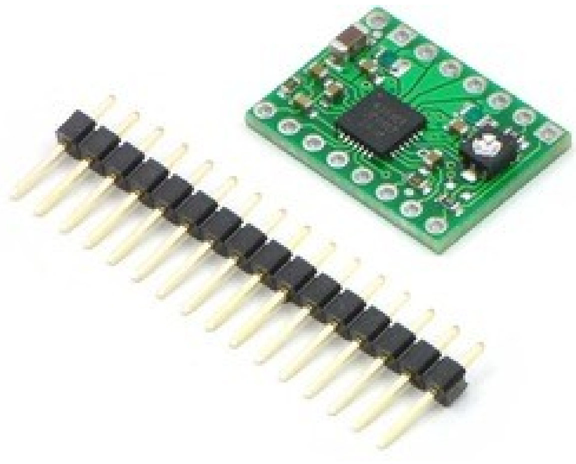
- Simple step and direction control interface
- Five different step resolutions: full-step, half-step, quarter-step, eighth-step, and sixteenth-step
- Adjustable current control lets you set the maximum current output with a potentiometer, which lets you use voltages above your stepper motor's rated voltage to achieve higher step rates
- Intelligent chopping control that automatically selects the correct current decay mode (fast decay or slow decay)
- Over-temperature thermal shutdown, under-voltage lockout, and crossover-current protection
- Short-to-ground and shorted-load protection (this feature is not available on the A4983)

Like nearly all our other carrier boards, this product ships with all surface-mount components —including the A4988 driver IC—installed as shown in the product picture.

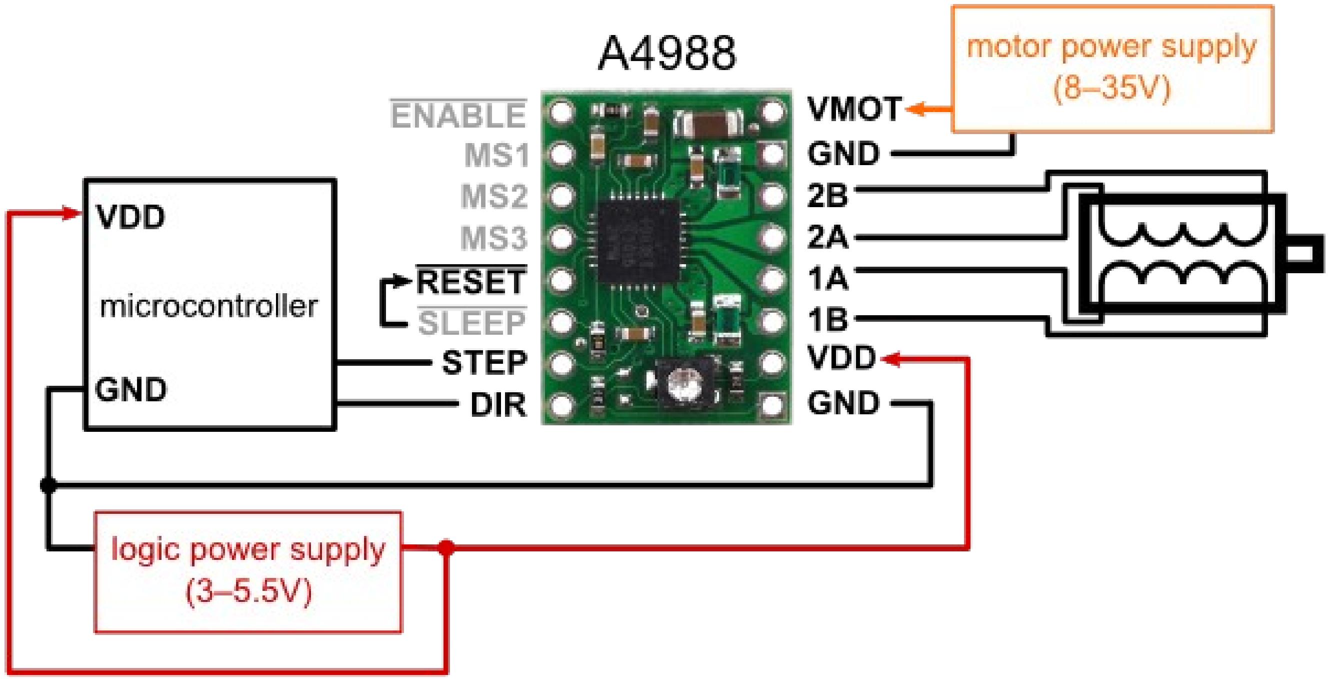
We also sell a larger version of the A4988 carrier that has reverse power protection on the main power input and built-in 5 V and 3.3 V voltage regulators that eliminate the need for separate logic and motor supplies.

Included hardware

The A4988 stepper motor driver carrier comes with one 1×16-pin breakaway 0.1" male header. The headers can be soldered in for use with solderless breadboards or 0.1" female connectors. You can also solder your motor leads and other connections directly to the board.



Using the driver



Minimal wiring diagram for connecting a microcontroller to an A4988 stepper motor driver carrier (full-step mode).

Power connections

The driver requires a logic supply voltage (3 – 5.5 V) to be connected across the VDD and GND pins and a motor supply voltage of (8 – 35 V) to be connected across VMOT and GND. These supplies should have appropriate decoupling capacitors close to the board, and they should be capable of delivering the expected currents (peaks up to 4 A for the motor supply).

Motor connections

Four, six, and eight-wire stepper motors can be driven by the A4988 if they are properly connected; a FAQ answer explains the proper wirings in detail.
Warning: Connecting or disconnecting a stepper motor while the driver is powered can destroy the driver. (More generally, rewiring anything while it is powered is asking for trouble.)

Warning: Connecting or disconnecting a stepper motor while the driver is powered can destroy the driver. (More generally, rewiring anything while it is powered is asking for trouble.)

Step (and microstep) size

Stepper motors typically have a step size specification (e.g. 1.8° or 200 steps per revolution), which applies to full steps. A microstepping driver such as the A4988 allows higher resolutions by allowing intermediate step locations, which are achieved by energizing the coils with intermediate current levels. For instance, driving a motor in quarter-step mode will give the 200-step-per-revolution motor 800 microsteps per revolution by using four different current levels.

The resolution (step size) selector inputs (MS1, MS2, MS3) enable selection from the five step resolutions according to the table below. MS1 and MS3 have internal 100kΩ pull-down resistors and MS2 has an internal 50kΩ pull-down resistor, so leaving these three microstep selection pins disconnected results in full-step mode. For the microstep modes to function correctly, the current limit must be set low enough (see below) so that current limiting gets engaged. Otherwise, the intermediate current levels will not be correctly maintained, and the motor will effectively operate in a full-step mode.

MS1	MS2	MS3	Microstep Resolution
Low	Low	Low	Full step
High	Low	Low	Half step
Low	High	Low	Quarter step
High	High	Low	Eighth step
High	High	High	Sixteenth step

Control inputs

Each pulse to the STEP input corresponds to one microstep of the stepper motor in the direction selected by the DIR pin. Note that the STEP and DIR pins are not pulled to any particular voltage internally, so you should not leave either of these pins floating in your application. If you just want rotation in a single direction, you can tie DIR directly to VCC or

GND. The chip has three different inputs for controlling its many power states: RST, SLP, and EN. For details about these power states, see the datasheet. Please note that the RST pin is floating; if you are not using the pin, you can connect it to the adjacent SLP pin on the PCB.

Current limiting

To achieve high step rates, the motor supply is typically much higher than would be permissible without active current limiting. For instance, a typical stepper motor might have a maximum current rating of 1 A with a 5Ω coil resistance, which would indicate a maximum motor supply of 5 V. Using such a motor with 12 V would allow higher step rates, but the current must actively be limited to under 1 A to prevent damage to the motor.

The A4988 supports such active current limiting, and the trimmer potentiometer on the board can be used to set the current limit. One way to set the current limit is to put the driver into full-step mode and to measure the current running through a single motor coil without clocking the STEP input. The measured current will be 0.7 times the current limit (since both coils are always on and limited to 70% in full-step mode). Please note that the current limit is dependent on the Vdd voltage.

Another way to set the current limit is to measure the voltage on the “ref” pin and to calculate the resulting current limit (the current sense resistors are 0.05Ω). The ref pin voltage is accessible on a via that is circled on the bottom silkscreen of the circuit board. See the A4988 datasheet for more information.

Power dissipation considerations

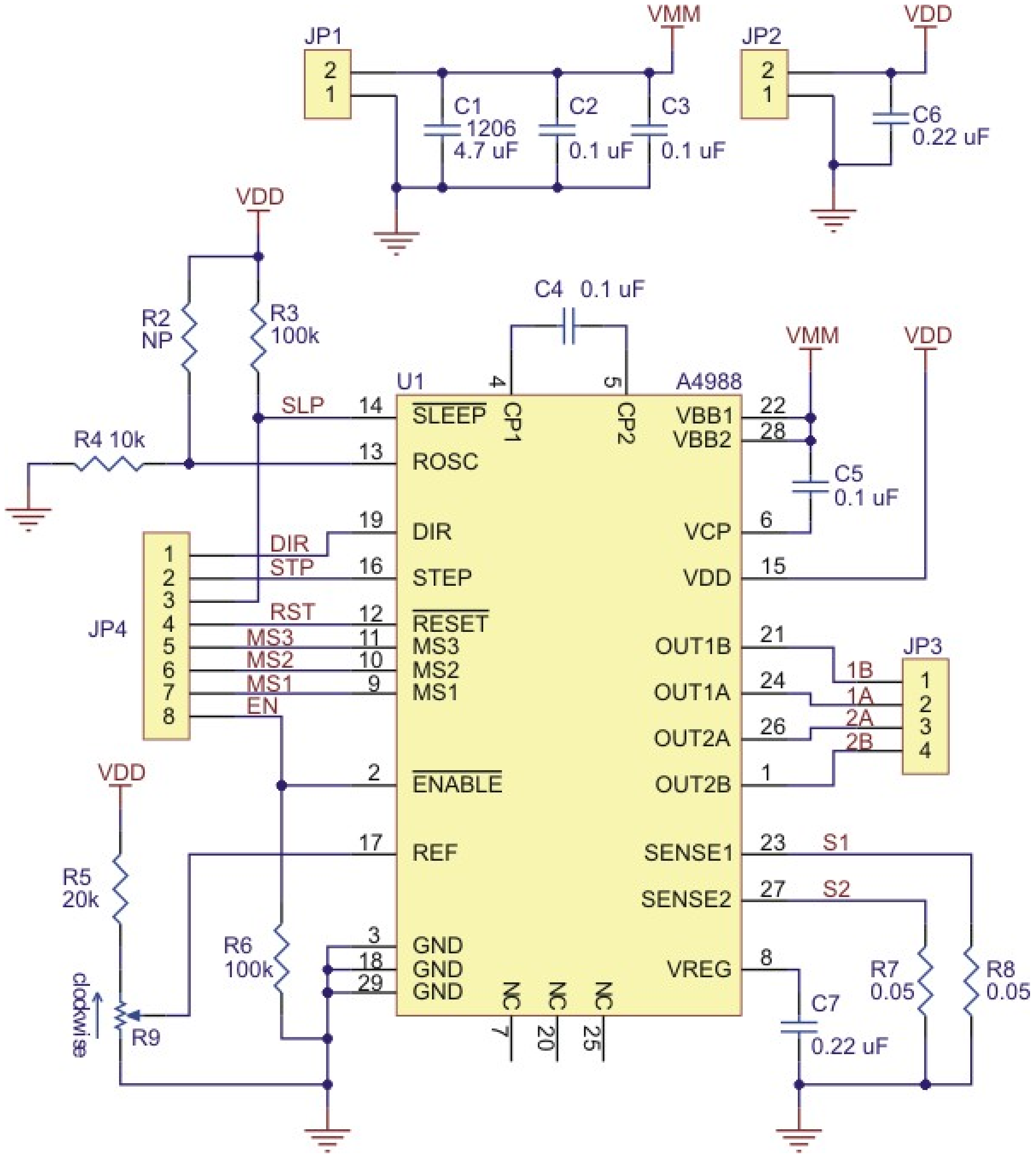
The A4988 driver IC has a maximum current rating of 2 A per coil, but the actual current you can deliver depends on how well you can keep the IC cool. The carrier’s printed circuit board is designed to draw heat out of the IC, but to supply more than approximately 1 A per coil, a heat sink or other cooling method is required.

This product can get hot enough to burn you long before the chip overheats. Take care when handling this product and other components connected to it.

Please note that measuring the current draw at the power supply does not necessarily provide an accurate measure of the coil current. Since the input voltage to the driver can be significantly higher than the coil voltage, the measured current on the power supply can be quite a bit lower than the coil current (the driver and coil basically act like a switching step-down power supply). Also, if the supply voltage is very high compared to what the motor needs to achieve the set current, the duty cycle will be very low, which also leads to significant differences between average and RMS currents.

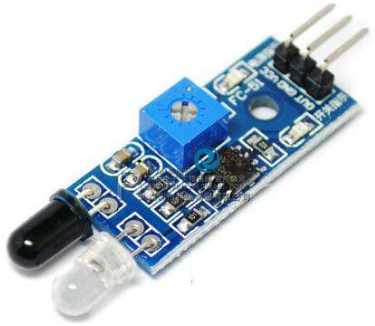
Schematic diagram

Schematic diagram of the md09b A4988 stepper motor driver carrier.



A.7 IR Proximity Sensor Data Sheet

Arduino IR Infrared Obstacle Avoidance Sensor Module



The sensor module adaptable to ambient light, having a pair of infrared emitting and receiving tubes, transmitting tubes emit infrared certain frequency, when the direction of an obstacle is detected (reflection surface), the infrared reflected is received by the reception tube, After a comparator circuit processing, the green light is on, but the signal output interface output digital signal (a low-level signal), you can adjust the detection distance knob potentiometer, the effective distance range of 2 ~ 30cm, the working voltage of 3.3V- 5V. Detection range of the sensor can be obtained by adjusting potentiometer, with little interference, easy to assemble, easy to use features, can be widely used in robot obstacle avoidance, avoidance car, line count, and black and white line tracking and many other occasions.

Specification

1. When the module detects an obstacle in front of the signal, the green indicator lights on the board level, while the OUT port sustained low signal output, the module detects the distance 2 ~ 30cm, detection angle 35 °, the distance can detect potential is adjusted clockwise adjustment potentiometer, detects the distance increases; counter clockwise adjustment potentiometer, reducing detection distance.

2. The sensor active infrared reflection detection, target reflectivity and therefore the shape is critical detection distance. Where the minimum detection distance black, white, maximum; small objects away from a small area, a large area from the Grand.

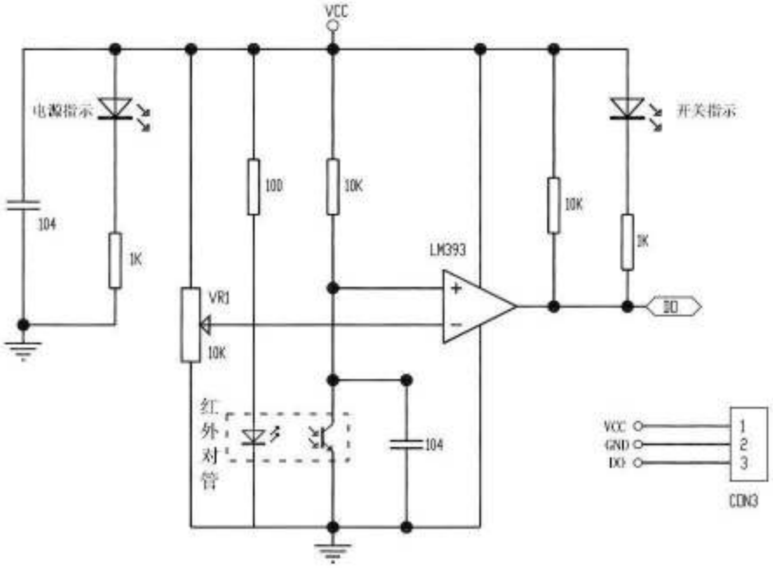
3. The sensor module output port OUT port can be directly connected to the microcontroller IO can also be directly drive a 5V relay; Connection: VCC-VCC; GND-GND; OUT-IO

4. Comparators LM393, stable;

5. The module can be 3-5V DC power supply. When the power is turned on, the red power indicator lights;
6. With the screw holes 3mm, easy fixed installation;
7. Board size: 3.2CM * 1.4CM
8. Each module has been shipped threshold comparator voltage adjusted by potentiometer good, non-special case, do not adjustable potentiometer.

Module Interface Description

1. VCC : 3.3V-5V external voltage (can be directly connected to 5v and 3.3v MCU)
2. GND : GND External
3. OUT : small board digital output interface (0 and 1)



Appendix B

PIC Microcontroller Main Program

```
// eeprom modify, 5 runs
#include <xc.h>
#include <stdio.h>
#include <stdbool.h>
#include "configBits.h"
#include "lcd.h"
#include "I2C.h"

typedef struct data {
    bool exist;
    bool full;
    bool received_ball;
    unsigned int distance_from_start;
} data;

typedef struct rtc {
    unsigned char year;
    unsigned char month;
    unsigned char day;
    unsigned char hour;
    unsigned char minute;
    unsigned char second;
} rtc;

typedef struct general {
    rtc timing;
    unsigned char op_time;
    unsigned char num_canister;
    unsigned char supplied_balls;
} general;


const char keys[] = "123A456B789C*0#D";
const int total_balls = 10;
const char time_setting[7] = {
    0x00, // Seconds
    0x52, // Minutes
    0x20, // Hour
    0x00, // Wed
    0x07, // Date
    0x04, // Month
    0x19 // Year
};

void __interrupt() pic_isr(void);
unsigned int encoder_to_distance(void);

//EEPROM READING/WRITING
unsigned char EEPROM_ReadByte(unsigned short address);
void EEPROM_WriteByte(unsigned short address, unsigned char data);
void ReadLog();
void WriteLog(rtc start_time, unsigned int op_time);
void Push_Back(rtc start_time, unsigned int op_time);

//MOTOR CONTROL
void stepperON(unsigned int n);
void stepperClockWise(void);
void stepperCounterClockWise(void);
void ball_to_right(void);
void ball_to_left(void);
void stepper_left(void);
void stepper_right(void);


//USER INTERFACE
int retrieve_data(int idx);
void interface(bool begin);
int main_menu(bool begin);
void invalid(void);
void no_such_canister(void);

//REAL TIME CLOCK
void getTime(void);
void rtc_set_time(void);

//operation
float side_us_sensor_distance_output(void);
void canister_operation(unsigned int position, float side_us_distance, bool left_ball,
    ↪ bool right_ball, unsigned int arduino_distance);
void ball_dispensing(bool facing_left);
void go_back(void);
unsigned int operation_time(rtc start_time);

float median(float *list);


rtc rtc_time; //store RTC
data history_info[5 * 10]; //store detailed canister data of last 5 runs
data canister_info[10]; //store data of current operation
general history_general[5]; //store general information of last 5 runs
volatile unsigned long encoder_count;
bool left_right = false;
```

```
void main(void) {
    //acquire data from EEPROM
    ReadLog();
    TRISA = 0x00;
    TRISEbits.RE0 = 1;
    TRISEbits.RE1 = 1;
    TRISC = 0b01111000;
    TRISD = 0x00; //LCD
    LATD = 0x00;
    LATCbits.LATC7 = 0;
    LATAbits.LATA4 = 0;
    LATAbits.LATA5 = 1; //stepper driver board disabled
    // Set all A/D ports to digital (pg. 222)
    ADCON1 = 0b00001111;

    //set up Arduino
    I2C_Master_Init(100000);
    I2C_Master_Start();
    I2C_Master_Write(0b00010000); // 7-bit Arduino slave address + write
    I2C_Master_Stop();

    //set up RTC
    I2C_Master_Start(); // Start condition
    I2C_Master_Write(0b11010000); // 7 bit RTC address + Write
    I2C_Master_Write(0x00); // Set memory pointer to seconds
    I2C_Master_Stop(); // Stop condition


    ///////////////////////////////////////////////////////////////////////
    //initialize variables

    unsigned int op_time = 30;
    unsigned int num_canister = 5;
    unsigned int supplied_balls = 2;
    unsigned int balls_left = total_balls - supplied_balls;
    unsigned int position = 0;


    //rtc_set_time();
    bool before_operation = true;
    interface(before_operation);
    //set port B output and input after user interface
    TRISB = 0b00110011;
    rtc start_time = rtc_time;



    // Set timer 2 prescaler to 16
    T2CKPS0 = 1;
    T2CKPS1 = 1;

    // Enable timer 2
    TMR2ON = 1;
```

```
lcd_clear();
printf("Operating...");
//keypad disable
LATCbits.LATC7 = 1;

//main loop
float ai_distance = 0;
unsigned int counter = 0;
bool left_ball = false;
bool right_ball = false;
unsigned char arduino_data = 0x00;
lcd_clear();

bool start_motor = true;

unsigned int accumulated_distance = 0;
float list[3];

RBPU = 0; // enable weak pullups
INTEDG0 = 1; // interrupt on rising edge
// enable INT0,INT1 interrupts
IPEN = 0;
INT0IF = 0;
INT0IE = 1;
PEIE = 1;
GIE = 1;

bool terminate = false;

while (1) {

    if (start_motor) {
        arduino_data = 1; //all forward
        // }
        //send data to arduino to move forward

        I2C_Master_Start(); // Start condition
        I2C_Master_Write(0b00010000); // 7-bit Arduino slave address + write
        I2C_Master_Write(arduino_data); // Write key press data
        I2C_Master_Stop();
        start_motor = false;
    }
    //get distance to the next canister from front US sensor

    //if (!shifted) {
    if (counter == 0) {
        while (!PORTEbits.RE0) { //green
            left_right = true;
            arduino_data = 0b00000000; //stop
            I2C_Master_Start(); // Start condition
            I2C_Master_Write(0b00010000); // 7-bit Arduino slave address + write
            I2C_Master_Write(arduino_data);
            I2C_Master_Stop();
```

```
                __delay_ms(100);

                arduino_data = 3; //shift right
                I2C_Master_Start(); // Start condition
                I2C_Master_Write(0b00010000); // 7-bit Arduino slave address + write
                I2C_Master_Write(arduino_data);
                I2C_Master_Stop();

                __delay_ms(100);

                arduino_data = 5; //stop
                I2C_Master_Start(); // Start condition
                I2C_Master_Write(0b00010000); // 7-bit Arduino slave address + write
                I2C_Master_Write(arduino_data);
                I2C_Master_Stop();
                __delay_ms(200);




            }
            while (!PORTEbits.RE1) {//yellow
                left_right = true;
                arduino_data = 0b00000000; //stop
                I2C_Master_Start(); // Start condition
                I2C_Master_Write(0b00010000); // 7-bit Arduino slave address + write
                I2C_Master_Write(arduino_data);
                I2C_Master_Stop();
                __delay_ms(100);

                arduino_data = 4; //shift right
                I2C_Master_Start(); // Start condition
                I2C_Master_Write(0b00010000); // 7-bit Arduino slave address + write
                I2C_Master_Write(arduino_data);
                I2C_Master_Stop();

                __delay_ms(100);

                arduino_data = 5; //stop
                I2C_Master_Start(); // Start condition
                I2C_Master_Write(0b00010000); // 7-bit Arduino slave address + write
                I2C_Master_Write(arduino_data);
                I2C_Master_Stop();
                __delay_ms(200);

                //shifted = true;
            }
        }
        if (left_right) {
            arduino_data = 1;
            I2C_Master_Start(); // Start condition
            I2C_Master_Write(0b00010000); // 7-bit Arduino slave address + write
            I2C_Master_Write(arduino_data); // Write key press data
            I2C_Master_Stop();
            left_right = false;
        }

        //}

        float side_us_distance = side_us_sensor_distance_output();
        // if (!PORTCbits.RC5) { //right
        // right_ball = true;
        // }
        //lcd_clear();
        //printf("D = %f cm", side_us_distance);




        //break main loop if 10 canisters visited
        if (operation_time(start_time) >= 170) {
            arduino_data = 0b00000000; //stop
            I2C_Master_Start(); // Start condition
            I2C_Master_Write(0b00010000); // 7-bit Arduino slave address + write
            I2C_Master_Write(arduino_data);
            I2C_Master_Stop();
            printf("time");
            terminate = true;
            break;
        } else if (accumulated_distance >= 430) {
            printf("dis");
            __delay_ms(100);

            break; //break if travel more than 400 cm
            //go back
        } else if ((side_us_distance <= 30) && (side_us_distance > 0)) { //canister in
            ↪ range detected

            list[counter] = side_us_distance;
            counter = counter + 1;

            if (counter >= 3) {
                //request distance at arrival
                unsigned int arrival_distance = encoder_to_distance();
                //spin flag
                LATAbits.LATA4 = 1;
                bool flag_on = true;
                counter = 0;
                float mean_distance = median(list);
                unsigned int right_ball_count = 0;
                unsigned int left_ball_count = 0;
                while (1) {
                    unsigned int moving_distance = encoder_to_distance();
                    // lcd_clear();
                    // printf("MD = %d",moving_distance);
                    if (position == 0) {
                        if ((moving_distance - arrival_distance) >= 25) {
                            break;
                        }
                    } else {
                        if ((moving_distance - arrival_distance) >= 24) {
                            break;
                        }
                    }
                    if ((moving_distance - arrival_distance) >= 15 && flag_on) {
                        LATAbits.LATA4 = 0;
                        flag_on = false;
                    }


                    if (!PORTCbits.RC5) { //right
                        //left_ball = true;
                        right_ball_count++;
                    }

                    if (!PORTCbits.RC6) { //left
                        //left_ball = true;
                        left_ball_count++;
                    }

                }

                if (left_ball_count >= 150) {
                    left_ball = true;
                } else {
                    left_ball = false;
                }

                if (right_ball_count >= 200) {
                    right_ball = true;
                } else {
                    right_ball = false;
                }
                //LATBbits.LATB1 = 1;

                arduino_data = 0b00000000; //stop
                I2C_Master_Start(); // Start condition
                I2C_Master_Write(0b00010000); // 7-bit Arduino slave address + write
                I2C_Master_Write(arduino_data);
                I2C_Master_Stop();

// lcd_clear();
// lcd_set_ddram_addr(LCD_LINE3_ADDR);
// //printf("found!!!");
// printf("L count = %d", left_ball_count);
// lcd_set_ddram_addr(LCD_LINE4_ADDR);
// printf("R count = %d", right_ball_count);
                __delay_ms(100);

                canister_operation(position, mean_distance, left_ball, right_ball,
                    ↪ arrival_distance);
                lcd_clear();
                position = position + 1;
                left_ball = false;
                right_ball = false;
                start_motor = true;

            }


        }

        //request data from arduino to get distance travelled
        accumulated_distance = encoder_to_distance();

        // lcd_set_ddram_addr(LCD_LINE4_ADDR);
        //
        // printf("TD: %d", accumulated_distance);

    }

    printf("exit");



    if (!terminate) {
        go_back();
    }


    //write canister doesn't exist to the rest of memory
    while (1) {
        if (position >= 10) {
            break;
        }
        canister_info[position].exist = false;
        position = position + 1;
    }
    lcd_clear();
    ///////////////////////////////////////////////////////////////////
    //the end
    LATCbits.LATC7 = 0; //keypad enable

    op_time = operation_time(start_time);
    WriteLog(start_time, op_time);
    TRISB = 0xFF;
    Push_Back(start_time, op_time);
    interface(!before_operation);
    lcd_clear();
    printf("Bye␣Bye");
    while (1);

}

void __interrupt() pic_isr(void) {
    if (INT0IF) {
        INT0IF = 0;
        // toggle active edge
        //volatile ++
        if (!left_right) {
            encoder_count++;
        }

    }


}

unsigned int encoder_to_distance(void) {
    return encoder_count / 20.907;
}

//EEPROM READ/WRITE

unsigned char EEPROM_ReadByte(unsigned short address) {
    //set address of EEPROM
    EEADRH = address >> 8;
    EEADR = address;

    EECON1bits.EEPGD = 0; //select data
    EECON1bits.CFGS = 0; //access
    EECON1bits.RD = 1; //start reading

    while (EECON1bits.RD == 1);

    return EEDATA; //return data
}

void EEPROM_WriteByte(unsigned short address, unsigned char data) {

    while (EECON1bits.WR); // Waits Until Last Attempt To Write Is Finished
    EEADRH = address >> 8;
    EEADR = address;

    EEDATA = data; //write data to EEDATA
    EECON1bits.EEPGD = 0; // Cleared To Point To EEPROM Not The Program Memory
    EECON1bits.WREN = 1; // Enable The Operation !
    INTCONbits.GIE = 0; // Disable All Interrupts Untill Writting Data Is Done
    EECON2 = 0x55; // Part Of Writing Mechanism..
    EECON2 = 0xAA; // Part Of Writing Mechanism..
    EECON1bits.WR = 1; // Part Of Writing Mechanism..
    INTCONbits.GIE = 1; // Re-Enable Interrupts
    EECON1bits.WREN = 0; // Disable The Operation
    EECON1bits.WR = 0; // Ready For Next Writting Operation
}

void ReadLog() {
    //oldest operation (0)
    //line 1: month
    //line 2: date
    //line 3: hour
    //line 4: minute
    //line 5: duration in seconds
    //line 6: balls supplied
    //line 7: canister 1 information
    // bit 0: exist(1) not exist(0)
    // bit 1: full(1) empty(0)
    // bit 2: ball received(1) not received(0)
    // bit 3: distance highest bit
    // bit 4-7: unused
    //line 8: distance
    //line 9,10: canister 2 information
    //line 11,12: canister 3 information
    //...
    //line 25,26: canister 10 information

    //line 27-52: operation 1
    //line 53-78: operation 2
    //line 79-104: operation 3
    //line 105-130: operation 4
    int i = 0;
    for (i = 0; i < 5; i = i + 1) {
        history_general[i].timing.month = EEPROM_ReadByte(i * 26 + 1);
        history_general[i].timing.day = EEPROM_ReadByte(i * 26 + 2);
        history_general[i].timing.hour = EEPROM_ReadByte(i * 26 + 3);
        history_general[i].timing.minute = EEPROM_ReadByte(i * 26 + 4);
        history_general[i].op_time = EEPROM_ReadByte(i * 26 + 5);
        history_general[i].supplied_balls = EEPROM_ReadByte(i * 26 + 6);
        int j = (26 * i) + 7;
        int canister_num = 0;
        for (; j <= 26 * (i + 1); j = j + 2) {
            unsigned char basic = EEPROM_ReadByte(j);
            unsigned char dist = EEPROM_ReadByte(j + 1);
            history_info[i * 10 + canister_num].exist = (basic & (0x01));
            history_info[i * 10 + canister_num].full = ((basic >> 1)&(0x01));
            history_info[i * 10 + canister_num].received_ball = ((basic >> 2)&(0x01));
            unsigned int full_distance = basic & (0b00001000);
            full_distance = (full_distance << 5) | dist;
            history_info[i * 10 + canister_num].distance_from_start = full_distance;
            canister_num = canister_num + 1;
        }

    }
}

void WriteLog(rtc start_time, unsigned int op_time) {
    //delete the earliest entry and move the other three forward
    int i = 0;
    int j = 0;

    for (i = 0; i < 4; i = i + 1) {
        EEPROM_WriteByte(i * 26 + 1, history_general[i + 1].timing.month);
        EEPROM_WriteByte(i * 26 + 2, history_general[i + 1].timing.day);
        EEPROM_WriteByte(i * 26 + 3, history_general[i + 1].timing.hour);
        EEPROM_WriteByte(i * 26 + 4, history_general[i + 1].timing.minute);
        EEPROM_WriteByte(i * 26 + 5, history_general[i + 1].op_time);
        EEPROM_WriteByte(i * 26 + 6, history_general[i + 1].supplied_balls);
        j = (26 * i) + 7;
        unsigned int canister_num = 0;
        for (; j <= 26 * (i + 1); j = j + 2) {
            unsigned int full_distance = canister_info[(i + 1)*10 + canister_num].
                ↪ distance_from_start;
            full_distance = full_distance >> 8;
            unsigned char basic = history_info[(i + 1)*10 + canister_num].exist |
                ↪ canister_info[(i + 1)*10 + canister_num].full << 1 | canister_info[(i +
                ↪  1)*10 + canister_num].received_ball << 2 | full_distance << 3;
            EEPROM_WriteByte(j, basic);
            EEPROM_WriteByte(j + 1, history_info[(i + 1)*10 + canister_num].
                ↪ distance_from_start);
            canister_num++;
        }
    }


    // for (i = 0; i < 3; i = i + 1) {
    // for (j = 1; j <= 26; j = j + 1) {
    // EEPROM_WriteByte(i * 26 + j, EEPROM_ReadByte((i + 1)*26 + j));
    // }
    // }

    //determine supplied balls
    unsigned char supplied = 0;
    for (i = 0; i < 10; i = i + 1) {
        if (canister_info[i].exist) {
            if (canister_info[i].received_ball) {
                supplied = supplied + 1;
            }
        } else {
            break;
        }
    }

    EEPROM_WriteByte(4 * 26 + 1, start_time.month);
    EEPROM_WriteByte(4 * 26 + 2, start_time.day);
    EEPROM_WriteByte(4 * 26 + 3, start_time.hour);
    EEPROM_WriteByte(4 * 26 + 4, start_time.minute);
    EEPROM_WriteByte(4 * 26 + 5, op_time);
    EEPROM_WriteByte(4 * 26 + 6, supplied);
    j = 0;
    for (i = 4 * 26 + 7; i <= 26 * 5; i = i + 2) {
        unsigned int full_distance = canister_info[j].distance_from_start;
        full_distance = full_distance >> 8;
        unsigned char basic = canister_info[j].exist | canister_info[j].full << 1 |
            ↪ canister_info[j].received_ball << 2 | full_distance << 3;
        EEPROM_WriteByte(i, basic);
        EEPROM_WriteByte(i + 1, canister_info[j].distance_from_start);
        j = j + 1;
    }


}

void Push_Back(rtc start_time, unsigned int op_time) {
    for (int i = 0; i < 4; i++) {
        history_general[i] = history_general[i + 1];
        for (int j = 0; j < 10; j++) {
            history_info[i * 10 + j] = history_info[(i + 1)*10 + j];
        }
    }

    unsigned char supplied = 0;
    for (int i = 0; i < 10; i = i + 1) {
        if (canister_info[i].exist) {
            if (canister_info[i].received_ball) {
                supplied = supplied + 1;
            }
        } else {
            break;
        }
    }

    history_general[4].op_time = op_time;
    history_general[4].supplied_balls = supplied;
    history_general[4].timing = start_time;

    for (int i = 0; i < 10; i++) {
        history_info[40 + i] = canister_info[i];
    }

}

//MOTOR CONTROL

void stepperON(unsigned int n) {

    for (unsigned int i = 0; i < n; i = i + 1) {
        LATAbits.LA3 = 1;
        __delay_ms(0.7);
        LATAbits.LA3 = 0;
        __delay_ms(0.7);
    }
}

void stepperClockWise(void) {
    LATAbits.LA1 = 1;
}

void stepperCounterClockWise(void) {
    LATAbits.LA1 = 0;
}
```

```
void ball_to_right(void) {
    LATAbits.LATA5 = 0; //stepper enabled
    stepperCounterClockWise(); //ground is left
    stepperON(1950);
    __delay_ms(800);
    stepperClockWise();
    stepperON(2650);
    __delay_ms(200);
    stepperCounterClockWise();
    stepperON(700);
    LATAbits.LATA5 = 1;
}

void ball_to_left(void) {
    LATAbits.LATA5 = 0; //stepper enabled
    stepperClockWise();
    stepperON(1950);
    __delay_ms(800);
    stepperCounterClockWise();
    stepperON(2650);
    __delay_ms(200);
    stepperClockWise();
    stepperON(700);
    LATAbits.LATA5 = 1;
}

void stepper_left(void) {
    LATAbits.LATA5 = 0; //stepper enabled
    stepperClockWise();
    stepperON(1950);
    LATAbits.LATA5 = 1;
}

void stepper_right(void) {
    LATAbits.LATA5 = 0; //stepper enabled
    stepperCounterClockWise();
    stepperON(1950);
    LATAbits.LATA5 = 1;
}

//USER INTERFACE

void invalid(void) {
    lcd_clear();
    printf("Invalid␣Input");
    __delay_ms(1000);
}

void no_such_canister(void) {
    lcd_clear();
    printf("␣␣␣␣NO␣SUCH");
    lcd_set_ddram_addr(LCD_LINE2_ADDR);
    printf("␣␣␣CANISTER");
    __delay_ms(1000);

    lcd_clear();
    printf("USE␣KEYPAD␣TO");
    lcd_set_ddram_addr(LCD_LINE2_ADDR);
    printf("SELECT␣SPECIFIC␣");
    lcd_set_ddram_addr(LCD_LINE3_ADDR);
    printf("CANISTER");
}

void interface(bool before_operation) {
    int idx = 4;
    int direction = 0;
    int status = main_menu(before_operation); //acquire status from main menu
    while (1) {
        if (status == 0) {
            //if status is false, retrieve the data from log
            direction = retrieve_data(idx); //direction determines scrolling left or right
            if (direction == 1) { //if direction is 1, go left
                idx = idx - 1;
                if (idx < 0) {
                    invalid();
                    idx = idx + 1;
                } else {
                    lcd_clear();

                    lcd_set_ddram_addr(LCD_LINE2_ADDR);
                    printf("␣␣<-------");
                    __delay_ms(500);
                }

            } else if (direction == 0) { //if direction is 0, go right
                idx = idx + 1;
                if (idx >= 5) {
                    invalid();
                    idx = idx - 1;
                } else {
                    lcd_clear();

                    lcd_set_ddram_addr(LCD_LINE2_ADDR);
                    printf("␣␣------->");
                    __delay_ms(500);
                }

            } else if (direction == 2) {
                continue;
            } else if (direction == 3) { //if direction is 3, exit detailed data and go
                ↪ back to main menu
                status = main_menu(before_operation);
            }

        } else if (status == 1) {
            lcd_clear();
            // exit interface to run the operation when status is true
            break;
        }
    }
}

int retrieve_data(int idx) {
    int direction = 0;

    while (1) {
        lcd_clear();
        //print the time
        printf("%d.␣%02x/%02x␣␣%02x:%02x", idx, history_general[idx].timing.month,
            ↪ history_general[idx].timing.day, history_general[idx].timing.hour,
            ↪ history_general[idx].timing.minute);
        lcd_set_ddram_addr(LCD_LINE2_ADDR);
        printf("%ds␣OPERATED", history_general[idx].op_time);
        lcd_set_ddram_addr(LCD_LINE3_ADDR);
        printf("%d␣BALLS␣SUPPLIED", history_general[idx].supplied_balls);
        lcd_set_ddram_addr(LCD_LINE4_ADDR);
        printf("%d␣BALLS␣LEFT", 10 - history_general[idx].supplied_balls);

        //read keypad
        while (PORTBbits.RB1 == 0) {
            continue;
        }
        unsigned char keypress = (PORTB & 0xF0) >> 4;
        while (PORTBbits.RB1 == 1) {
            continue;
        }

        //Nop(); // Apply breakpoint here to prevent compiler optimizations

        unsigned char temp = keys[keypress];

        int selected = temp - '0';
        if (temp == 'A') {
            continue;
        } else if (temp == 'D') {//going back
            return 3;
        } else if (temp == 'B') { //b is scrolling left
            if ((idx - 1) < 0) {
                invalid();
            } else {
                return 1;
            }
        } else if (temp == 'C') { //c is scrolling right
            if ((idx + 1) >= 5) {
                invalid();
            } else {
                return 0;
            }
        } else if (temp == '#') { // view detailed info
            lcd_clear();
            printf("USE␣KEYPAD␣TO");
            lcd_set_ddram_addr(LCD_LINE2_ADDR);
            printf("SELECT␣SPECIFIC␣");
            lcd_set_ddram_addr(LCD_LINE3_ADDR);
            printf("CANISTER");

            while (1) {
                //read keypad
                while (PORTBbits.RB1 == 0) {
                    continue;
                }
                keypress = (PORTB & 0xF0) >> 4;

                while (PORTBbits.RB1 == 1) {
                    continue;
                }

                Nop(); // Apply breakpoint here to prevent compiler optimizations

                temp = keys[keypress];
                selected = temp - ’0’;
                if (selected == 0) {
                    if (!history_info[idx * 10 + 9].exist) {
                        no_such_canister();
                    } else {
                        lcd_clear();
                        printf("CANISTER␣10");
                        lcd_set_ddram_addr(LCD_LINE2_ADDR);
                        if (!history_info[idx * 10 + 9].full) {
                            printf("Empty␣Canister");
                        } else {
                            printf("Full␣Canister");
                        }
                        lcd_set_ddram_addr(LCD_LINE3_ADDR);
                        if (history_info[idx * 10 + 9].received_ball) {
                            printf("Ball␣Received");
                        } else {
                            printf("Not␣Supplied");
                        }
                        lcd_set_ddram_addr(LCD_LINE4_ADDR);
                        printf("Distance:␣%d␣cm", (history_info[idx * 10 + 9].
                            ↪ distance_from_start));
                    }
                } else if ((selected >= 1)&&(selected <= 9)) {
                    if (!history_info[idx * 10 + (selected - 1)].exist) {
                        no_such_canister();
                    } else {
                        lcd_clear();
                        printf("CANISTER␣#%d", selected);
                        lcd_set_ddram_addr(LCD_LINE2_ADDR);
                        if (!history_info[idx * 10 + (selected - 1)].full) {
                            printf("Empty␣Canister");
                        } else {
                            printf("Full␣Canister");
                        }
                        lcd_set_ddram_addr(LCD_LINE3_ADDR);
                        if (history_info[idx * 10 + (selected - 1)].received_ball) {
                            printf("Ball␣Received");
                        } else {
                            printf("Not␣Supplied");
                        }
                        lcd_set_ddram_addr(LCD_LINE4_ADDR);
                        printf("Distance:␣%d␣cm", (history_info[idx * 10 + selected - 1].
                            ↪ distance_from_start));
                    }
                } else if (temp == 'D') {//going back
                    return 2;
                }


            }
        } else {
            invalid();

        }

    }
    return direction;

}

int main_menu(bool begin) {
    int status = 0;
    initLCD();


    // Main loop
    while (1) {
        lcd_clear();
        printf("␣␣␣BALL␣BALL␣U");
        lcd_set_ddram_addr(LCD_LINE2_ADDR);
        if (begin) {
            printf("PRESS␣*␣TO␣START");
        } else {
            printf("␣TASK␣COMPLETE");
        }
        lcd_set_ddram_addr(LCD_LINE3_ADDR);
        printf("␣␣␣␣");
        getTime();
        //print current time
        printf("%02x/", rtc_time.year);

        printf("%02x/", rtc_time.month);

        printf("%02x", rtc_time.day);
        lcd_set_ddram_addr(LCD_LINE4_ADDR);
        printf("␣␣␣␣");
        printf("%02x:", rtc_time.hour);

        printf("%02x:", rtc_time.minute);

        printf("%02x", rtc_time.second);

        __delay_ms(1000);

        unsigned char keypress;

        I2C_Master_Start();
        I2C_Master_Write(0b00010001); // 7-bit Arduino slave address + Read
        unsigned char flag = I2C_Master_Read(NACK); // Read one char only
        I2C_Master_Stop();

        if (begin) {

            if (flag == 254) {
                status = 1;
                break;
            }

        }
        if (flag == 2) {
            stepper_right();
        } else if (flag == 3) {
            stepper_left();
        }

        if (PORTBbits.RB1) {
            keypress = (PORTB & 0xF0) >> 4;
            unsigned char temp = keys[keypress];
            if (temp == ’*’) {
                status = 1;
                break;
            } else if (temp == ’A’) {
                status = 0;
                break;

            } else {
                continue;
            }
        }

    }
    return status;
}

//REAL TIME CLOCK

void getTime(void) {
    unsigned char time[7];

    I2C_Master_Start(); // Start condition
    I2C_Master_Write(0b11010000); // 7 bit RTC address + Write
    I2C_Master_Write(0x00); // Set memory pointer to seconds
    I2C_Master_Stop(); // Stop condition


    // Read current time
    I2C_Master_Start(); // Start condition
    I2C_Master_Write(0b11010001); // 7 bit RTC address + Read
    for (unsigned char i = 0; i < 6; i++) {

        time[i] = I2C_Master_Read(ACK); // Read with ACK to continue reading

    }

    time[6] = I2C_Master_Read(NACK); // Final Read with NACK

    I2C_Master_Stop();

    rtc_time.year = time[6];
    rtc_time.month = time[5];
    rtc_time.day = time[4];

    rtc_time.hour = time[2];
    rtc_time.minute = time[1];
    rtc_time.second = time[0];
}

void rtc_set_time(void) {
    //this function is to set real time clock to current time
    I2C_Master_Start(); // Start condition
    I2C_Master_Write(0b11010000); //7 bit RTC address + Write
    I2C_Master_Write(0x00); // Set memory pointer to seconds

    // Write array
    for (char i = 0; i < 7; i++) {
        I2C_Master_Write(time_setting[i]);
    }

    I2C_Master_Stop(); //Stop condition
}


//OPERATION

float side_us_sensor_distance_output(void) {
    lcd_set_ddram_addr(LCD_LINE2_ADDR);

    TMR1H = 0; //Sets the Initial Value of Timer
    TMR1L = 0; //Sets the Initial Value of Timer
    LATBbits.LATB7 = 1; //TRIGGER HIGH
    __delay_us(10);
    LATBbits.LATB7 = 0; //TRIGGER LOW

    while (!PORTBbits.RB5);
    TMR1ON = 1;
    //Waiting for Echo
    //Timer Starts
    while (PORTBbits.RB5);

    TMR1ON = 0;
    //Waiting for Echo goes LOW
    //Timer Stops

    float front_us_distance = (TMR1L | (TMR1H << 8)); //Reads Timer Value
    front_us_distance = front_us_distance / 58.82; //Converts Time to Distance
    front_us_distance = front_us_distance / 2;
    return front_us_distance;
}

void canister_operation(unsigned int position, float side_us_distance, bool left_ball,
    ↪ bool right_ball, unsigned int arduino_distance) {

    unsigned char arduino_data = 0;
    //1: adjust to left
    //2: adjust to right

    bool supplied_ball = false;
    bool canister_full = false;
    bool facing_left = true;

    //spin flag to indicate canister found

    //__delay_ms(1000);

    canister_info[position].exist = true;

    canister_info[position].distance_from_start = arduino_distance;
    // if (position == 0) {
    // canister_info[position].distance_from_start = arduino_distance;
    // } else {
    // canister_info[position].distance_from_start = (arduino_distance / 2) - 2;
    // }


    //calculate distance from encoder

    // lcd_clear();
    // printf("US = %f cm", side_us_distance);
    // lcd_set_ddram_addr(LCD_LINE2_ADDR);
    // printf("DIST = %d cm", arduino_distance);
    // lcd_set_ddram_addr(LCD_LINE3_ADDR);
    // printf("LEFT = %d ", left_ball);
    // lcd_set_ddram_addr(LCD_LINE4_ADDR);
    // printf("RIGHT = %d", right_ball);
    // __delay_ms(1000);

    unsigned int distance_between = 0;
```

```
if (position == 0) {
    distance_between = 50;
} else {
    distance_between = arduino_distance - (canister_info[position - 1].
        ↪ distance_from_start);
}



// determine the opening of canister
//lcd_clear();
if (side_us_distance > 17.5) { //open oppposite sensor (left)


    //printf("facing left");
    facing_left = true;
    //__delay_ms(500);
    //check left IR
    if (left_ball) {
        //there is ball already
        canister_full = true;
    } else {
        if (distance_between >= 30) {
            canister_full = true;
            supplied_ball = true;
            //ready to supply ball

            //shift right
            if (side_us_distance >= 25) {
                left_right = true;
                arduino_data = 3; //shift right
                I2C_Master_Start(); // Start condition
                I2C_Master_Write(0b00010000); // 7-bit Arduino slave address + write
                I2C_Master_Write(arduino_data);
                I2C_Master_Stop();

                __delay_ms(100);


                arduino_data = 5; //stop
                I2C_Master_Start(); // Start condition
                I2C_Master_Write(0b00010000); // 7-bit Arduino slave address + write
                I2C_Master_Write(arduino_data);
                I2C_Master_Stop();
                __delay_ms(200);
            }
            if (side_us_distance >= 27.5) {
                left_right = true;
                arduino_data = 3; //shift right
                I2C_Master_Start(); // Start condition
                I2C_Master_Write(0b00010000); // 7-bit Arduino slave address + write
                I2C_Master_Write(arduino_data);
                I2C_Master_Stop();
```

```
                    __delay_ms(100);

                    arduino_data = 5; //stop
                    I2C_Master_Start(); // Start condition
                    I2C_Master_Write(0b00010000); // 7-bit Arduino slave address + write
                    I2C_Master_Write(arduino_data);
                    I2C_Master_Stop();
                    __delay_ms(200);
                }


                ball_dispensing(facing_left);
            }
        }
    } else { //open towards to sensor (right)

        //printf("facing right");
        facing_left = false;
        //__delay_ms(500);
        //check right IR
        if (right_ball) {
            //there is ball already
            canister_full = true;
        } else {
            if (distance_between >= 30) {
                canister_full = true;
                supplied_ball = true;
                //ready to supply ball
                //shift left
                if (side_us_distance <= 13) {
                    left_right = true;
                    arduino_data = 4; //shift left
                    I2C_Master_Start(); // Start condition
                    I2C_Master_Write(0b00010000); // 7-bit Arduino slave address + write
                    I2C_Master_Write(arduino_data);
                    I2C_Master_Stop();

                    __delay_ms(100);


                    arduino_data = 5; //stop
                    I2C_Master_Start(); // Start condition
                    I2C_Master_Write(0b00010000); // 7-bit Arduino slave address + write
                    I2C_Master_Write(arduino_data);
                    I2C_Master_Stop();
                    __delay_ms(200);
                }
                if (side_us_distance <= 10.5) {
                    left_right = true;
                    arduino_data = 4; //shift left
                    I2C_Master_Start(); // Start condition
                    I2C_Master_Write(0b00010000); // 7-bit Arduino slave address + write
                    I2C_Master_Write(arduino_data);
                    I2C_Master_Stop();
                    __delay_ms(100);

                    arduino_data = 5; //stop
                    I2C_Master_Start(); // Start condition
                    I2C_Master_Write(0b00010000); // 7-bit Arduino slave address + write
                    I2C_Master_Write(arduino_data);
                    I2C_Master_Stop();
                    __delay_ms(200);
                }


                ball_dispensing(facing_left);

            }
        }
    }
    canister_info[position].full = canister_full;
    canister_info[position].received_ball = supplied_ball;



    left_right = false;


}

void ball_dispensing(bool facing_left) {

    //lcd_clear();
    //printf("Stepper ON");
    //__delay_ms(100);

    if (!facing_left) {
        ball_to_left();
    } else {
        ball_to_right();
    }
}

unsigned int operation_time(rtc start_time) {
    int accumulated_seconds = 0;

    unsigned int start_second = ((start_time.second & 0b01110000) >> 4)*10 + ((start_time
        ↪ .second)&0x0F);
    unsigned int start_minute = (start_time.minute >> 4)*10 + ((start_time.minute)&0x0F);
    unsigned int start_hour = ((start_time.hour & 0b00110000) >> 4)*10 + ((start_time.
        ↪ hour)&0x0F);

    getTime();

    unsigned int current_second = ((rtc_time.second & 0b01110000) >> 4)*10 + ((rtc_time.
        ↪ second)&0x0F);
    unsigned int current_minute = (rtc_time.minute >> 4)*10 + ((rtc_time.minute)&0x0F);
    unsigned int current_hour = ((rtc_time.hour & 0b00110000) >> 4)*10 + ((rtc_time.hour)
        ↪ &0x0F);

    accumulated_seconds = (current_hour - start_hour)*3600 + (current_minute -
        ↪ start_minute)*60 + (current_second - start_second);
    return accumulated_seconds;
}

void go_back(void) {
    //send data to arduino to move back
    unsigned char arduino_data = 2; //stop
    I2C_Master_Start(); // Start condition
    I2C_Master_Write(0b00010000); // 7-bit Arduino slave address + write
    I2C_Master_Write(arduino_data); // Write key press data
    I2C_Master_Stop();

    while (1) {
        I2C_Master_Start();
        I2C_Master_Write(0b00010001); // 7-bit Arduino slave address + Read
        unsigned char flag = I2C_Master_Read(NACK); // Read one char only
        I2C_Master_Stop();
        if (flag == 253) {
            break;
        }
    }


    arduino_data = 0; //stop
    I2C_Master_Start(); // Start condition
    I2C_Master_Write(0b00010000); // 7-bit Arduino slave address + write
    I2C_Master_Write(arduino_data); // Write key press data
    I2C_Master_Stop();
    __delay_ms(1000);
    arduino_data = 6; //stop
    I2C_Master_Start(); // Start condition
    I2C_Master_Write(0b00010000); // 7-bit Arduino slave address + write
    I2C_Master_Write(arduino_data); // Write key press data
    I2C_Master_Stop();
    __delay_ms(600);


}

float median(float *list) {
    unsigned int n = 3;
    for (int i = 0; i < n; i++) //Loop for ascending ordering
    {
        for (int j = 0; j < n; j++) //Loop for comparing other values
        {
            if (list[j] > list[i]) //Comparing other array elements
            {
                float tmp = list[i]; //Using temporary variable for storing last value
                list[i] = list[j]; //replacing value
                list[j] = tmp; //storing last value
            }
        }
    }
    return list[1];

}
```

Appendix C

PIC Microcontroller I2C Program

```
/**
 * @file
 * @author Michael Ding
 * @author Tyler Gamvrelis
 *
 * Created on August 4, 2016, 3:22 PM
 *
 * @ingroup I2C
 */

/********************************* Includes ***********************************/
#include "I2C.h"

/***************************** Private Functions ******************************/
/**
 * @brief Private function used to poll the MSSP module status. This function
 * exits when the I2C module is idle.
 * @details The static keyword makes it so that files besides I2C.c cannot
 * "see" this function
 */
static inline void I2C_Master_Wait(){
    // Wait while:
    // 1. A transmit is in progress (SSPSTAT & 0x04)
    // 2. A Start/Repeated Start/Stop/Acknowledge sequence has not yet been
    // cleared by hardware
    while ((SSPSTAT & 0x04) || (SSPCON2 & 0x1F)){
        continue;
    }
}

/***************************** Public Functions *******************************/
void I2C_Master_Init(const unsigned long clockFreq){
    // Disable the MSSP module
    SSPCON1bits.SSPEN = 0;

    // Force data and clock pin data directions
    TRISCbits.TRISC3 = 1; // SCL (clock) pin
    TRISCbits.TRISC4 = 1; // SDA (data) pin
    // See section 17.4.6 in the PIC18F4620 datasheet for master mode details.
    // Below, the baud rate is configured by writing to the SSPADD<6:0>
    // according to the formula given on page 172
    SSPADD = (_XTAL_FREQ / (4 * clockFreq)) - 1;

    // See PIC18F4620 datasheet, section 17.4 for I2C configuration
    SSPSTAT = 0b10000000; // Disable slew rate control for cleaner signals

    // Clear errors & enable the serial port in master mode
    SSPCON1 = 0b00101000;

    // Set entire I2C operation to idle
    SSPCON2 = 0b00000000;
}

void I2C_Master_Start(void){
    I2C_Master_Wait(); // Ensure I2C module is idle
    SSPCON2bits.SEN = 1; // Initiate Start condition
}

void I2C_Master_RepeatedStart(void){
    I2C_Master_Wait(); // Ensure I2C module is idle
    SSPCON2bits.RSEN = 1; // Initiate Repeated Start condition
}

void I2C_Master_Stop(void){
    I2C_Master_Wait(); // Ensure I2C module is idle
    SSPCON2bits.PEN = 1; // Initiate Stop condition
}

void I2C_Master_Write(unsigned byteToWrite){
    I2C_Master_Wait(); // Ensure I2C module is idle

    // Write byte to the serial port buffer for transmission
    SSPBUF = byteToWrite;
}

unsigned char I2C_Master_Read(unsigned char ackBit){
    I2C_Master_Wait(); // Ensure I2C module is idle
    SSPCON2bits.RCEN = 1; // Enable receive mode for I2C module

    I2C_Master_Wait(); // Wait until receive buffer is full

    // Read received byte from the serial port buffer
    unsigned char receivedByte = SSPBUF;

    I2C_Master_Wait(); // Ensure I2C module is idle
    SSPCON2bits.ACKDT = ackBit; // Acknowledge data bit
    SSPCON2bits.ACKEN = 1; // Initiate acknowledge bit transmission sequence

    return receivedByte;
}
```

Appendix D

PIC Microcontroller LCD Program

```
/**
 * @file
 * @author Michael Ding
 * @author Tyler Gamvrelis
 *
 * Created on July 18, 2016, 12:11 PM
 * @ingroup CharacterLCD
 */

/********************************* Includes **********************************/
#include "lcd.h"

/******************************** Constants **********************************/
const unsigned char LCD_SIZE_HORZ = 16;
const unsigned char LCD_SIZE_VERT = 4;

const unsigned char LCD_LINE1_ADDR = 0;
const unsigned char LCD_LINE2_ADDR = 64;
const unsigned char LCD_LINE3_ADDR = 16;
const unsigned char LCD_LINE4_ADDR = 80;

/***************************** Private Functions *****************************/
/**
 * @brief Pulses the LCD register enable signal, which causes the LCD to latch
 * the data on LATD. Interrupts are disabled during this pulse to
 * guarantee that the timing requirements of the LCD's protocol are met
 */
static inline void pulse_e(void){
   unsigned char interruptState = INTCONbits.GIE;
   di();
   E = 1;
   // This first delay only needs to be 1 microsecond in theory, but 25 was
   // selected experimentally to be safe
   __delay_us(25);
   E = 0;
   __delay_us(100);
   INTCONbits.GIE = interruptState;
}
```

```
/**
 * @brief Low-level function to send 4 bits to the display
 * @param data The byte whose 4 least-significant bits are to be sent to the LCD
 */
static void send_nibble(unsigned char data){
    // Send the 4 least-significant bits
    LATD = (unsigned char)(LATD & 0x0F); // Clear LATD[7:4]
    LATD = (unsigned char)((data << 4) | LATD); // Write data[3:0] to LATD[7:4]
    pulse_e();
}

/**
 * @brief Low-level function to send a byte to the display
 * @param data The byte to be sent
 */
static void send_byte(unsigned char data){
    // Send the 4 most-significant bits
    send_nibble(data >> 4);

    // Send the 4 least-significant bits
    send_nibble(data);
}

/***************************** Public Functions *****************************/
void lcdInst(char data){
    RS = 0;
    send_byte(data);
}

void initLCD(void){
    __delay_ms(15);

    RS = 0;
    // Set interface length to 4 bits wide
    send_nibble(0b0011);
    __delay_ms(5);
    send_nibble(0b0011);
    __delay_us(150);
    send_byte(0b00110010);

    send_byte(0b00101000); // Set N = number of lines (1 or 2) and F = font
    send_byte(0b00001000); // Display off
    send_byte(0b00000001); // Display clear
    __delay_ms(5);
    send_byte(0b00000110); // Entry mode set

    // Enforce on: display, cursor, and cursor blinking
    lcd_display_control(true, true, true);
}

void lcd_shift_cursor(unsigned char numChars, lcd_direction_e direction){
    for(unsigned char n = numChars; n > 0; n--){
        lcdInst((unsigned char)(0x10 | (direction << 2)));
    }
}

void lcd_shift_display(unsigned char numChars, lcd_direction_e direction){
    for(unsigned char n = numChars; n > 0; n--){
        lcdInst((unsigned char)(0x18 | (direction << 2)));
    }
}

void putch(char data){
    RS = 1;
    send_byte((unsigned char)data);
}
```

Appendix E

PIC Microcontroller Configuration Bits

```
#ifndef CONFIG_BITS_H
#define CONFIG_BITS_H

// CONFIG1H
#pragma config OSC = HS // Oscillator Selection bits (HS oscillator)
#pragma config FCMEN = OFF // Fail-Safe Clock Monitor Enable bit (Fail-Safe Clock Monitor
    ↪  disabled)
#pragma config IESO = OFF // Internal/External Oscillator Switchover bit (Oscillator
    ↪ Switchover mode disabled)

// CONFIG2L
#pragma config PWRT = OFF // Power-up Timer Enable bit (PWRT disabled)
#pragma config BOREN = OFF // Brown-out Reset Enable bits (Brown-out Reset enabled in
    ↪ hardware only (SBOREN is disabled))
#pragma config BORV = 3 // Brown Out Reset Voltage bits (Minimum setting)

// CONFIG2H

#pragma config WDT = OFF // Watchdog Timer Enable bit (WDT disabled (control is placed on
    ↪  the SWDTEN bit))
#pragma config WDTPS = 32768 // Watchdog Timer Postscale Select bits (1:32768)

// CONFIG3H
#pragma config CCP2MX = PORTC // CCP2 MUX bit (CCP2 input/output is multiplexed with RC1)
#pragma config PBADEN = ON // PORTB A/D Enable bit (PORTB<4:0> pins are configured as
    ↪ analog input channels on Reset)
#pragma config LPT1OSC = OFF // Low-Power Timer1 Oscillator Enable bit (Timer1 configured
    ↪  for higher power operation)
#pragma config MCLRE = ON // MCLR Pin Enable bit (MCLR pin enabled; RE3 input pin
    ↪ disabled)

// CONFIG4L
#pragma config STVREN = ON // Stack Full/Underflow Reset Enable bit (Stack full/underflow
    ↪  will cause Reset)
#pragma config LVP = OFF // Single-Supply ICSP Enable bit (Single-Supply ICSP disabled)
#pragma config XINST = OFF // Extended Instruction Set Enable bit (Instruction set
    ↪ extension and Indexed Addressing mode disabled (Legacy mode))
```

```
// CONFIG5L
#pragma config CP0 = OFF // Code Protection bit (Block 0 (000800-003FFFh) not code-
    ↪ protected)
#pragma config CP1 = OFF // Code Protection bit (Block 1 (004000-007FFFh) not code-
    ↪ protected)
#pragma config CP2 = OFF // Code Protection bit (Block 2 (008000-00BFFFh) not code-
    ↪ protected)
#pragma config CP3 = OFF // Code Protection bit (Block 3 (00C000-00FFFFh) not code-
    ↪ protected)

// CONFIG5H
#pragma config CPB = OFF // Boot Block Code Protection bit (Boot block (000000-0007FFh)
    ↪ not code-protected)
#pragma config CPD = OFF // Data EEPROM Code Protection bit (Data EEPROM not code-
    ↪ protected)

// CONFIG6L
#pragma config WRT0 = OFF // Write Protection bit (Block 0 (000800-003FFFh) not write-
    ↪ protected)
#pragma config WRT1 = OFF // Write Protection bit (Block 1 (004000-007FFFh) not write-
    ↪ protected)
#pragma config WRT2 = OFF // Write Protection bit (Block 2 (008000-00BFFFh) not write-
    ↪ protected)
#pragma config WRT3 = OFF // Write Protection bit (Block 3 (00C000-00FFFFh) not write-
    ↪ protected)

// CONFIG6H
#pragma config WRTC = OFF // Configuration Register Write Protection bit (Configuration
    ↪ registers (300000-3000FFh) not write-protected)
#pragma config WRTB = OFF // Boot Block Write Protection bit (Boot Block (000000-0007FFh)
    ↪  not write-protected)
#pragma config WRTD = OFF // Data EEPROM Write Protection bit (Data EEPROM not write-
    ↪ protected)

// CONFIG7L
#pragma config EBTR0 = OFF // Table Read Protection bit (Block 0 (000800-003FFFh) not
    ↪ protected from table reads executed in other blocks)
#pragma config EBTR1 = OFF // Table Read Protection bit (Block 1 (004000-007FFFh) not
    ↪ protected from table reads executed in other blocks)
#pragma config EBTR2 = OFF // Table Read Protection bit (Block 2 (008000-00BFFFh) not
    ↪ protected from table reads executed in other blocks)
#pragma config EBTR3 = OFF // Table Read Protection bit (Block 3 (00C000-00FFFFh) not
    ↪ protected from table reads executed in other blocks)

// CONFIG7H
#pragma config EBTRB = OFF // Boot Block Table Read Protection bit (Boot Block
    ↪ (000000-0007FFh) not protected from table reads executed in other blocks)

// #pragma config statements should precede project file includes.
// Use project enums instead of #define for ON and OFF.

#include <xc.h>
```

```
#define _XTAL_FREQ 10000000 // Define osc freq for use in delay macros


#endif /* CONFIG_BITS_H */
```

Appendix F

Arduino Microcontroller Main Program

```
#include <PID_v1.h>
#include <Wire.h>
#include <PinChangeInt.h>
#include <IRremote.h>

IRrecv irrecv(7);
decode_results results;
//unsigned long key_value = 0;


bool left = false;
bool right = false;
bool start = true;
bool back = false;
bool adjust_left = false;
bool adjust_right = false;
bool remote_start = false;
bool remote_stop = false;
bool pid_back = false;
unsigned long recorded_encoder = 0xFFFF;
bool back_to_line = false;
bool remote_stepper_left = false;
bool remote_stepper_right = false;



const int encoder_back_right = A3; //back right 160
const int encoder_back_left = 3; //back left 127
const int encoder_front_left = A2;
const int encoder_front_right = 2;

const int front_left = 0;
const int front_right = 3;
const int back_left = 2;
const int back_right = 1;

const int pwm[4] = {9, 6, 10, 5};//11
const int motor_positive[4] = {12, 4, 8, A0};
const int motor_negative[4] = {13, 13, 11, A1};
```

```
volatile unsigned int encoder_count_back_right = 0;
volatile unsigned int encoder_count_back_left = 0;
volatile unsigned int encoder_count_front_right = 0;
volatile unsigned int encoder_count_front_left = 0;

volatile unsigned long encoder_br_shift = 0;
volatile unsigned long encoder_bl_shift = 0;
volatile unsigned long encoder_fr_shift = 0;
volatile unsigned long encoder_fl_shift = 0;

//////////////////////////////////////////////////////////
//PID setup

double setpoint = 0;
//double setpoint_fl = 0;

double input_front_right = 0;
double input_back_right = 0;
double input_front_left = 0;
double input_back_left = 0;

double output_front_right = 61; //60
double output_back_right = 61; //61
double output_front_left = 34; //40
double output_back_left = 42; //42



double kp = 0.06;//0.05
double ki = 0;
double kd = 0.2;//0.2


PID PID_front_right(&input_front_right, &output_front_right, &setpoint, kp, ki, kd,
    ↪ DIRECT);
PID PID_back_right(&input_back_right, &output_back_right, &setpoint, kp, ki, kd, DIRECT);
PID PID_front_left(&input_front_left, &output_front_left, &setpoint, kp, ki, kd, DIRECT);
PID PID_back_left(&input_back_left, &output_back_left, &setpoint, kp, ki, kd, DIRECT);


volatile unsigned char distance_data = 0;
volatile unsigned int non_erased_count = 0;
volatile bool enable = false;


void setup() {
  irrecv.enableIRIn();

  Wire.begin(8); // Join I2C bus with address 8

  // Register callback functions
  Wire.onReceive(receiveEvent); // Called when this slave device receives a data
      ↪ transmission from master
```

```
Wire.onRequest(requestEvent); // Called when master requests data from this slave
    ↪ device


for (int i = 0; i < 4; i++) {
  pinMode(pwm[i], OUTPUT);
  pinMode(motor_positive[i], OUTPUT);
  pinMode(motor_negative[i], OUTPUT);
}


pinMode(encoder_back_right, INPUT);
pinMode(encoder_back_left, INPUT);
pinMode(encoder_front_right, INPUT);
pinMode(encoder_front_left, INPUT);

digitalWrite(encoder_back_right, HIGH); // turn on pull-up resistor
digitalWrite(encoder_back_left, HIGH); // turn on pull-up resistor
digitalWrite(encoder_front_right, HIGH); // turn on pull-up resistor
digitalWrite(encoder_front_left, HIGH); // turn on pull-up resistor

digitalWrite(motor_positive[back_left], LOW);
digitalWrite(motor_negative[back_left], HIGH);

digitalWrite(motor_positive[front_right], LOW);
digitalWrite(motor_negative[front_right], HIGH);

digitalWrite(motor_positive[front_left], HIGH);
digitalWrite(motor_negative[front_left], LOW);

digitalWrite(motor_positive[back_right], HIGH);
digitalWrite(motor_negative[back_right], LOW);


for (int i = 0; i < 4; i++) {

  digitalWrite(motor_positive[i], HIGH);
  digitalWrite(motor_negative[i], LOW);


}


// D2 interrupt
attachInterrupt(0, do_encoder_front_right, RISING);
// D3 interrupt
attachInterrupt(1, do_encoder_back_left, RISING);

PCintPort::attachInterrupt(encoder_front_left, do_encoder_front_left, RISING);
PCintPort::attachInterrupt(encoder_back_right, do_encoder_back_right, RISING);

PCintPort::attachInterrupt(7, IR, CHANGE);

//Serial.begin(9600);
```

```
  PID_front_right.SetMode(AUTOMATIC);
  PID_back_right.SetMode(AUTOMATIC);
  PID_front_left.SetMode(AUTOMATIC);

  PID_back_left.SetMode(AUTOMATIC);

  //////////////////////////////
  //remote setup




  //FL: 76.18 FR: 92.21 BL: 85.96 BR: 76.28
  //
  //Index: BR:2842 BL:2720 FR:2763 FL:2842
  //
  //
  //FL: 79.18 FR: 89.28 BL: 84.90 BR: 76.16
  //
  //Index: BR:2894 BL:2768 FR:2815 FL:2892
  //
  //
  //FL: 79.18 FR: 90.24 BL: 88.02 BR: 76.04
  //
  //Index: BR:2948 BL:2815 FR:2868 FL:2939
  //
  //
  //FL: 82.36 FR: 91.27 BL: 90.26 BR: 72.74
  //
  //Index: BR:2998 BL:2859 FR:2915 FL:2985
  //
  //
  //FL: 84.66 FR: 94.38 BL: 92.62 BR: 80.92




}

void loop() {


  if (left) {
    digitalWrite(motor_positive[back_right], LOW);
    digitalWrite(motor_negative[back_right], HIGH);

    digitalWrite(motor_positive[front_left], LOW);
    digitalWrite(motor_negative[front_left], HIGH);

    analogWrite(pwm[back_left], 195);
    analogWrite(pwm[front_right], 200);
    analogWrite(pwm[front_left], 205);
    analogWrite(pwm[back_right], 213);
  }
```

```
else if (right) {
  digitalWrite(motor_positive[front_right], LOW);
  digitalWrite(motor_negative[front_right], HIGH);

  digitalWrite(motor_positive[back_left], LOW);
  digitalWrite(motor_negative[back_left], HIGH);

  analogWrite(pwm[back_left], 225);
  analogWrite(pwm[front_right], 200);
  analogWrite(pwm[front_left], 185);
  analogWrite(pwm[back_right], 185);
}

else if (back) {
  for (int i = 0; i < 4; i++) {
    if ((i == 0) || (i == 1)) {
      digitalWrite(motor_positive[i], HIGH);
      digitalWrite(motor_negative[i], LOW);
    }
    else {
      digitalWrite(motor_positive[i], LOW);
      digitalWrite(motor_negative[i], HIGH);
    }
  }
  analogWrite(pwm[back_left], 255);
  analogWrite(pwm[front_right], 220);
  analogWrite(pwm[front_left], 220);
  analogWrite(pwm[back_right], 220);
  delay(1110);

  back = false;

  for (int i = 0; i < 4; i++) {
      digitalWrite(motor_positive[i], LOW);
      digitalWrite(motor_negative[i], HIGH);
  }
  recorded_encoder = non_erased_count;

  pid_back = true;



}

else if (enable) {
  setpoint += 0.08;//0.08


  input_front_right = encoder_count_front_right * 1.06; //slow down 1.04
  input_front_left = encoder_count_front_left*0.98;
  input_back_right = encoder_count_back_right * 0.97;
  input_back_left = encoder_count_back_left / 2 * 3 * 1.065; //slow down
  PID_front_right.Compute();
  PID_back_right.Compute();
  PID_front_left.Compute();
  PID_back_left.Compute();


  analogWrite(pwm[back_left], output_back_left);
  analogWrite(pwm[front_right], output_front_right);
  analogWrite(pwm[front_left], output_front_left);
  analogWrite(pwm[back_right], output_back_right);

  //delay(1);

}
else if (pid_back){
  setpoint += 0.9;//0.08
  input_front_right = encoder_count_front_right * 1.06; //slow down 1.04
  input_front_left = encoder_count_front_left;
  input_back_right = encoder_count_back_right * 0.97;
  input_back_left = encoder_count_back_left / 2 * 3 * 1.065; //slow down


  PID_front_right.Compute();
  PID_back_right.Compute();
  PID_front_left.Compute();
  PID_back_left.Compute();


  analogWrite(pwm[back_left], output_back_left);
  analogWrite(pwm[front_right], output_front_right);
  analogWrite(pwm[front_left], output_front_left);
  analogWrite(pwm[back_right], output_back_right);

}

else {
  analogWrite(pwm[back_left], 0);
  analogWrite(pwm[front_right], 0);
  analogWrite(pwm[front_left], 0);
  analogWrite(pwm[back_right], 0);

}







// Serial.print("FL: ");
// Serial.print(output_front_left);
// Serial.print(" FR: ");
// Serial.print(output_front_right);
// Serial.print(" BL: ");
```

```
  // Serial.print(output_back_left);
  //
  // Serial.print(" BR: ");
  // Serial.println(output_back_right);
  // Serial.println();
  //
  // Serial.print("Index: BR:");
  // Serial.print(encoder_count_back_right);
  //
  // Serial.print(" BL:");
  // Serial.print(encoder_count_back_left);
  //
  //
  // Serial.print(" FR:");
  // Serial.print(encoder_count_front_right);
  //
  // Serial.print(" FL:");
  // Serial.print(encoder_count_front_left);
  // Serial.println();
  // Serial.println();
  // Serial.println();

}


uint8_t receiveEvent(void) {
  uint8_t x = Wire.read(); // Receive byte
  //Serial.println(x, BIN);
  if (x == 0) {
    pid_back = false;
    left = false;
    right = false;
    back = false;
    enable = false;
    setpoint -= 1;
        analogWrite(pwm[back_left], 0);
    analogWrite(pwm[front_right], 0);
    analogWrite(pwm[front_left], 0);
    analogWrite(pwm[back_right], 0);
  }
  if (x == 1) {//start motor
    for (int i = 0; i < 4; i++) {
      digitalWrite(motor_positive[i], HIGH);
      digitalWrite(motor_negative[i], LOW);

    }
    output_front_right = 63; //118
    output_back_right = 63; //133
    output_front_left = 34; //75
    output_back_left = 42; //110
    enable = true;


  }
```

```
else if (x == 2) {
  enable = false;
  back = true;
  //front left, back right






}
else if (x == 4) {
  //adjust to left
  left = true;
  back = false;
  enable = false;

}
else if (x == 3) {
  //adjust to right
  //adjust_right = true;
  right = true;
  back = false;
  enable = false;


}

else if (x == 5) {
  analogWrite(pwm[back_left], 0);
  analogWrite(pwm[front_right], 0);
  analogWrite(pwm[front_left], 0);
  analogWrite(pwm[back_right], 0);

  left = false;
  right = false;
  back = false;
  enable = false;
  setpoint -= 1;

}

else if (x==6){
  back_to_line = true;
}



else {
  //stop motor
  left = false;
  right = false;
  back = false;
```

```
    enable = false;
    setpoint -= 1;

  }




  //Serial.println((char)x); // Print to serial output as char (ASCII representation)
  return x;
}
/** @brief Callback for when the master requests data */
void requestEvent(void) {
  if (non_erased_count>=(recorded_encoder+9000)){
    Wire.write(253);
  }
  else if (remote_start) {
    //remote_start = false;
    Wire.write(254);
  }
  else if (remote_stepper_left){
    remote_stepper_left = false;
    Wire.write(3);
  }
  else if (remote_stepper_right){
    remote_stepper_right = false;
    Wire.write(2);
  }
  else {
    Wire.write(1);
  }



}
unsigned char encoder_to_distance(void) {
  return non_erased_count / 20.907 * 0.5;
}


void do_encoder_back_right() {
  if ((!left) && (!right) && (!back)) {
    encoder_count_back_right++;
  }


}


void do_encoder_back_left() {
  if ((!left) && (!right) && (!back)) {
    encoder_count_back_left++;
  }
}
```

```
void do_encoder_front_right() {
  if ((!left) && (!right) && (!back)) {
    non_erased_count++;
    encoder_count_front_right++;
  }
}


void do_encoder_front_left() {
  if ((!left) && (!right) && (!back)) {
    encoder_count_front_left++;
  }
}

void IR() {
  if (irrecv.decode(&results)) {

    if (results.value == 0xFFC23D) {

      if (!remote_start) {
        remote_start = true;
      }
      else {
        analogWrite(pwm[back_left], 0);
        analogWrite(pwm[front_right], 0);
        analogWrite(pwm[front_left], 0);
        analogWrite(pwm[back_right], 0);
        remote_stop = true;
        while (1);
      }

    }
    else if (results.value == 0xFF22DD){
      remote_stepper_left = true;
    }
    else if (results.value == 0xFF02FD){
      remote_stepper_right = true;
    }
    irrecv.resume();
  }
}
```

Appendix G

PC Interface Main Code

```
from conversion import *
from structure import *

eeprom_file = open(’EEPROM.txt’,’r’)
content = eeprom_file.read()
eeprom_file.close()

operation_log = []
information = []

i = 0

while 1:
    if i>=len(content):
        break
    [success,converted] = toHex(content[i])
    if success:
        [second,lower_bit] = toHex(content[i+1])
        operation_log = operation_log + [(converted << 4) | lower_bit]
        i = i + 2
    else:
        i = i + 1

for i in range (0,5):
    each_trial = operation(operation_log[i*26+1],operation_log[i*26+2],operation_log[i
        ↪ *26+3],operation_log[i*26+4],operation_log[i*26+5],operation_log[i*26+6])
    j = 0
    for j in range(0,20,2):
        each_trial.add_canister(operation_log[i*26+7+j],operation_log[i*26+8+j])
    information = information + [each_trial]


output = open(’Operation_Log.txt’,’w’)

for i in range (0,5):
    timing = hex(information[i].month)[2:].zfill(2) + ’/’ + hex(information[i].day)[2:].
        ↪ zfill(2) + ’\t\t’ + hex(information[i].hour)[2:].zfill(2) + ’:’ + hex(
        ↪ information[i].minute)[2:].zfill(2) + ’\n’
    operation_time = str(information[i].duration) + 's␣OPERATED\n'

    balls_supplied = str(information[i].balls_supplied) + '␣BALLS␣SUPPLIED\n'

    balls_left = str(10-information[i].balls_supplied) + '␣BALLS␣LEFT\n'

    canister = []

    for j in range(0,information[i].canister_count):
        if information[i].list_of_canisters[j].exist:
            canister = canister + ['CANISTER␣'+str(j+1)+'\n']
            if (information[i].list_of_canisters[j].full):
                canister = canister + ['\tFull␣Canister\n']
            else:
                canister = canister + ['\tEmpty␣Canister\n']
            if (information[i].list_of_canisters[j].received_ball):
                canister = canister + ['\tBall␣Received\n']
            else:
                canister = canister + ['\tNot␣Supplied\n']
            canister = canister + ['Distance:␣'+str(information[i].list_of_canisters[j].
                ↪ distance_from_start())+'␣cm\n\n']

    output.write(timing)
    output.write(operation_time)
    output.write(balls_supplied)
    output.write(balls_left)
    for j in range(0,len(canister)):
        output.write(canister[j])
    output.write('\n\n\n')
```

Appendix H

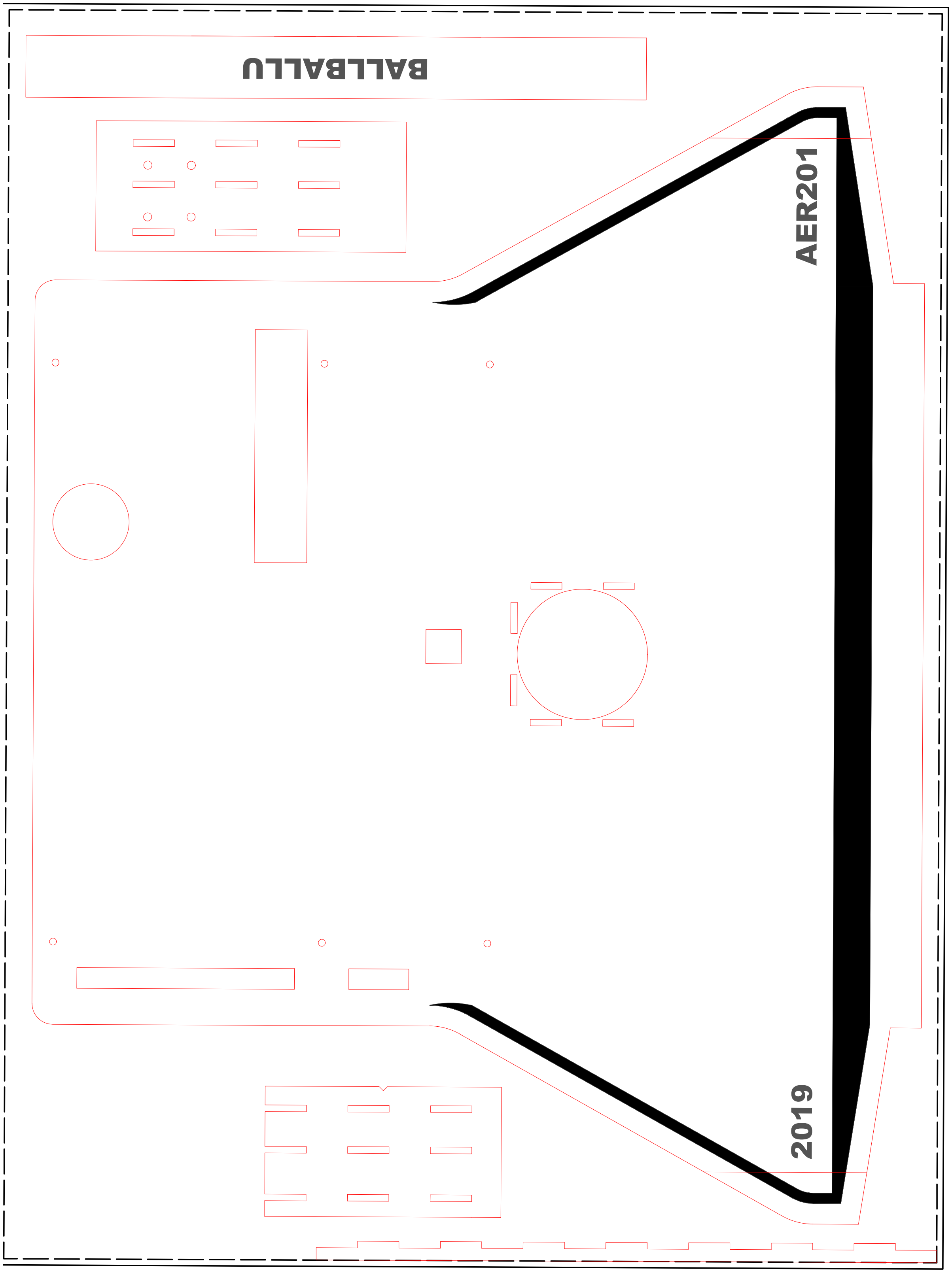
PC Interface Data Structure

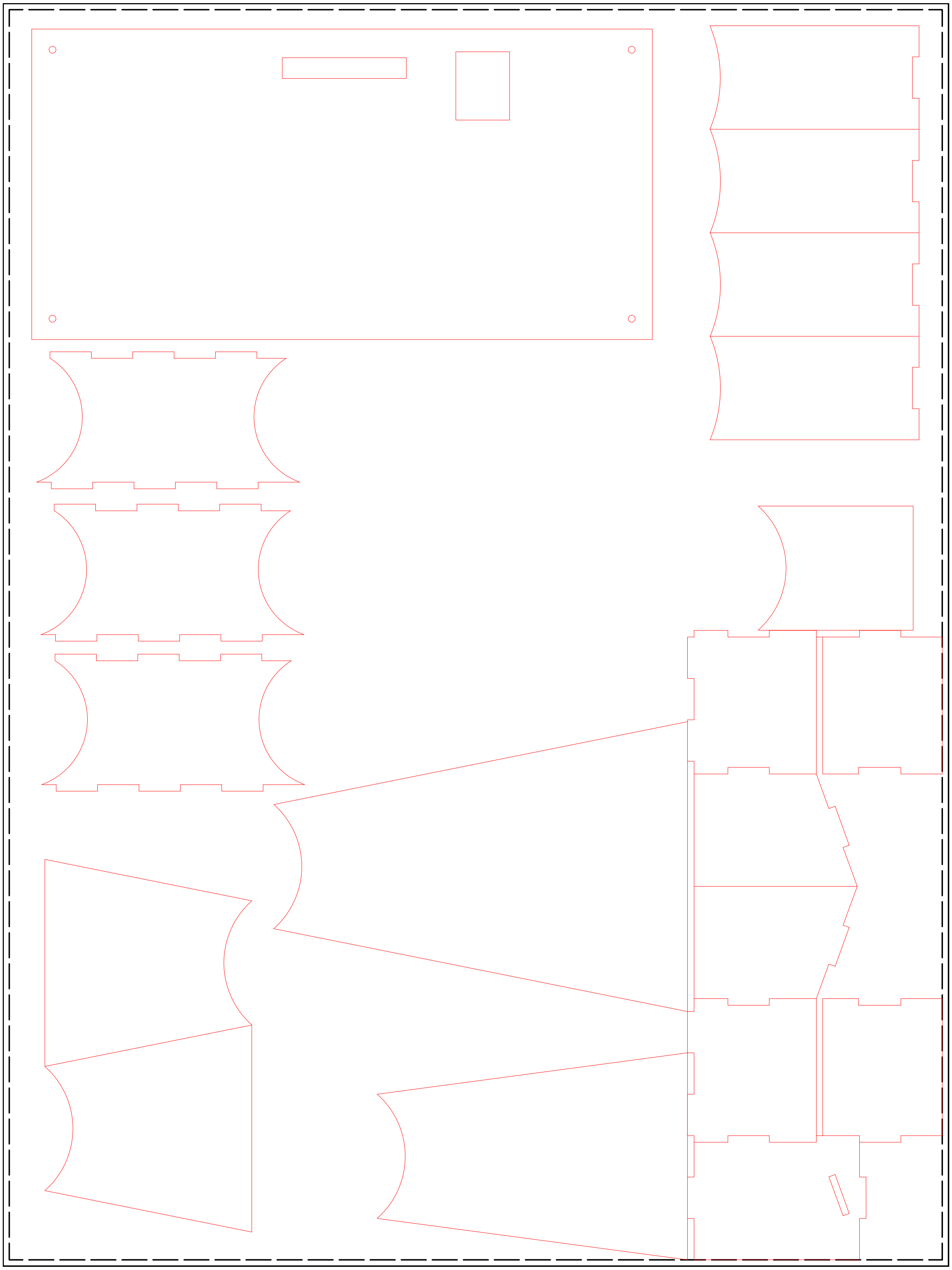
```
class operation:
    def __init__(self,month,day,hour,minute,duration,balls_supplied):
        self.month = month
        self.day = day
        self.hour = hour
        self.minute = minute
        self.duration = duration
        self.balls_supplied = balls_supplied
        self.list_of_canisters = []
        self.canister_count = 0

    def month(self):
        return self.month

    def day(self):
        return self.day

    def hour(self):
        return self.hour

    def minute(self):
        return self.minute

    def duration(self):
        return self.duration

    def balls_supplied(self):
        return self.balls_supplied

    def add_canister(self,basic,distance_LB):
        self.canister_count = self.canister_count + 1
        self.list_of_canisters = self.list_of_canisters + [canister_info(basic,distance_LB
            ↪ )]
        return True

    def canister_count(self):
        return self.canister_count
```

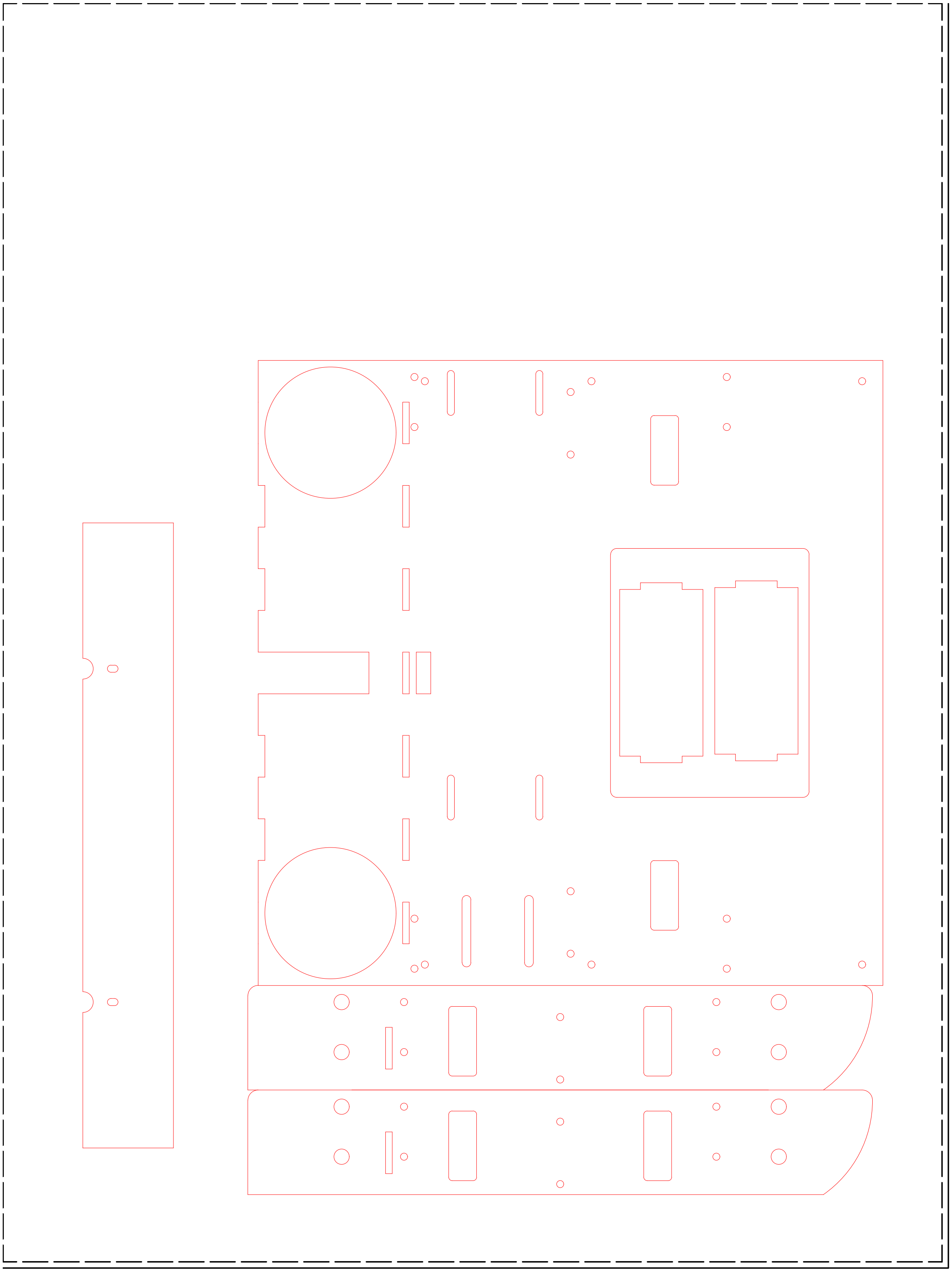
```
class canister_info:
    #this provides information of individual canister
    #bit 0: exist
    #bit 1: full, empty
    #bit 2: ball received or not
    #bit 3: distance MSB
    #second line: distance lower bits
    def __init__(self,basic,distance_LB):
        self.exist = basic & (0x01);
        self.full = (basic>>1) & (0x01)
        self.received_ball = (basic>>2)&(0x01);
        self.distance = ((basic & 0b00001000)<<5)|distance_LB;

    def distance_from_start(self):
        return self.distance

    def exist(self):
        return self.exist

    def full(self):
        return self.full

    def received_ball(self):
        return self.received_ball
```

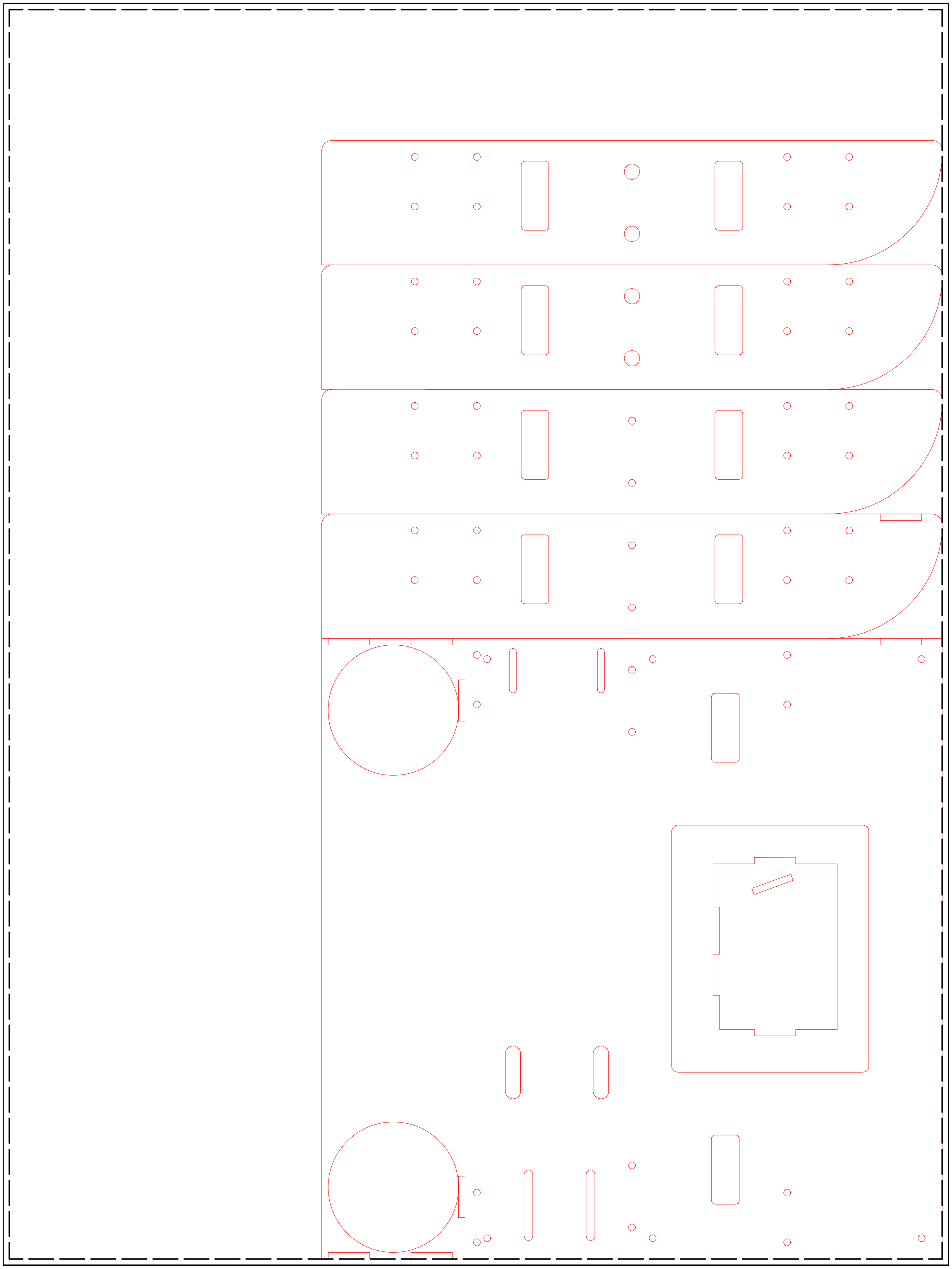
Appendix I

Laser Cut Files


BALLBALLU
AER201
2019

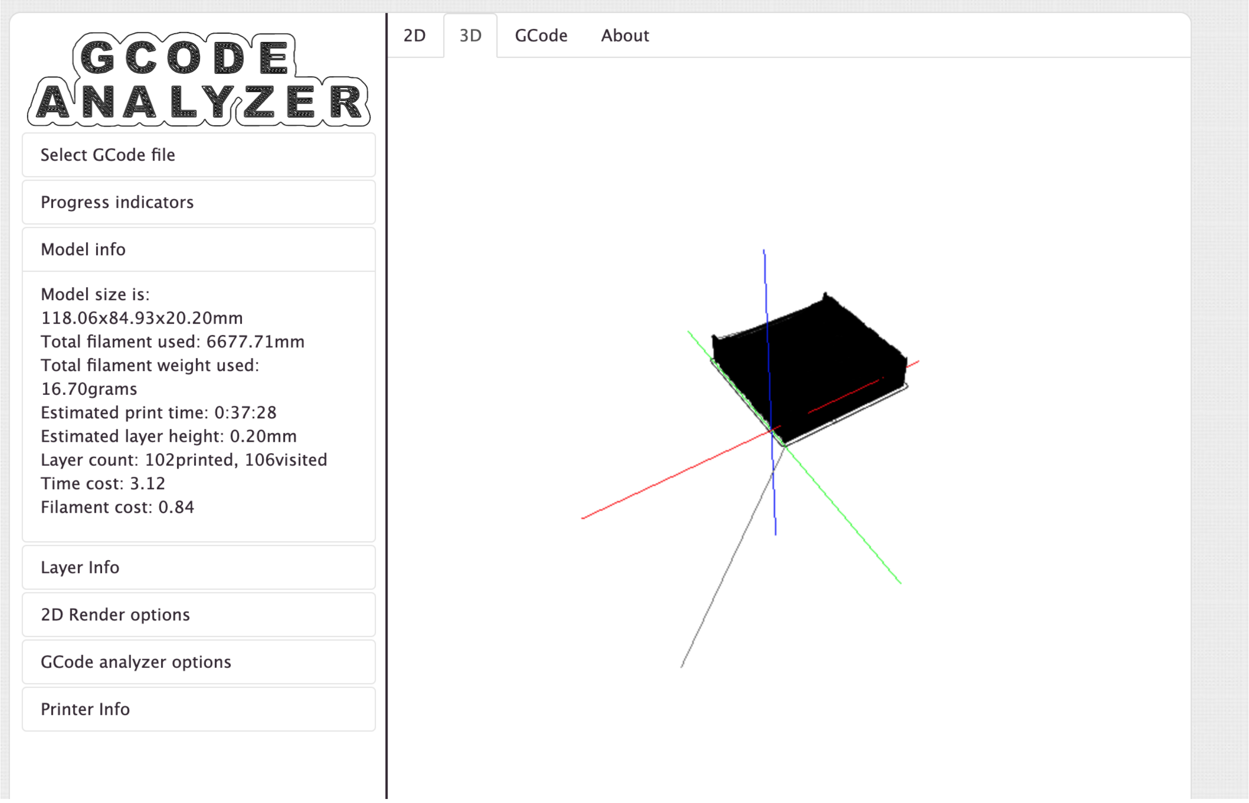






Appendix J

3D Print File



Model size is:
118.06x84.93x20.20mm
Total filament used: 6677.71mm
Total filament weight used:
16.70grams
Estimated print time: 0:37:28
Estimated layer height: 0.20mm
Layer count: 102printed, 106visited
Time cost: 3.12
Filament cost: 0.84

Figure J.1: G-code Analyze

The complete g-code could be accessed here: https://drive.google.com/open?id=1wZj97RG71Zv5yH71$_1$4$fS4ONFAxXB$