UNIVERSITY OF TORONTO

INSTITUTE FOR AEROSPACE STUDIES

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

The Traffic Cone Deployment Machine

prepared by

Team 02 – Monday

Ryan Chen (1003912992)
Hongyu Chen (1004560441)
Yutong Zhu (1004020631)

prepared for

Prof. M.R. Emami

A technical report submitted for
AER201 – Engineering Design

TA: Mollie Bianchi

April 11, 2019

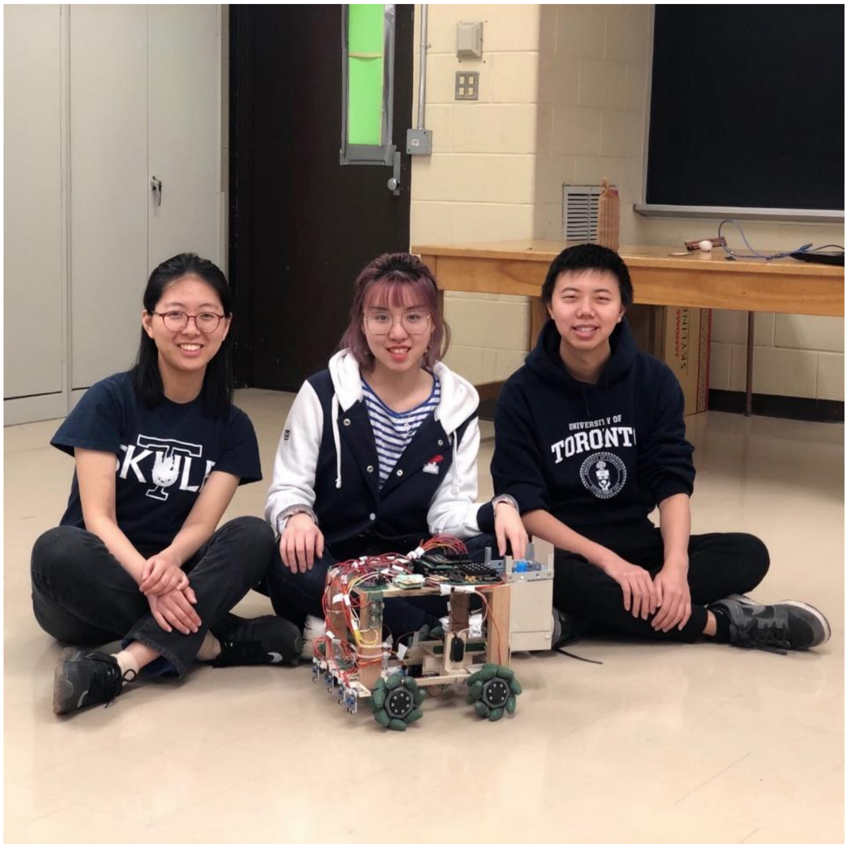
Division of Engineering Science, University of Toronto

AER201 – Engineering Design, Final Report

Prof. M. R. Emami

TA: Mollie Bianchi

The Traffic Cone Deployment Machine
Mr. Krabs



Team 2

Hongyu Chen, Yutong Zhu, Ryan Chen

Acknowledgement

This has been a challenging but precious experience. We, as a team, have taken efforts in this project. However, it would not have been such a success without the support and help of many people around us. We would like to extend our sincere thanks to the following:

We are highly indebted to Professor M.R. Emami for providing us the precious opportunity to have such valuable experience of making our first engineering robot, for his grateful guidance, extraordinary knowledge and generosity for providing us with accessories.

We would also like to express our thanks to our lovely teaching assistant, Mollie Bianchi, who has guided us through this entire journey with her enormous help and encouragement, as well as some necessary information regarding the project.

We would also like to thank our machine lab supervisor Colin Harry, who has constantly provided us guidance and supervision as well as the assistance every time we faced electromechanical challenges.

Our thanks and appreciations also go to our colleagues and peers in developing the project and people who have willingly helped us out with their abilities.

Abstract

This document is the project final report, which describes the design and construction of the robot Mr. Krabs, an autonomous traffic cone dispensing machine for the competition of AER201 Engineering Design Course 2019.

The robot Mr. Krabs has a dimension of 44cm * 34cm *29 cm, and an overall weight of 4.5kg. It is capable of moving along a straight or curved line and deploying mini traffic cones upon detection of obstacles on the road. During the operation, the robot is first positioned behind a Start Line. It begins operation when the start button on the keypad is pressed by the user. Once the operation is completed and the robot returns, the user can retrieve operation data using the keypad and a LCD. The total cost of the robot is $227.48 CAD. This robot features its omni directional driving system with 4 Mecanum wheels, which allow it to move in any directions.

Our robot has achieved great performance during the 7-day final testing. It presents history data of past four operations accurately and move along the designated lane via the two lone following sensors. However, during the competition, the middle detection sensor malfunctioned so holes were not detected. The risks and concerns can be addressed and lead to future improvement such as installing another row of sensors for risk management, etc.

Table of Contents

Symbols, Abbreviations and Definitions

AER201: University of Toronto Engineering Science Course – 2019 Winter: Engineering Design

PIC: Peripheral Interface Controller

IR Sensor: Infrared Sensor

PWM: Pulse Width Modulation

The Board: Customized Development Board Produced by AER201

LCD: Liquid Crystal Display

I/O: input and output

Transition Variables: Computer variables that will be computed and then used as inputs in the robot main operating routine

Record Variables: Computer variables that will be sent to permanent log

CAD: Computer-Aided Design

LDS: Left Detection Sensor

RDS: Right Detection Sensor

MDS: Middle Detection Sensor

1. Introduction

A traffic cone has a triangular face that are usually placed on the highways, road or footpath to temporarily redirect traffic. They are often used in creating separation or merging lane during the construction or accidents isolation.

A city engineering contractor needs to develop a machine that can automatically deploy these traffic cones given the following instructions such that:

The designed machine should be initially positioned behind the starting line in standby mode. The machine begins operation when the start button on the keypad is pressed. Once the operation is initiated, the machine will travel along the lane until the detection of a hole or crack, in which it follows the requirements such that:

1. If a crack is detected, the machine will deploy two cones on top of the crack, such that they do not contact each other.
2. If a hole is detected, the machine will deploy one cone on top of the hole.

Mr. Krabs is a mini traffic cone dispensing robot designed and constructed by Team 2 in 2019. This document is the project final report, which describes the design and construction of the robot Mr. Krabs, an autonomous traffic cone dispensing machine for the competition of AER201 Engineering Design Course 2019. This report outlines the construction details and integration progress, in response to the prototype described in the project proposal.

2. Perspective

2.1 Theory and History

Cone

Traffic cones were invented by Charles D. Scanlon from Los Angeles. The patent for his invention was granted in the year 1943 [1]. The orange traffic cones were first used in the UK and quickly spread across the pond in 1958. The first use of traffic cones in England coincide with the opening of the M6motorway, since then they have become a traffic staple [2].

2.2 Background Research

Automatic traffic cone dispensing machines are commonly used in traffic redirection and warning of roadwork or hazards. These machines allow cones to be safely set up and collected in the middle of the highway or other high capacity roads. These existing design solutions have provided inspirations for specific ideas as well as the foundation to this project that can be further explored.

2.2.1 Literature Survey

2.2.1.1 International Journal of Heavy Vehicle Systems

The International Journal of Heavy Vehicle Systems, Volume 11, Issue 2 presents the automated machine for highway cone placement and retrieval. The journal emphasizes the importance of traffic cones as a barrier that separates designed work zones and lanes from fast moving cars. This paper also reports on an automated machine for placing and collecting cones, as well as its impact on work efficiency and safety. The integration of the design is described in detail in terms of its architecture, supporting vehicles and operations [3].

2.2.1.2 University Tun Hussein Onn Malaysia Master's Project

This engineering design paper reports the development of an automatic traffic cone dispenser and collector system utilizing a computer-controlled robotic arm as shown in Figure 1. The system consists of five degrees of freedom in order to achieve maximum flexibility and efficiency [4].

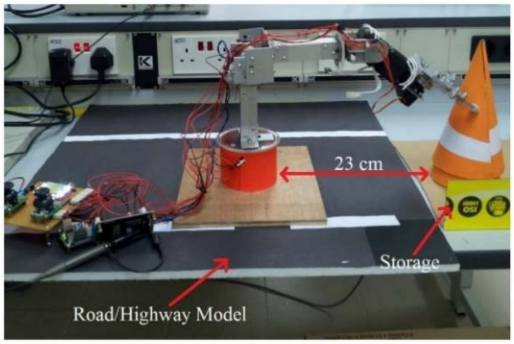


Figure 1. The Robotic Arm Picking Up a Traffic Cone

2.2.2 Idea Survey

2.2.2.1 AER201 2011 cone dispensing machine

The cone dispensing machine designed by a 2011 AER201 project group presents a lot of similarities compared to our project, which can be considered as a valuable source of idea generation [5]. As shown in Figure 2, this team used aluminum for its core structure, with hollow spaces in between different sheets to reduce the overall weight of the robot. This idea complies with our design objective that the robot should be lightweight, which induces less stress onto the driving motors. In addition, the structural design of the cone dispensing mechanism at the back of the cart also aligns with our initial concept of the mechanism.

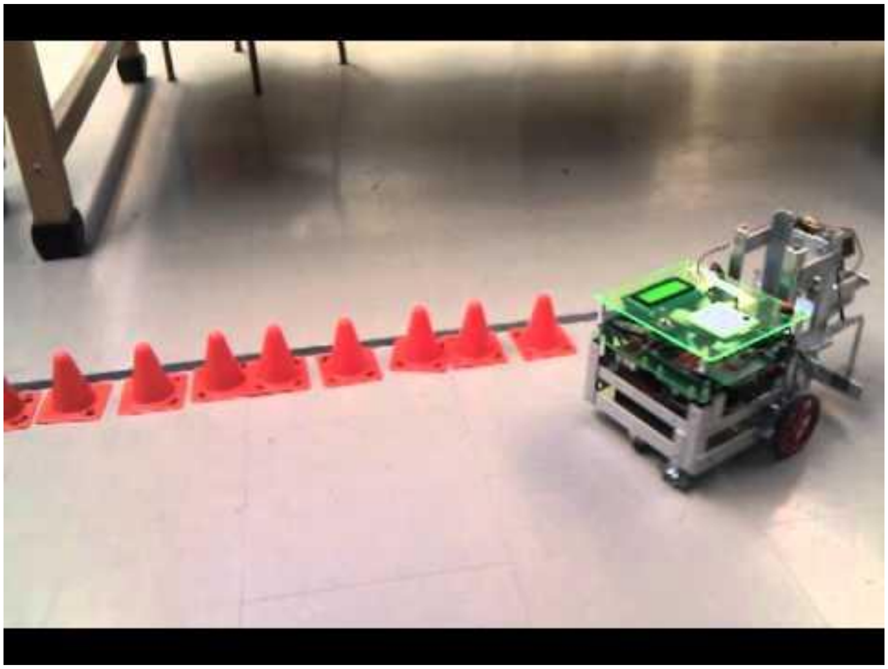


Figure 2. The Overall Structure of the Robot Designed by a 2011 AER201 Team

2.3.3 Market Survey

2.3.3.1 U.S. Patent: Highway Cone Dispenser and Collector

This patented device consists of a rotatable cone conveyor that moves the cone up and down, as well as a stripper for removing the cones from the conveyor at either upper or lower location. This collector-dispenser system can be mounted on any standard pickup truck without specific modification for both rotational and translational movement. Furthermore, the cone conveyor includes flexible fingers that can firmly grip the cones [6]. Diagrams of the detailed design of the device are included in Appendix C - 1.

2.3.3.2 U.S. Patent: Device for the Placement and If Desired the Collection of Traffic Cone

This patent describes another vehicle mounted traffic cone placement and collection machine. The cones are stacked in a rectangular box with a hollow vertical body and an open top for cone placement. The dispensing platform is located below the cone storage area and consists of a fork formed of two fingers, which are horizontal and are spaced from each other to pass the body of the cone [7]. Diagrams of the detailed design of the device are included in Appendix C - 2.

2.3.3.3 The Roadrunner

The Roadrunner by Royal Truck & Equipment Inc. consists of a traffic cone dispensing mechanism as well as a retrieval mechanism. The dispensing mechanism utilizes a curved track system that allows the user to load each cone separately at the higher end of the track. Once loaded, the holders keeping the cone in place will release, which allows the cone to smoothly drop down onto the road with the guidance of the track. The mechanism can be easily mounted onto the side of a regular-sized truck [8].

3. Decision Making Standards

3.1 Requirements [9]

The core objectives considered in the design process are listed in the below in the Table 1. As we separate the design by its functionality, some supplementary objectives are considered in addition to the core objectives to help select the best choice for each function. These supplementary objectives can be found in the discussion of alternatives and decision-making process.

Table 1: Core Objectives of the Design with Corresponding Metrics

Objective	Parameter	Approach	Scale	Unit
Inexpensive	Direct material Cost	Survey	0-230	$CAD
Light	Weight of the machine	Survey	0-8	kg
Robust	Stiffness of material	Survey		
Accurate	Device Precision	Survey +Testing		
	Machine positioning error			
Fast	Motor speed, estimated time for one operation			
Energy efficient	Voltage/Power needed	Calculation, survey		

3.2 Constraints

- The entire prototype must completely fit within 50cm * 50cm * 50cm dimension at all operation times
- The weight of the machine must not exceed 8kg (including the cones)
- The total prototype cost must not exceed $230 CAD (before shipment and tax)
- The machine must use its own on-board power supply during any operation

- The operation time must not exceed 3 minutes. the time required for setting up the machine before the operation is 2 minutes.
- The machine must be autonomous without any interaction with PC or remote control
- The machine must have an easily accessible emergency STOP bottom to stop all mechanical moving parts immediately
- Two cones deployed on top of the crack must not contact each other and covers at least 5 cm of the crack each.
- After deploying on a hole, if there is another hole immediate next to it, within 15 cm separation, the machine must not deploy any cone over this immediate next hole; After deploying on a hole, if there is another crack immediate next to it, within 20 cm separation, the machine must not deploy any cone over the crack.
- After deploying on a crack, if there is another hole immediate next to it, within 20 cm separation, the machine must not deploy any cone over the immediate next hole; After deploying on a crack, if there is another crack immediate next to it, within 10 cm separation, the machine must not deploy any cone on the crack [9]

3.3 Values

The following characteristics are what the team will strive for during the conceptual design and construction of the machine:

Practical and Realistic: We try to ensure our estimates of the performance of the conceptual design closer to the actual performance. We do this through simulation and testing with the prototypes, researching and more rigorous modelling.

Efficiency: We try to eliminate the unnecessary design in the robot. For example, to reduce cost, size, weight, time or procedure. We do this through researching, brainstorming, and optimization methods in physics, circuits, material science etc. [9]

4. Budget

Table 2: Budget of Electromechanical Subsystem

Function	Material	Portion/Number Used on the Robot	Total Cost	Portion Cost
Basic Structure	S4S OAK1*2*4	80%	6.2	4.96
	1/4*4*2HB	100%	3.98	3.98
	Hinges Everbilt*50	10%	17.59	1.759
	Screws 6*1" 20pk	50%	2.79	1.395
	Screws [A]1/8"	20%	4.14	0.828
	Screws [A]1/6"	20%	3.82	1.91
	Nut*50	20%	12.5	2.5
Cone Dispensing System	8" TIE NAT	10%	3.48	0.348
	Gear (3D Printing) *2	50%	0.12	0.06
	Gear Rack	4%	25.99	1.0396
	FLASHING Alum 14"*5'	10%	13.99	1.399
	Aluminum (Paulin Angle)	30%	12.86	3.858
	Hinges Everbilt*20	10%	19.4	1.94
	IR Sensor	100%	0.425517	0.425517
	Foam Pad Beige16*16*1	5%	7.99	0.3995
	Continuous Servo motor	100%	7.772	7.772
	MG996R High Torque Servo Motor	100%	1.92357	1.92357
Driving System	Vex Mecanum Wheel*4	100%	77.99	77.99

	4*Adaptor (3D printint)	100%	1.2	1.2
	Straight Shaft Coupler(4mm-6mm)	100%		
	6mm D Shaft	20%	1.65155	0.33031
	DC Motor*4	300%	3.78885	11.36655
	Motor with encoder*1	100%	11.2694	11.2694
	Screws M3*6mm *4	20%	11.16	2.232
	Screws M3*10mm*3	20%	8.37	1.674
Sensing System	IR Sensor	500%	0.425517	2.127585

Table 3: Budget of Circuit Subsystem

Material	Portion/Number Used on the Robot	Total Cost	Portion Cost
PCB Board 7*9cm	300%	0.34974	1.04922
PCB Board 3*7 cm	100%	0.157383	0.157383
L298N Motor Driver Board	200%	0.9715	1.943
Shenzhen angled DC Motor (indicator)	100%	0.5829	0.5829
Switch	100%	0.283678	0.283678
Capacitors	3%	1.84585	0.0553755
Voltage Regulator L7805	10%	1.06865	0.106865
Voltage Regulator LM338	200%	0.07772	0.15544
Transistor TIP142	200%	0.068005	0.13601
Transistor TIP147	200%	0.069948	0.139896
Resistors	1.33%	1.851679	0.024689053
Amazon Basics High Cap Rechargeable AA	100%	25.281387	25.281387

Table 4: Budget of Microcontroller Subsystem

Material	Portion/Number Used on the Robot	Total Cost	Portion Cost
PIC	100%	55	55

The overall total budget of our robot is 227.48 CAD [10], with the Mecanum wheels being the most expensive component, followed by the PIC microcontroller board and Amazon Basics Rechargeable AA batteries.

5. Division of Problem

To conquer this challenge, the task of designing and fabricating the robot is properly divided into three subsystems to split between the three group members as a starting point. The three subsystems that will be tackled individually are electromechanical, circuits and microcontroller programming. Within each subsystem, the project is further divided into functional areas such as driving system and sensing system, which will be further explained in the following sections. As each member gets more familiar and comfortable with their own subsection, the design will converge into one single machine that integrates all three subsystems.

6. Electromechanical

6.1 Basic Structure

The basic framework and structure of the robot should be able to hold the traffic cones in place, contain the driving system, withstand the weight and secure the position of the PIC board and various electronic components. The structure should not violate any constraints on size, weight, materials, etc. as listed above.

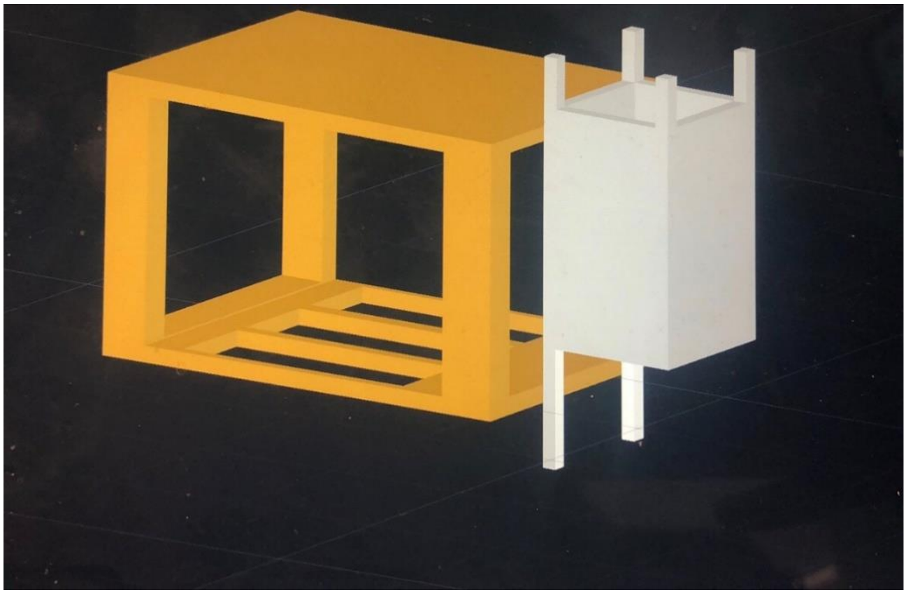


Figure 3. CAD Model of Main Body Structure

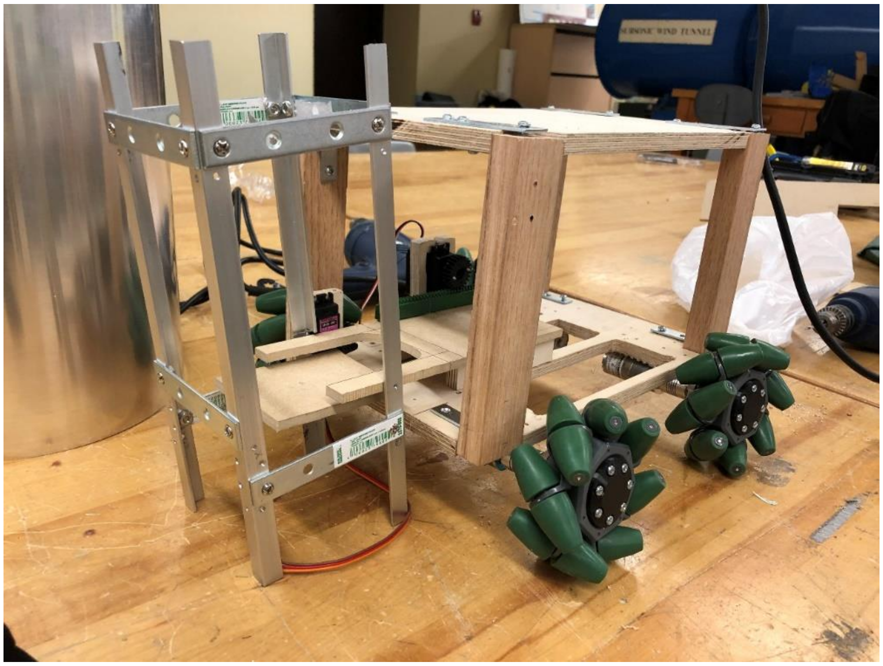


Figure 4. Physical Appearance of the General Structure

The basic structure for this robot consists of a cuboid body constructed with ply wood and an aluminum tubing for holding the cone. The dimension of the cuboid body is 30 cm* 20.8 cm* 20cm, and the tube is positioned at the central back of the body, with a dimension of 10cm* 10cm* 27.5cm. Both the upper and lower layers have a partially hollow design for holding the PIC board and other electronic components, with the purpose of reducing its weight. The layers and each supporting pillar are connected by a 90-degree hinge with 4 nails.

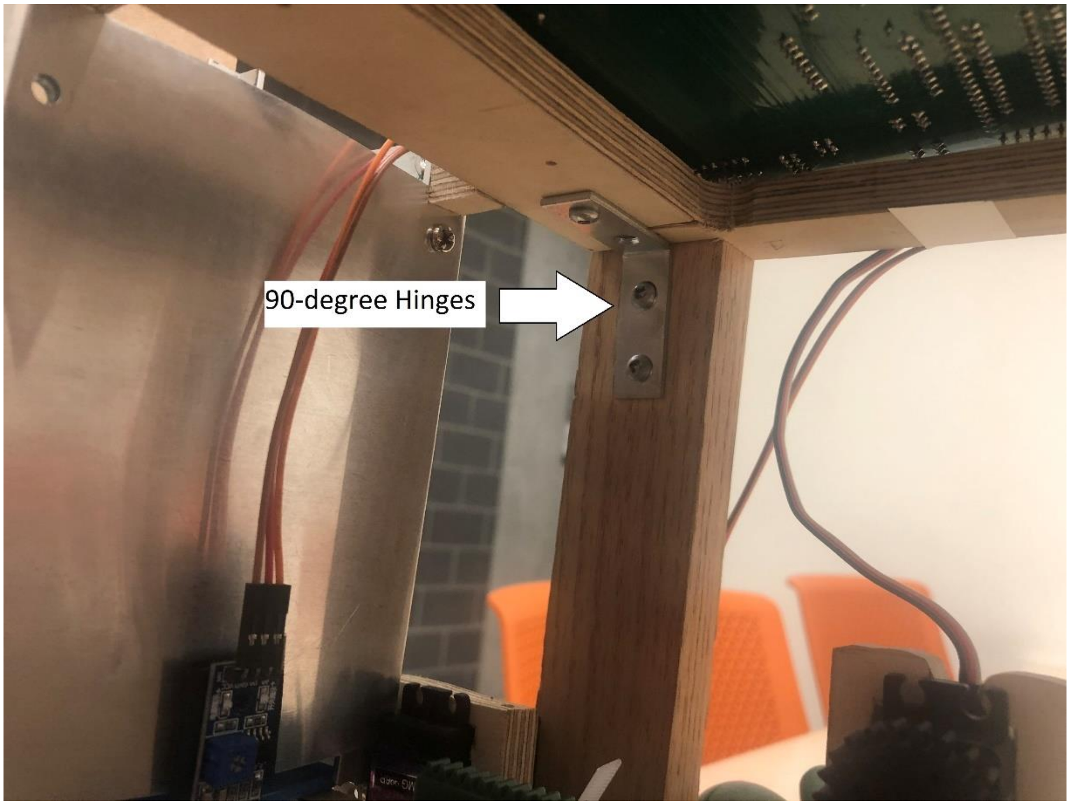


Figure 5. Picture of Hinges Connecting the Layer and Supporting Pillars

In comparison with the proposed design at the beginning of this project, the basic shape remains the same. Some adjustments are due to the issues arose during the construction process:

<u>Issues #1</u>

The proposed material for the body is aluminum, and wood for cone holder. During construction, it is discovered that the body need substantial adjustments for compromising with circuits design. Therefore, aluminum is not the best choice for the body. Also, cone holder does not need to be robust as its main function is to ensure the cones are in position.

<u>Adjustment #1</u>

It is discovered that ply wood is easier to calibrate and make adjustment (such as cutting, drilling, etc.), in which it complies with the body requirements for easy calibration. For cone holder, aluminum sheet and four 90-degree stripes are utilized as it is light-weighted than wood.

Issue #2

The proposed size for the robot is 34.2cm*35cm*30cm. During the actual construction, it is discovered that the robot can be reduced in width to achieve the requirement of 30 cm for extra feature.

Adjustment #2

The width of the body is constructed to be 20.8cm, which will be under 30cm adding the width four wheels at its side. Though, the length of the is made longer than proposed length due to its unique design of sliding door. Height remain unchanged from proposal.

Suggestions for Improvement of the Subsystem

1). The basic structure of the robot can be reduced in size to achieve the extra design features of compactness and portability. For example, length of the body with cone holder could be further reduced by changing its overall structure such as positioning the holder inside the body.
2). The weight of the robot could be further reduced by replacing the wooden pillars with some light-weighted wood.
3). For the purpose of elegance, a box for hiding the wires could be included on its side to store the wires.

6.2 Driving System

The driving system of this robot should allow it to move forward, backward and sideways for it to deploy two cones consecutively on the crack without being in contact with each other.

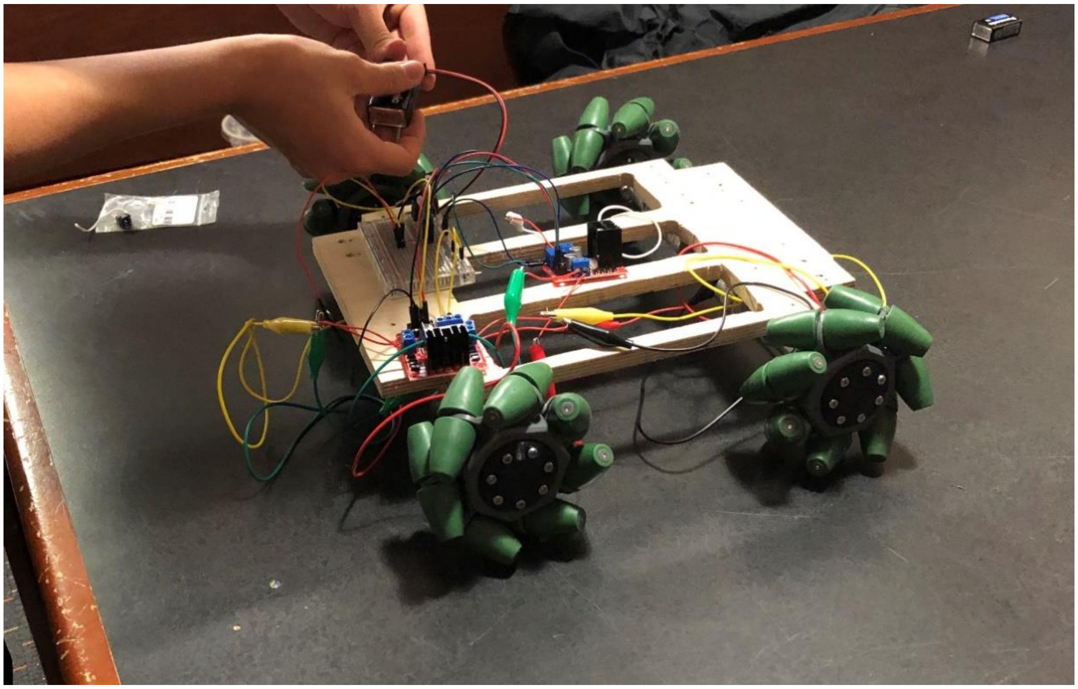


Figure 6. Omni Directional Driving System with Mecanum Wheels

The driving system chosen for the robot is omni directional driving system, which allows it to perform all directional movements. The wheels are Mecanum wheels, such that the rollers are 45 degrees positioned on each wheel. Each wheel is connected to a 3D printed adaptor due to its square-shaped opening at the wheel, which is not compatible with our D-shaped motor. A 4mm-6mm coupler and shaft are also used in connection to the DC motor. One of the wheels is attached to the DC motor with encode for distance record. Because the tolerance of error is +-10cm and four wheels are identical, multiple encoders are not necessary.

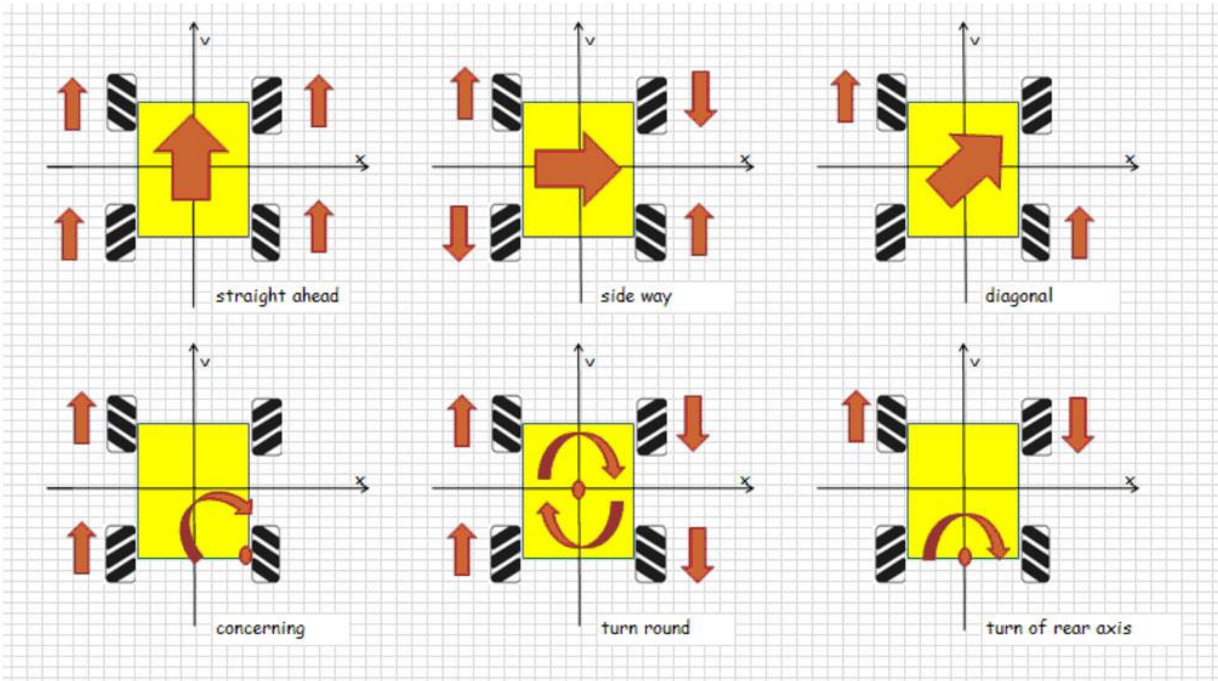
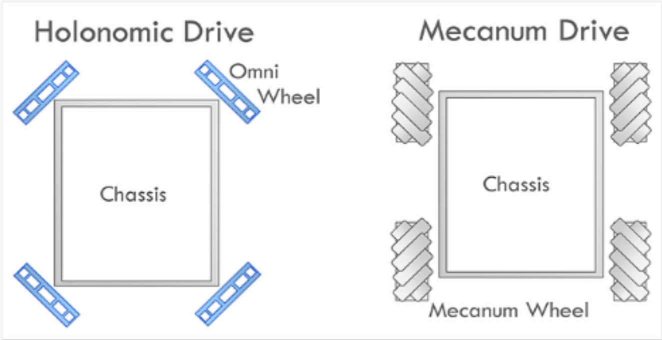


Figure 7. Directional Control of Mecanum Wheels Mounted Parallel to the Main Body

Issue #1

The proposed design for the wheels is omni wheel (Figure 8.). During the simulation, there are two scenarios that occurred, the first one is such that we have to increase the size of the frame, that compensates for the 45-degree installation of the wheels, which would also increase the overall weight of the robot; The other scenario is that there would not be enough space for the cone holder to be positioned in between two back omni wheels. It is resulted that such wheels are too inconvenient to install. (Refer to Appendix D)



(A) Omni Wheel (B) Mecanum Wheel

Figure 8. Omni Wheels vs Mecanum Wheels

Adjustment #1

The actual design consists of four Mecanum wheels, which has 45-degree rollers on the wheel. It allows the robot to move in all directions like omni wheels without specific installation requirements. This arrangement is easier to install and calibrate as shown in figure. ().

Issue #2

The opening of such Mecanum wheels is shaped in rectangular, which is nor compatible with the D-shaped DC motor.

Adjustment #2

During the construction of the driving system, a 3D printed adaptor is used to connect each wheel to a motor hub. One 4mm-6mm coupler and shaft are also used to connect the 4mm D-shaped motor with 6mm hub.

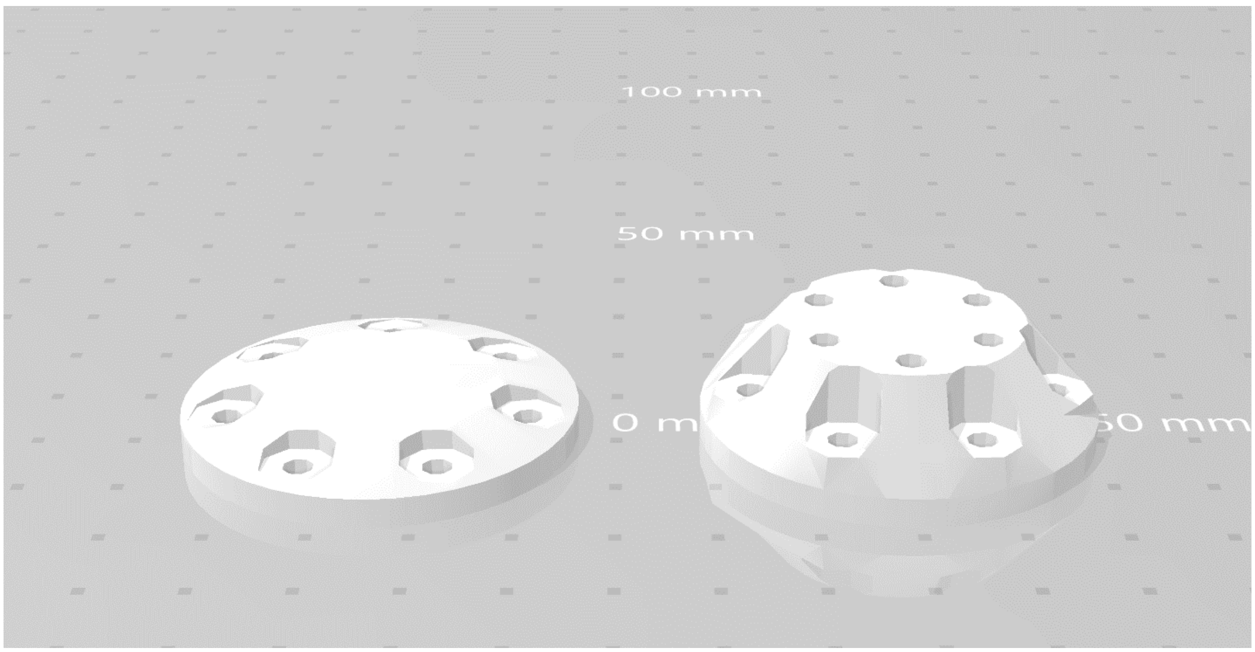


Figure 9. CAD Drawing of the 3D printed Vex Wheel Adaptor

Suggestions for Improvement of the Subsystem

1) The overall design of this driving system is capable of performing all the functions required. On the other hand, the fluctuation it caused has affected the accuracy of sensor detection during the testing process. Future improvement could be done by installing shock absorber to reduce the vibration thereby increasing the accuracy.
2) The connection of each wheel can be reduced by redesigning the 3D printed adaptor so that it is compatible with 4 mm D-shaped motor. By doing so, the coupler and shaft can be eliminated so that the connection part between the internal motor and wheel can be

reduced. Therefore, the wheels can be positioned closer to the robot body, with withstanding more force exerted by the weight of the robot.

6.3 Cone Dispensing Mechanism

The cone dispensing system should be able to hold 12 cones in position and perform the functionality of deploying one each time with holding the rest in place while operating.

Figure 10. Physical Appearance of Cone Holder [Front]

Figure 11. Physical Appearance of Cone Holder [Top]

The design for the cone dispensing system of the robot consists of one sliding door controlled by rack and pinion, one rotating door controlled by high torque servo motor, and an aluminum cone hold that positioned at the central back of the robot body.

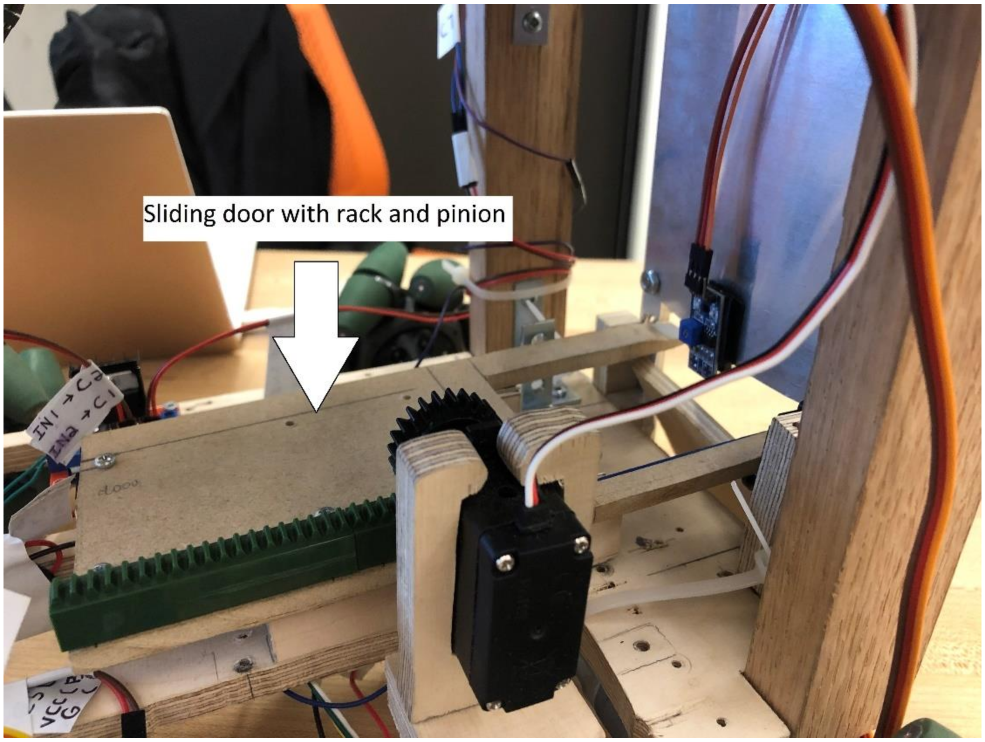


Figure 12. The physical appearance of sliding door with rack and pinion

While operating, the rotating door positions at 180 degrees (horizontal) to hold the cone in the holder. The sliding door slides out through rotation of gears, then slide in by reversing the direction of rotation to hold the second last cone up by its fork-like shape. Then the bottom door opens to a 90-degree position to drop one cone. Two servo motors are controlled directly by duty cycle. One IR sensor is used to detect whether there is cone left in the holder (The states are shown below).

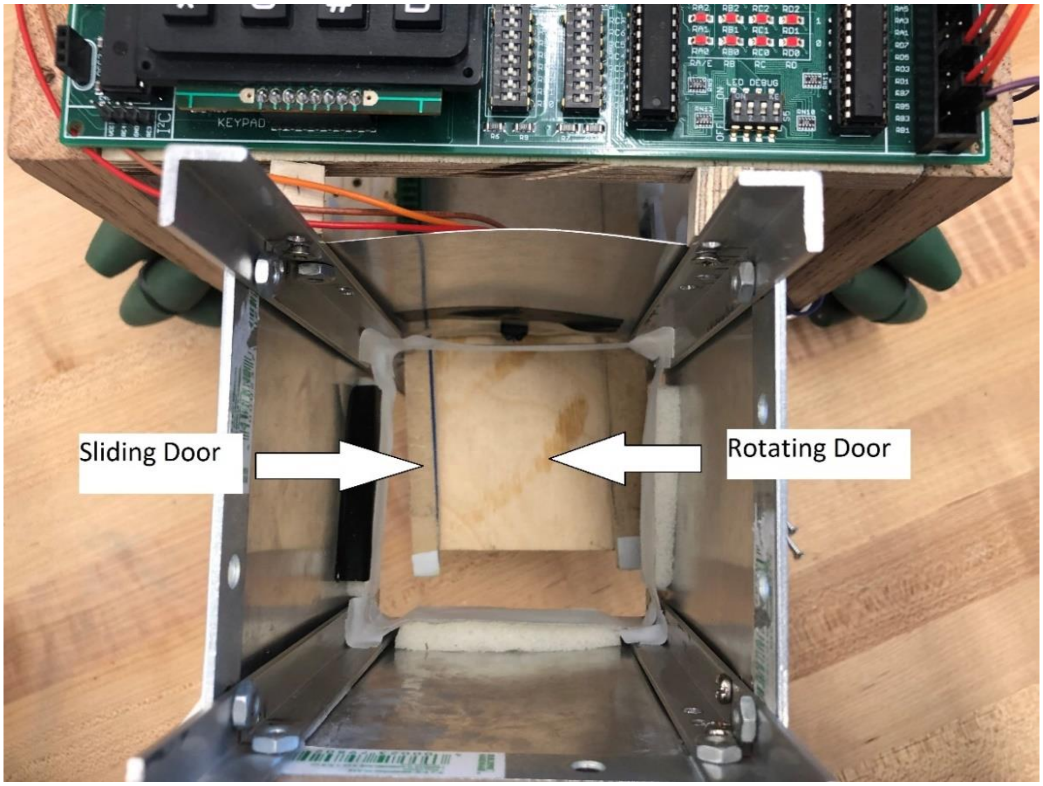


Figure 13. State #1: All cones are positioned above the sliding door

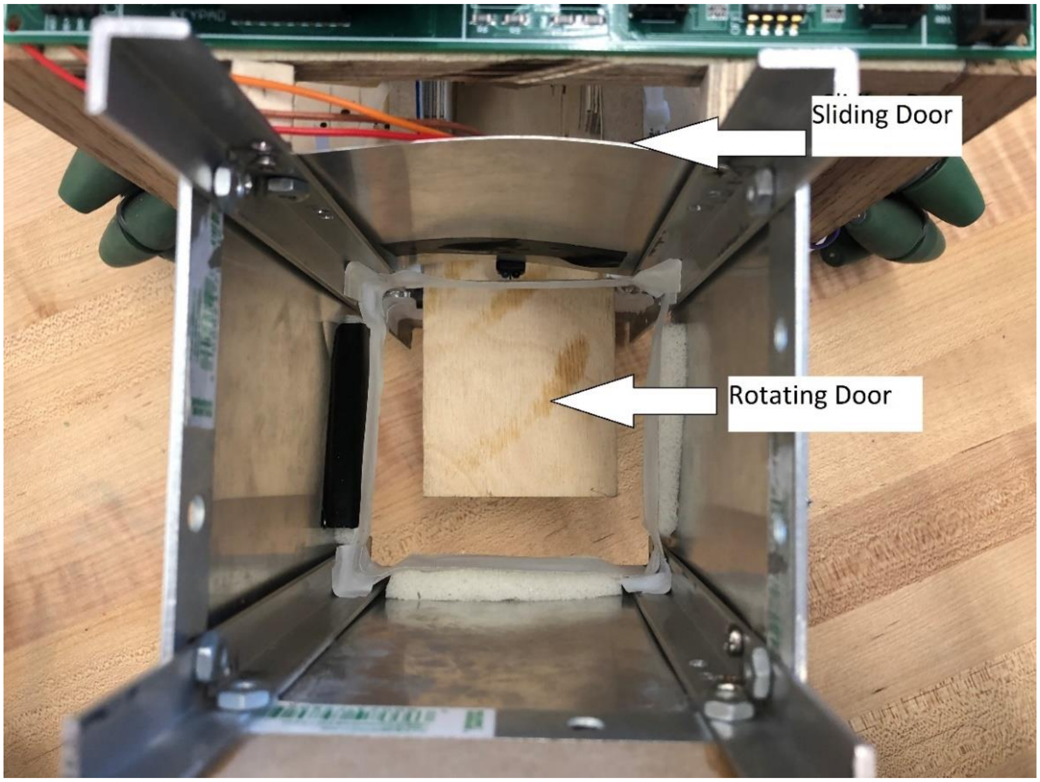


Figure 14. State #2: The sliding door slides out letting all cones sitting on the rotating door

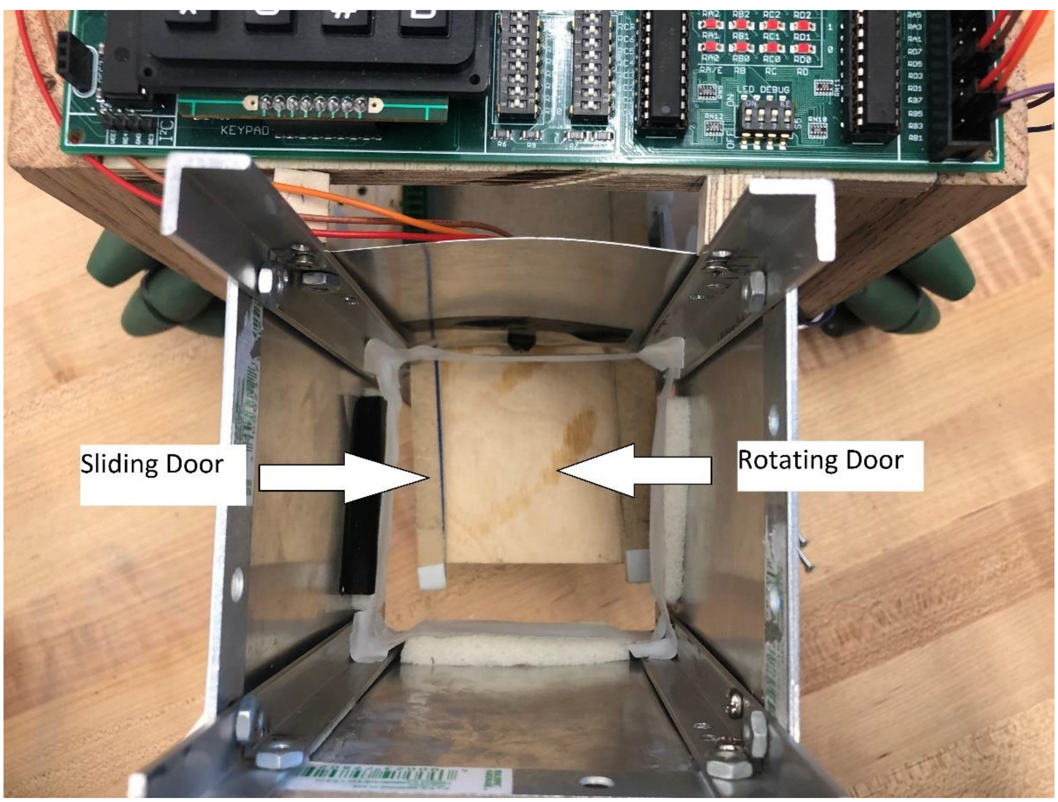


Figure 15. State #3: Sliding door slides in and hold up the upper cones except the first one

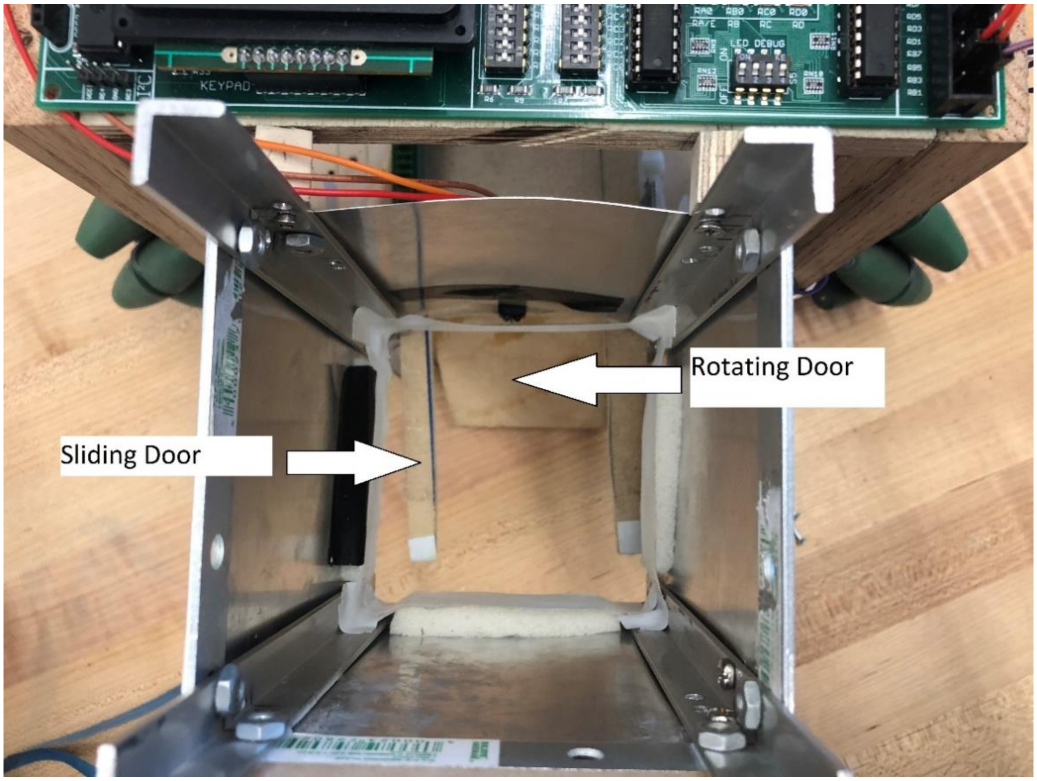


Figure 16. State #4: The rotating door open to 90 degree to drop one cone

<u>Issue #1</u>

In comparison with the proposed design, the alternating doors mechanism (Figure 17.) has been modified due to the results of prototype simulation. The prototype is made of 3 plastic pieces, connected by a wood stick acting as shaft. It resulted that the middle board did not have a sufficient length for the door to go in 9 cm in order to fit in between two stacked cones. (Refer to Appendix D)

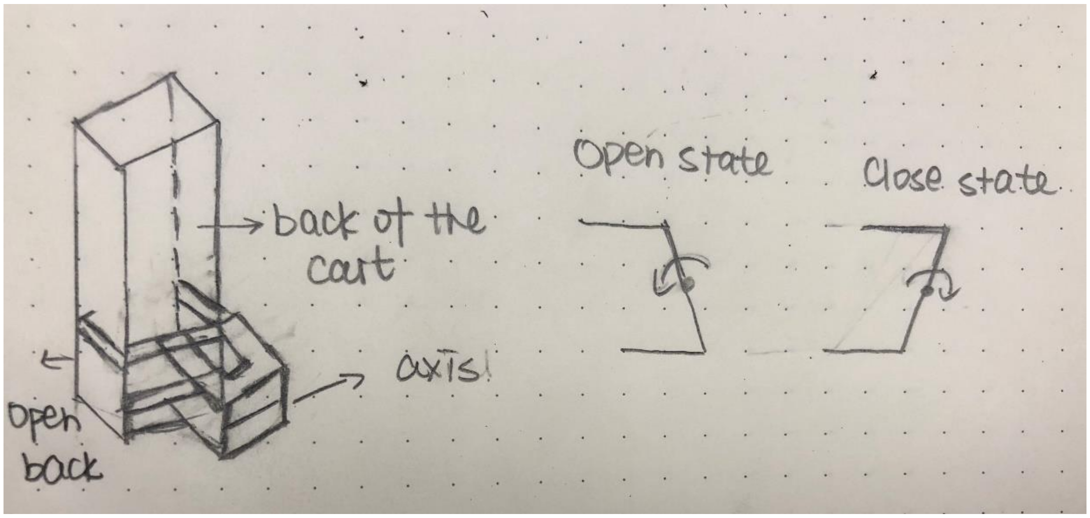


Figure 17. Diagram Illustrating the Alternating Doors Mechanism

<u>Adjustment #1</u>

The actual design follows the arrangement described above.(Figure 17.)

<u>Issue #2</u>

The compatible gear with the rack set has a rectangular opening in the centre, which is difficult to connect with the servo motor.

<u>Adjustment #2</u>

A designed 3D CAD model of gear is obtained from Myhal Fabrication Facility (shown in figure ()), and it is connected through the connector piece of the servo motor. Servo motor is placed in a hand-made holder shown in figure (), to hold it in place where the gear can perfectly align with the rack constructed on the sliding door.

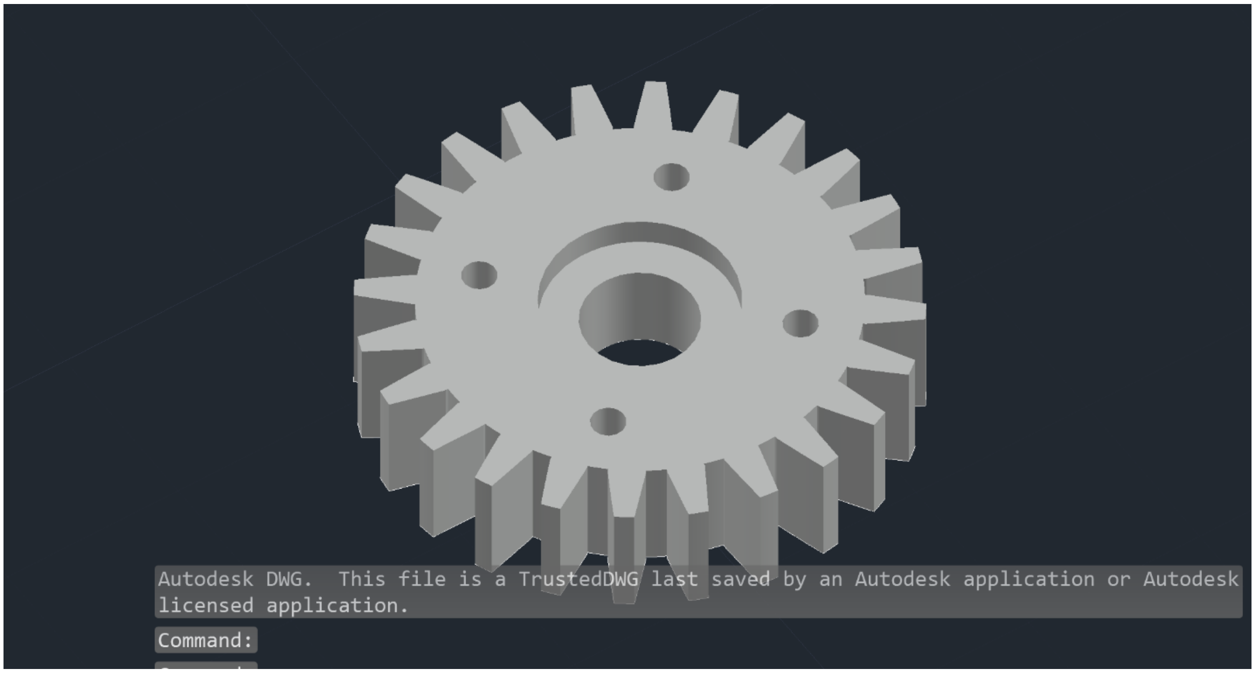


Figure 18. 3D CAD Model of the Gear Compatible with the Rack and Servo Motor

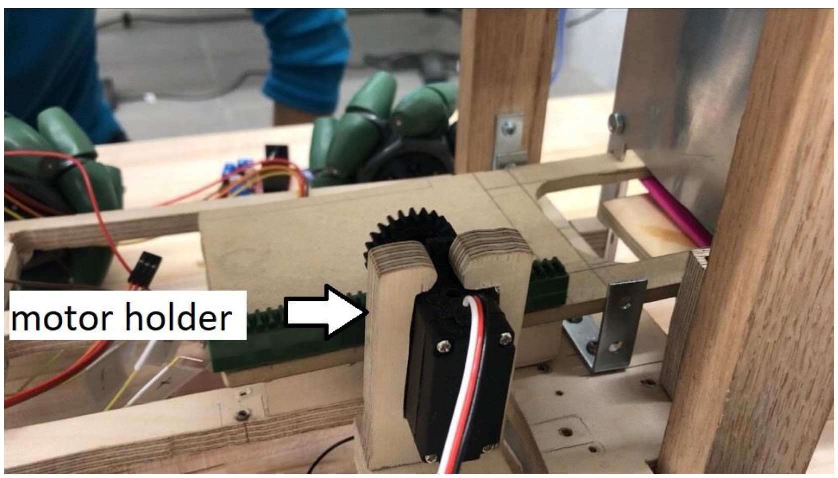


Figure 19. Physical Appearance of the Servo Motor and the Holder

<u>Issue #3</u>

For the detection of cones inside the holder, we originally decided on using a pressure sensor located on the door to detect whether there is cone left through the reading of weight. During construction, it is discovered that pressure sensor is costly and difficult to calibrate through microcontroller.

<u>Adjustment #3</u>

The replacement for pressure sensor is one IR sensor that positioned at the center bottom of the cone holder (on the aluminum sheet) shown below. The sensor will light on when there is cone inside and light off when no cone left.

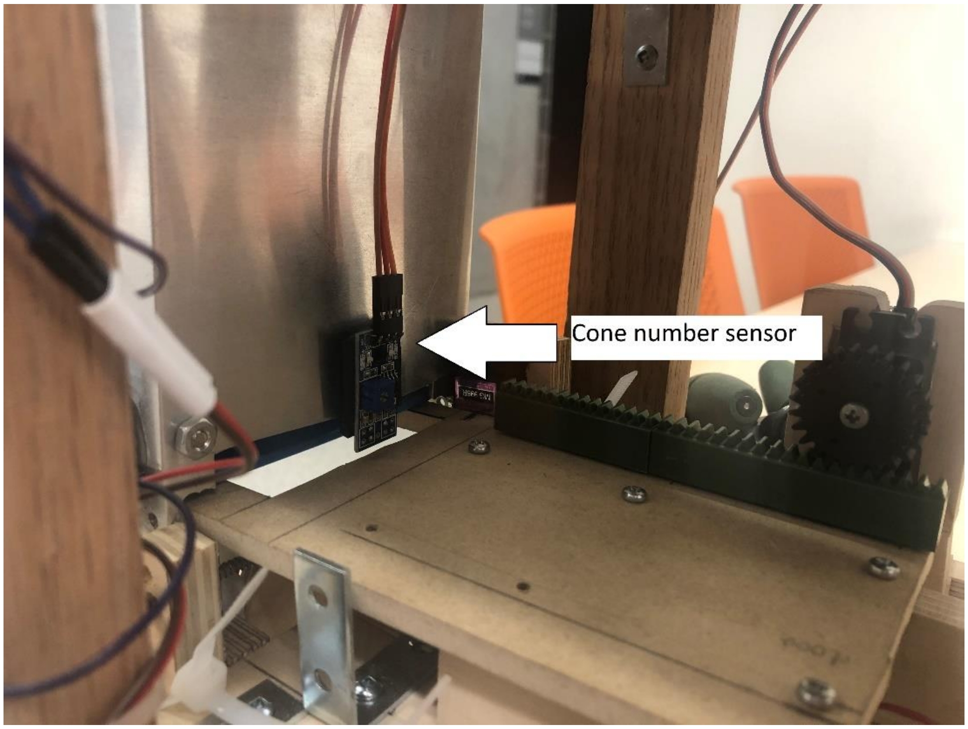


Figure 20. Physical Arrangement of IR Sensor Detecting Cone Number

<u>Suggestions for Improvement of the Subsystem</u>

1. The size of the cone holder can be reduced by 1cm per side to ensure the cone slide properly though four 90-degree aluminum pillars. (*note: the screws head need to be further polished or replaced by epoxy)
2. In order for the cone to be in position without rotating while being dropped, the future improvement could be done on constructing a track shaped as a quarter circle at each side of the rotating door. While operating, the track will ensure the cone slide from the door smoothly to improve accuracy. The shape will ensure the door and track are not in contact with the dropped cone.

6.4 Sensing System

The sensing system should be able to detect any hole or crack. For correctly dropping the cone on the right position, it should also recognize its position with an acceptable range of error within the lane. Considering the following constraints:

1. cones cannot overlap with each other.
2. each cone deployed on a crack must cover at least 5 cm of the crack

3. cones cannot be on or outside the lane.

4. two cones should be deployed on a crack.

We generalized the cases into three with the arrangement of sensors as following. In addition, the sensing system should also perform the line following function.

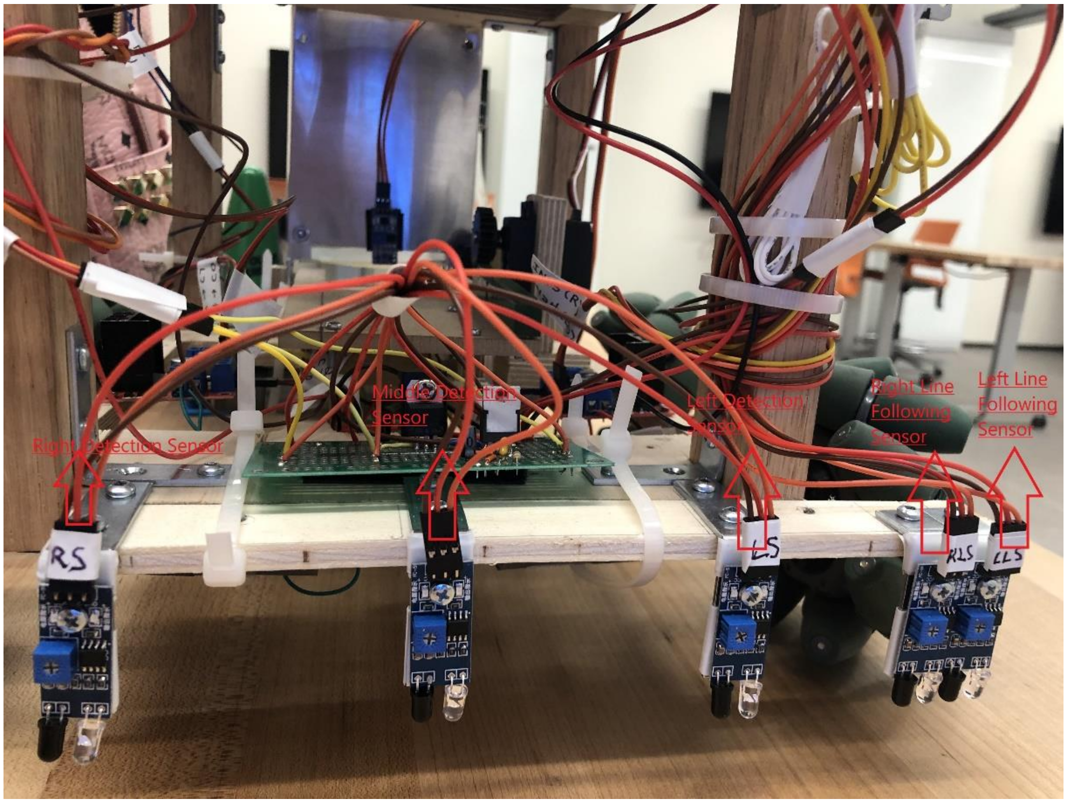


Figure 21. Picture of Detection Sensor Arrangement

The number of sensors chosen in the system is 5, such that 2 on the left of the robot are for line following, with a distance <= 2cm between two sensors. The other 3 sensors are for hole/crack detection. The position of the sensor is calculated (Table 5), define (1) as the right detection sensor, (2) as the middle sensor; (3) as the left detection sensor:

Table 5: The relative position of the first left detection sensor is 5cm from the line following sensor; The second one is 7.5cm from the first detection sensor; The third one is also 7.5cm, from the second detection sensor.

Steps vs. Conditions	(1) & (2) =0	(2) & (3) =0	(1) & (2) & (3) = 0	(2) = 0
1	Right Move 1.5cm	Left Move 1.5cm	Left Move 4.5cm	Drop 1 cone
2	Drop 1 cone	Drop 1 cone	Drop 1 cone	/

3	Left Move 9cm	Right Move 9cm	Right Move 9cm	/
4	Drop 1 cone	Drop 1 cone	Drop 1 cone	/
5	Right Move 7.5cm	Left Move 7.5cm	Left Move 4.5cm	/

The sensors are positioned at the front of the robot through one 25cm piece of wood and 90-degree hinges.

Issue #1

In the proposed design, a sensor mover is introduced, such that the sensor is moved through the rotation of rack and pinion. During construction, the process is difficult to achieve and require sophisticated calculations and manufacturing (refer to Appendix D)

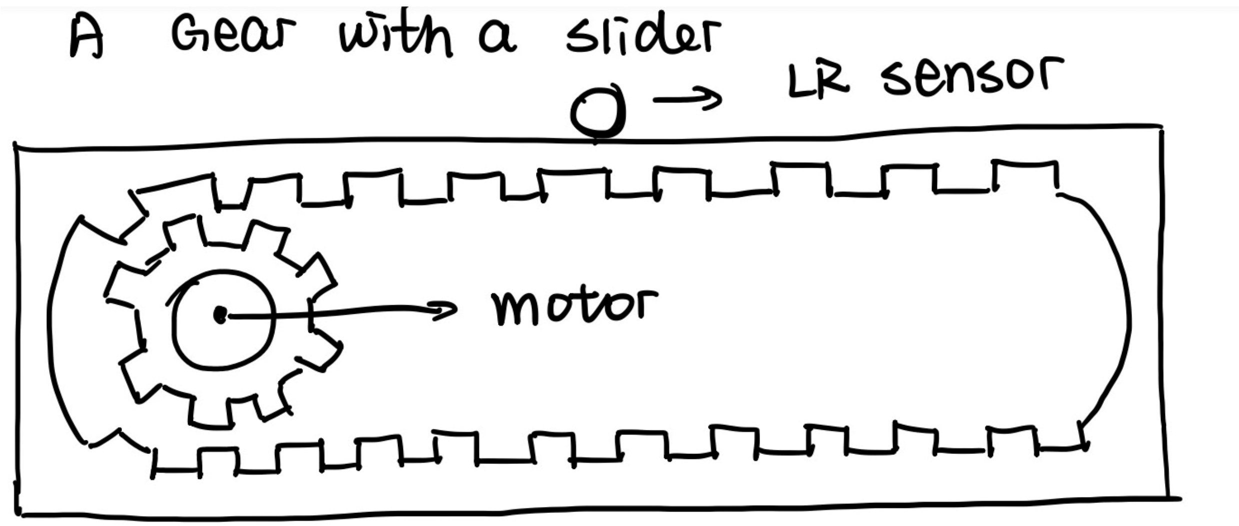


Figure 22. Diagram Illustration of the Slider Mechanism

Adjustment #1

The modified design consists of 5 sensors, such that 2 are for line following and 3 are for hole/crack detection. Detailed calculation shown in Table 5. The middle sensor is purposely constructed further back than the left and right detection sensors so that the side sensors would detect the crack in advance of the middle sensor thereby crack would not be recognized as a hole.

Suggestions for Improvement of the Subsystem

1. The sensors are too sensitive to external environment and lighting. Some future improvements could be done by adding cylindrical LED specifically around the IR receiver to ensure the accuracy of detection.

2. To reduce risks of not detecting hole or crack, adding another row of sensors will help increase the accuracy and reduce the uncertainties.

6.5 Indicating System

The indicating system should consists of an indicator that provides different forms of signal to notify for hole detection or crack detection.

The indicator used in the robot design is a laser cut rotating two-sided flag with hollow word "crack" and "hole" on each end, which is controlled by DC motor. H-Bridge circuit allows it to rotate to hold the right "word" up based on the obstacle (hole or crack) detected.

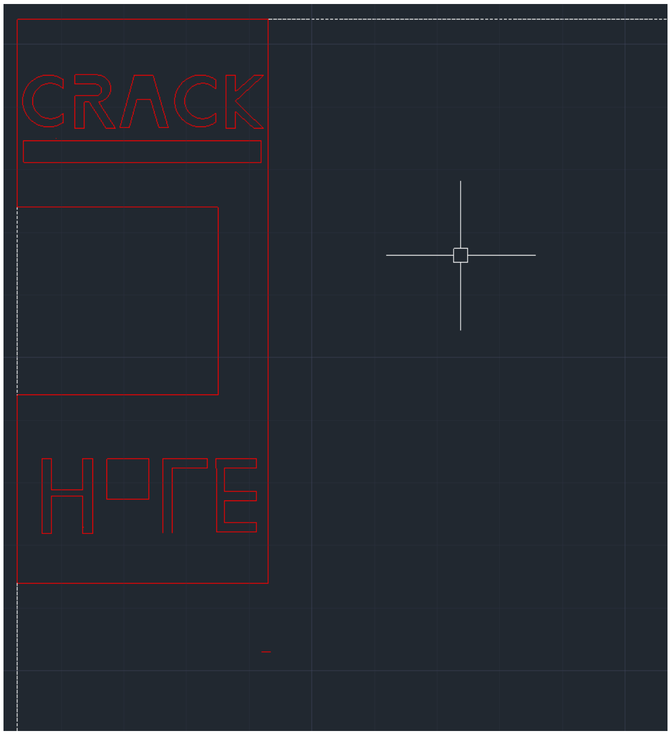


Figure 23. CAD Model of Indicator Flag

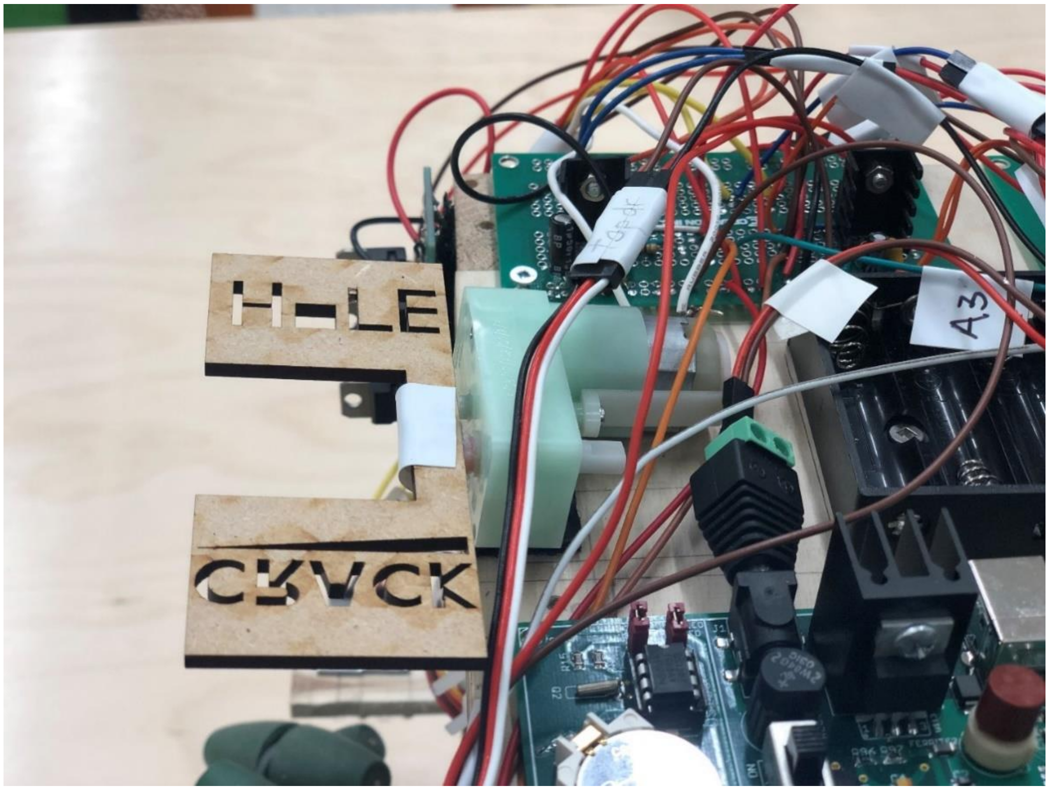


Figure 24. Picture of Laser Cut Double Flag

<u>Suggestions for Improvement of the Subsystem</u>

1. Future improvement could be done on securing the flag on the motor. For example, using screws and nuts
2. Replace the DC motor with servo motor would make the calibration process easier and more accurate as the rotation of servo motor is governed by a defined angle, it would be able to replicate an exact angle.

7. Circuits

7.1 Driving System Direction and Speed Control

In order to control the speed and direction that the robot drives at, the circuit that powers each Mecanum wheel needs to be set up so that the rotation of the motors can be controlled by output signals from the PIC microcontroller. To achieve this goal, H-Bridge circuits are utilized to translate high and low signals from the PIC to clockwise and counter-clockwise rotation of the driving motors.

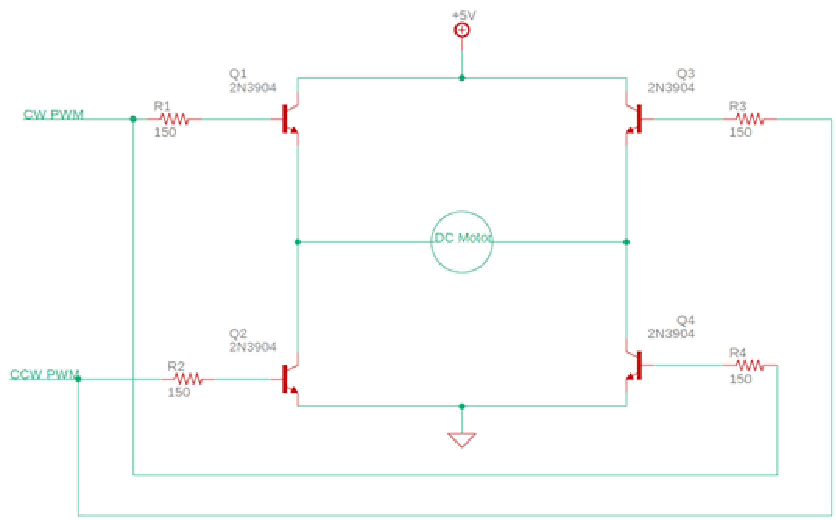


Figure 25. H-Bridge Motor Driver Circuit

As shown in Figure 25, if the clockwise signal is at 5V and the counter-clockwise signal is at 0V, the connected motor will spin in the clockwise direction, and vice versa. However, given the configuration in the diagram, it can be seen that if the four wheels were to be controlled separately, a total of 8 pins would be needed on the microcontroller. To prevent this redundant use of output pins from the microcontroller board, the first solution that we came to was to share the PWM signals given to DC motors powering diagonal wheels. This approach was made possible due to the rotation combination of the Mecanum wheels we are using.

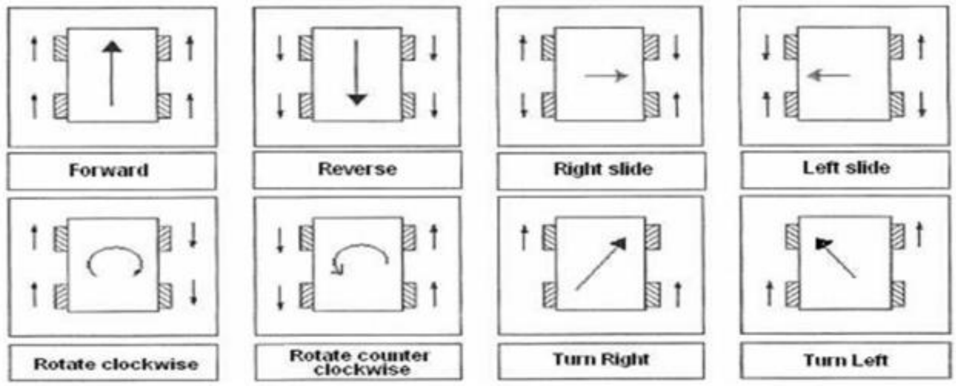


Figure 26. Mecanum Wheel Rotation Combination and Corresponding Motion

Under normal operation routines, the robot will only perform forward, reverse, right slide and left slide motions. As shown in Figure 26, in these four motion modes the diagonal wheels will always rotate in the same direction. As a result, by connecting the DC motors powering these diagonal wheels in parallel in the H-Bridge circuit, each pair of diagonal motors are controlled by the same signals from the PIC. This reduces the number of pins required from the microcontroller board by half.

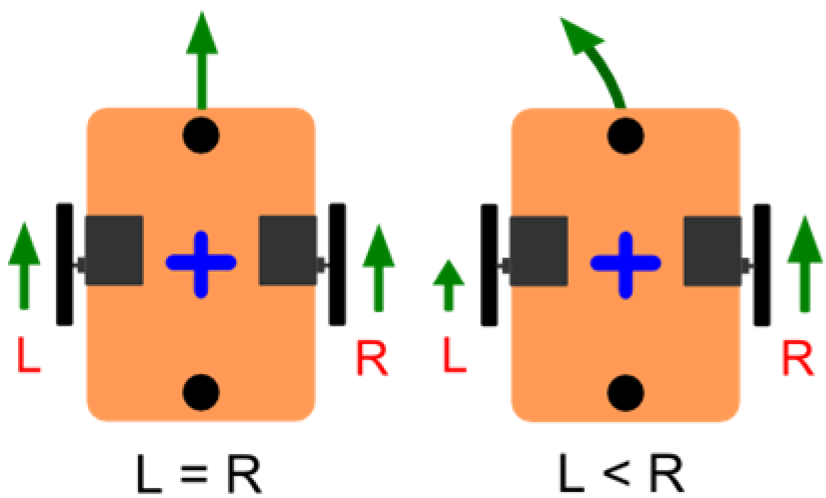


Figure 27. Left and Right Adjustments via Speed Difference

Nevertheless, we discovered that this design has a fatal flaw. In order to effectively follow the side lines of the track, the robot will need to make small adjustments to the left or right while maintaining forward motion. As shown in Figure 27, this can only be achieved if there is a difference in rotational speed between the left-side wheels and right-side wheels of the robot. Having the same PWM signal for diagonal DC motors satisfies our directional control requirements, but conflicts with the speed control adjustments that are crucial to line following.

Since both pairs of diagonal wheels are controlled using the same PWM signals, it means that they will have a relatively similar rotational speed. Thus, it is impossible to make small shifts while driving forward as we intended with the parallel setup.

To deal with this issue, the approach we decided to take was to use three pairs of PWM signals to control the four DC motors. While the front-left and back-right wheels are still controlled by the same signals and maintains a similar speed, the front-right and back-left wheels are controlled separately by two pairs of PWM signals. As a result, a difference of speed between the left-side and the right-side of the robot can be achieved by varying the duty cycle of the two independently controlled wheels utilizing a reduced total of 6 pins from the microcontroller board.

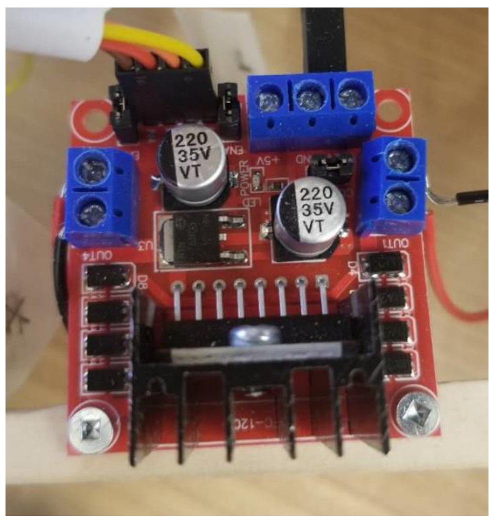


Figure 28. L298N Motor Driver Module

Furthermore, to ensure consistency between the H-Bridge circuits used for the driving system, the premade L298N motor driver modules were used instead of handmade H-Bridge circuits. This eliminates the possible fluctuations in voltage supplied to the DC motors due to poor connections between electrical components.

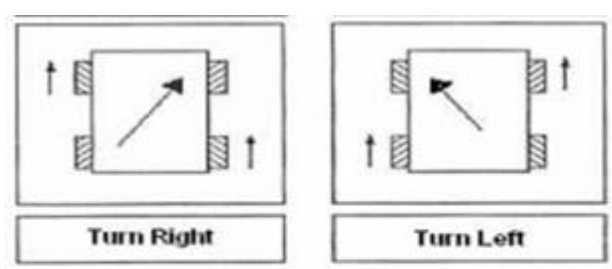


Figure 29. Left and Right Adjustment via Characteristics of Mecanum Wheels

Looking back to the final solution that we settled on, an improvement that could be made is to keep the diagonal setup with only 4 pins and adjust the direction of the robot using the benefits of Mecanum wheels instead. If the left and right adjustments during line following were achieved using the method indicated in Figure 29, the setup that we initially came up with would have remained an effective approach, which saves another 2 pins for other inputs and outputs.

7.2 Detection and Line Following Sensors

The sensors responsible for crack and hole detection as well as trajectory maintenance are IR reflective sensors. These sensors are the best choice for our project because we do not need to detect physical objects. Furthermore, out of all the light sensors in a reasonable price range within the budgeting limit, IR sensors have the best price to performance ratio.

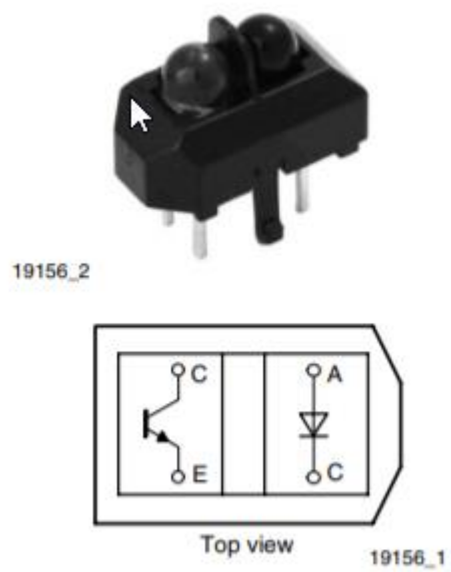


Figure 30. Physical and Circuit Diagram of TCRT5000L IR Sensors

The particular type of IR sensors that were used composed of IR LEDs and IR transistor pairs as shown in Figure 30. The IR LED transmits a beam of IR light into one direction, and the IR transistor will have a different discharge response based on the intensity of the IR light bounced backed after contacting a surface. This sensing mechanism is perfect for crack and hole detection for dark surfaces like the hockey tape that will be used, which reflects a minimum amount of IR light.

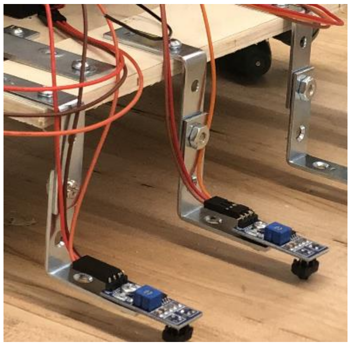


Figure 31. The TCRT5000L Sensors were Installed to Maintain a 2.5 mm Distance to the Ground

The first model of IR sensors that we installed onto the robot were TCRT5000L IR sensors. The TCRT5000L operates very accurately from an operating distance of 1mm to 8mm, with the peak performance distance being 2.5mm. Despite its high-performance ratings, the downfall of these sensors was that they were too sensitive for our operations. Due to constant vibrations of the robot while in motion, the distance between the sensor and the ground are constantly fluctuating. Therefore, the optimal operating range of the sensor cannot be set as the detecting range changes in its already small range of operating distance. As a result, whenever the robot approaches a hole or crack, the sensor constantly switches between reading high and low.

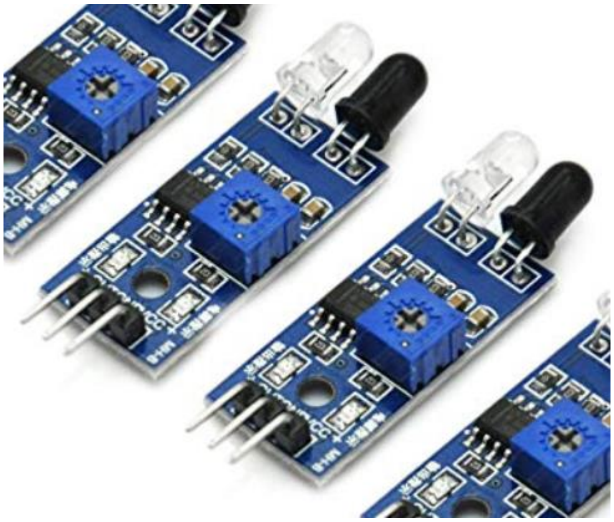


Figure 32. EK1254x5C IR Reflective Sensor Modules

As an alternative, we chose to replace the TCRT5000Ls with EK1254x5C IR sensor modules instead. The EK1254x5C modules have a much flexible operating range of 2 – 30cm. So, we were able to set a feasible sensitivity within this range so that it reads stable inputs under vibrations.

Nevertheless, IR sensors are still very susceptible to ambient lighting and environment changes, which means that it requires careful calibration when placed in a new lighting and environmental condition. This means that a possible point of improvement is to add more security to the sensor readings by using analog IR sensors instead of digital ones. Multiple readings can be taken over a short period of time with analog sensors until they can be trusted. By taking analog readings, filtering processes can also be added using the microcontroller to ensure the readings are relatively reliable rather than completely trusting the digital inputs. In addition, more sensors could be added to the front of the to ensure accurate detection of holes and cracks. This adds an enhanced layer of protection because it levitates the pressure of relying solely on the input from a single sensor in its position.

7.3 Directional Control of Indicator Flag Rotation

Similar to controlling the rotation direction of the driving DC motors, the DC motor powering the indicator flag is also connected using a H-Bridge circuit structure.

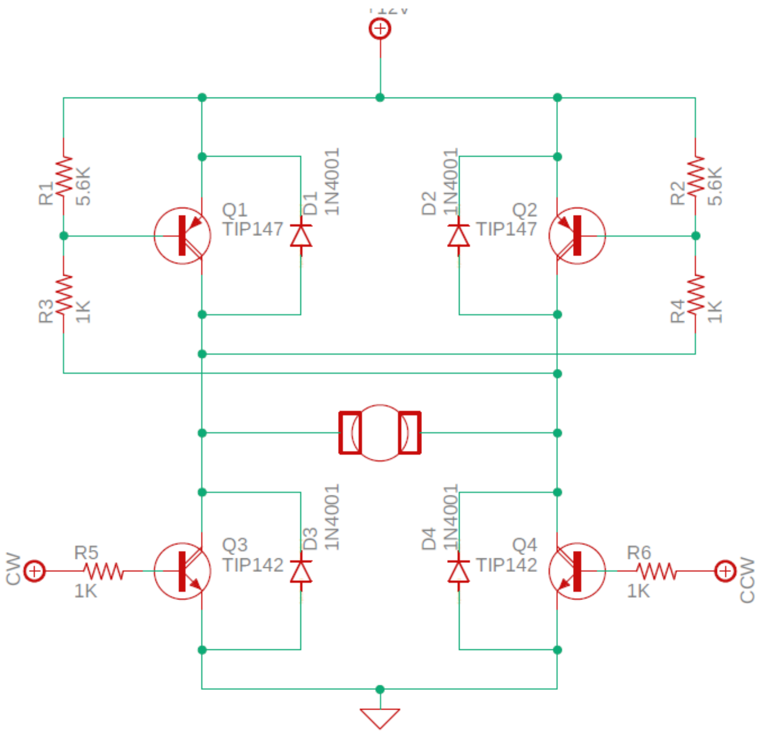


Figure 33. Circuit Diagram of Modified H-Bridge

However, after soldering the basic H-Bridge layout as shown in Section 7.1 of this report, a major issue that arises from that configuration of the circuit is that the transistors tend to heat up quite a bit when being powered. After carefully consulting Section 6.2.2.5 from the AER201 textbook, a modified version of the basic H-Bridge circuit was completed as shown in Figure 33.

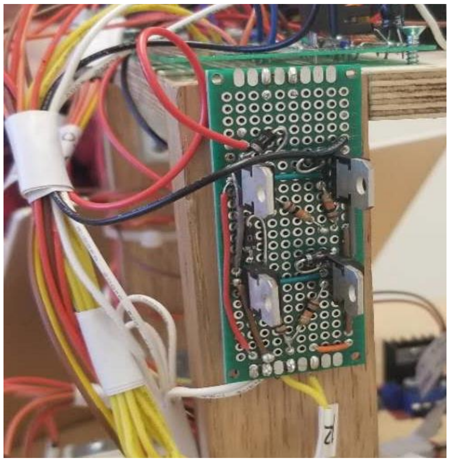


Figure 34. Soldered Modified H-Bridge Circuit

Since DC motors can draw a large amount of current when operating, the transistors get heated due to the current moving through it, which leads to power dissipation. In this version of the H-Bridge circuit, an additional six resistors are added to the base of the transistors to prevent high amounts of current flowing into the circuit. On top of that, diodes are added between the collector and emitter of the transistors to prevent reserve biasing of the current. Overall, the modified version of the H-Bridge circuit adds extra layers of protection to levitate the deterioration of components and eliminate possibilities of short circuiting.

7.4 Power Management

Table 6: Power Consumption of Electrical Components in the Robot

Component	Voltage Supplied (V)	Current Drawn (mA)	# of Unit	Total Current (mA)
DC Motor	10.8 - 12	250	4	1000
MG996R Servo	5	500	1	500
SM-S4306R Servo	5	100	1	100
IR Sensor	5	30	6	180
Rotary Encoder	5	5	1	5
PIC	10.8 - 12	500	1	500

Based on the calculations done in the table above, it can be concluded that the ideal power supply for the robot would be generating 10.8 - 12 V in terms of voltage and at least 2200 mAh in capacity to avoid constant charging. Therefore, we chose to use the Amazon Basics High Capacity Rechargeable Batteries as our source of power. Each cell of these batteries can generate 1.2 V of voltage and store 2400 mAh of power. With 6 of these cells connected in series, the total voltage of the power supply can reach 10.8 V while still maintaining a 2400 mAh capacity.

Figure 35. A Cell of Amazon Basics High Capacity Rechargeable Battery

With this power supply, both the driving DC motors and the PIC microcontroller board are powered directly without any regulations. This is due to the fact that we need as much power as possible on the driving DC motors to pull the weight of the entire robot to go forward. As for the PIC, it has an embedded voltage regulator on it so it can be self-regulated. For the rest of the components, the batteries are regulated down to 5V to satisfy the power ratings of the components. The components are regulated by two LM338 and one LM7805 voltage regulators, and they are evenly distributed based on current consumption. This distribution reduces the amount of power dissipation that could occur while current travels through the voltage regulators. However, heat sinks were installed on the regulators as well just in case heat is still produced in the process.

8. Microcontroller

Microcontroller takes charge in processing and sending signals from and to circuits, instructing the robot to handle external situations and communicating information to the user. It controls the robot through invisible software computation.

A PIC microcontroller from Microchip Technology Inc. and a customized development board (produced by AER201) are used as the main foundation in the software design. The advantages of the PIC microcontroller, as recommended by the instructor and client, are its fast operation, low power, low cost and ease of programming (Appendix D).

The required functions can be divided into the following categories:

User Interface: LCD and Keypad Interrupt and interface design for retrieving data and starting an operation

Real-Time Clock: Independent real time clock on the robot

IR sensor signal reading and processing

Encoder signal reading and processing

Permanent memory of operation result (read and write)

Servo motors control for cone deployment gates

Speed and overall direction control of Mecanum wheels

Main Function

Global variables

PC interface: allow operation data to be readily download and display on a computer

Line following

Indicator Flag DC motor control

The following paragraphs explain how we approached these functions required. See Appendix B for codes and detailed pseudo code (as the comments of codes) of all the functions.

8.1 User Interface:

A 4 by 4 Keypad is installed on the development board to allow users to input various commands. Keypad input is connected to RB4-RB7 of the I/O pins. The four bits have 16 (2^4) combinations, corresponding to the 16 buttons on the keypad. Pins RB4-RB7 have the function of "interrupt on change". Once any of the four bits changes (keypad pressed), RB1 (interrupt 1) will set its flag to 1 and hence signals the processor to handle the interrupt (if interrupt 1 is enabled). Reading the value of RB4-RB7 in the interrupt handler, one could tell which command is issued hence change the corresponding global variable(s) and re-directs the main function.

A Character LCD is used to display user instructions and history operation results.

For parallel pages, the display rotates automatically to reduce the steps a user may otherwise take for skimming. At the same time, information is broken down to three hierarchical levels to reduce the period of rotating in each level.

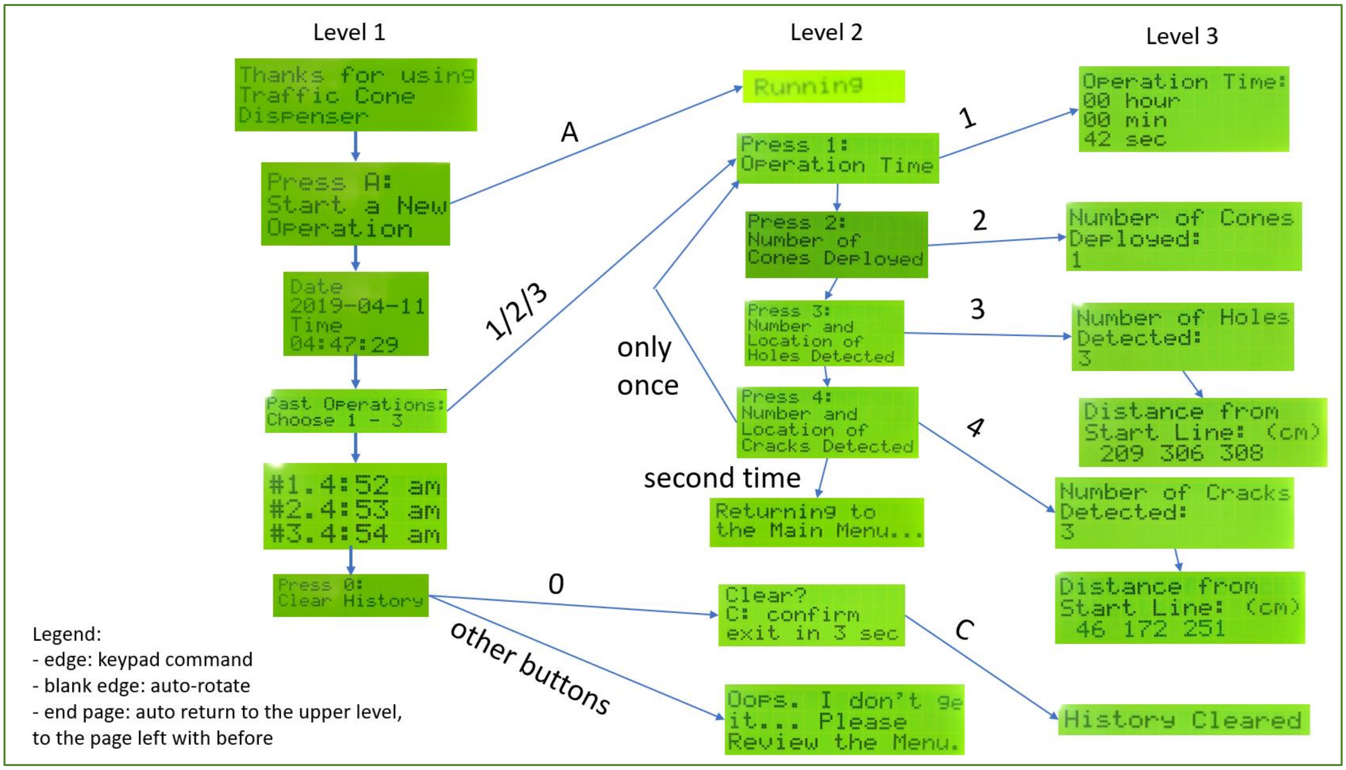


Figure 36. Flowchart of User Interface.

Some extra features are achieved in combination with permanent data function.

1. Users can clear the history data through keypad buttons without going to back-end organization, which is error-prone.

2. Past operations can be easily identified by its starting time.

3. The length of operation list is dynamic and can change based on memory, instead of displaying some blank slots when unnecessary.

8.2 Real-Time Clock:

DS1307 64 x 8, Serial, I2C Real-Time Clock is installed onto the board. It transmits clock data to the main PIC through I2C pins and can be read by programs. Once initial time is reset to the local time, the clock will be synced if an independent lithium coin battery is connected to the RTC. In this robot, RTC is read both in standby display and throughout its operation to find the operation duration. To ensure the time is not overrode repeatedly, the programmer needs to call the reset function only once and call it outside of main function.

8.3 IR Sensor Signal Reading and Processing

The five IR sensors used take up 5 independent input pins. As a digital input, a “high” (1) indicates that it sees a black color while a “low” (0) indicates it sees a lighter color. This is used to identify the black hockey tapes that simulate the lanes and obstacles. The left two sensor signals are processed together to correct robot position when moving forward and following the lane. The rest three sensors, based on their physical position on the robot, tell the programs shape and horizontal location of the obstacles.

Once the obstacle is identified, the program records it on a detection list, and compares it with historical data to decide if cone(s) need to be deployed at this location. It has a separate list indicating the deployment information.

8.4 Encoder Signal Reading and Processing

Only one encoder is needed in the robot to indicate its overall motion. We calibrate the parameters to relate encoder reading and the distance it travels sideways and forward (or backward) respectively. According to the datasheet of the encoder [], the encoder output produces 334 pulses every cycle rotated. To avoid missing pulses, we use an external interrupt to read the encoder signal, instead of polling it at the main function which contains other operations. Encoder output is read every iteration in the main robot operation loop. Temporary cycle counters and distance counters are separate. Number of interrupts is accumulated into temporary cycle counters. Temporary cycle counters are accumulated into the corresponding distance counter (horizontal or straight) and then being cleared.

8.5 Permanent Memory of Operation Data (Read and Write)

Permanent memory is stored using “data EEPROM memory” of the PIC. It has the capacity of 1024 bytes in total and each address/byte has 8 bits capacity. Table 7 shows the storage of operation data. The first two bytes store the pointer to the next writing address. They are updated every time memory writing is finished. They are also used to calculate the number of operations since last reset of memory. Each operation uses a 56-byte block to store all its data (as indicated by the green block). Once an operation is completed, a function writes the current data into EEPROM byte by byte and at the end adds the write_pointer (bytes 0 and 1) by 56. Note that each distance value is stored by 2 bytes because the largest possible value is around 400(cm) however 1 byte can only represent 255 at largest.

Table 7: Data EEPROM Memory Organization.

0	ADDRH	33	NUMBER OF CRACKS DETECTED
1	ADDR	34	DISTANCES OF THE CRACKS FROM THE START LINE (MAX 12 HOLES), EACH DISTANCE TAKES UP 2 BYTES TO STORE. DEFAULT = 0. 1ST BYTE = INT(DISTANCE/256). 2ND BYTE = INT(DISTANCE%256).
2	START_TIME_HH(24)	35	
3	START_TIME_MM	36	
4	OPERATION_TIME_(DURATION)_HOURS	37	
5	OPERATION_TIME_(DURATION)_MINUTES	38	
6	OPERATION_TIME_(DURATION)_SECONDS	39	
7	NUMBER OF CONES DEPLOYED	40	
8	NUMBER OF HOLES DETECTED	41	
9	DISTANCES OF THE HOLES FROM THE START LINE (MAX 12 HOLES), EACH DISTANCE TAKES UP 2 BYTES TO STORE. DEFAULT = 0. 1ST BYTE = INT(DISTANCE/256). 2ND BYTE = INT(DISTANCE%256).	42	
10		43	
11		44	
12		45	
13		46	
14		47	
15		48	
16		49	
17		50	
18		51	
19		52	
20		53	
21		54	
22		55	
23		56	
24		57	
25		…	(NEW OPERATION DATA INSERTED TO THE END OF THE EXISTING DATA)
26		…	
27		1023	
28			
29			
30			
31			
32			

A “one_byte_reader” function is also written to be called when displaying the data.

8.6 Servo Motor Control for Cone Deployment Gates

The two servo motors used in the final design are MG996R and SM-4306R. MG996R controls the lower gate (wood plate) which rotates ~90 degrees when instructed to open or close. The angular position is directly controlled by the duty cycle of the input. The second servo motor rotates the pinion and hence slides in and out the second gate [pic of mech]. SM-4306R uses duty cycles to control the orientation and speed of rotation. When duty cycle is around 1.5ms/20ms (middle point), the motor stops. When duty cycle is less than that, the motor rotates clockwise and vice versa. Speed is directly proportional to the difference between the set duty cycle and the middle point. Outputs to the servo motors are set to high and low alternatingly with different time lags to simulate the desired duty cycle. Looping time is calibrated for the second motor to control its angular displacement and hence the distance the second gate slides in and out.

8.7 Mecanum Wheels (DC Motors) Speed and Overall Direction Control

The speed of DC motors is controlled by duty cycles. As mentioned in section 7.1, direction of the robot is regulated by slowing down one side of the robot.

8.8 Main Function

The main function integrates all the subfunctions to call them when necessary. The operation routine is organized by the main function.

<u>1. The Configuration of I/O Ports.</u>

Pins are assigned as following:

Table 8: Pin Assignment on PIC Ports

Pin name	Input /Output	Description
RD0:RD1	Out	Indicator Flag DC Motor
RD2:RD7	Out	LCD Display
RB4:RB7	In	Keypad Input
RB1	In	Interrupt 1, Keypad Interrupt on Change Bit
RC3:RC4	In	RTC
RB0	In	Interrupt 0, Encoder of wheels
RB2	In	Left Detection Sensor
RB3	In	Middle Detection Sensor
RA1	In	Right Detection Sensor
RA3	In	Cone Number Sensor
RA4	Out	Sliding Door Servo Motor
RA5	Out	Rotating Door Servo Motor
RE0	In	Left Line Following Sensor
RE1	In	Right Line Following Sensor
RC1:RC2	Out	Back Left Wheel Motor
RC5:RC6	Out	Front Left and Back Right Wheel Motors
RC7, RC0	Out	Front Right Wheel Motor

All ports are configured to be digital. In general, one DC motor needs 2 pins to control 3 types of motions:

1) clockwise, 2) anticlockwise, 3) stop.

2. Regular Counter Updater

This function updates operation time counter by reading from RTC. To make sure the robot returns to the start line before time's up (3 minutes). As tested, it usually takes 1.5 minutes to complete the operation, which is far from the limit. Therefore, the precision of the counter can be lowered to save cost: the updater is only called per several iterations in the main loop to save time cost.

Originally, the distance counter is also updated inside the regular updater. But the precision required for distance is much higher (should be as high as possible), so distance updater is moved to the outside and called every iteration of the main loop.

3. Completion Return

Indicator flag returns to "None" (flat) position. -> robot slide to the right until the entire robot is outside the lane -> robot moves backward until it's entirely behind the start line -> robot slide to the left until it arrives the standby position -> close the sliding door if it's open before -> read the final operation time -> write operation data into permanent memory -> toggle the "completion" Boolean variable and initiate standby display page counter.

4. Restart and Initialization

Restart time and distance counter, initialize record variables, initialize transition variables, enable encoder interrupt, communicate with the user about the state (start running), and toggle the "completion" Boolean variable to re-directs the branch to go to in the following main loop iteration.

8.9 Global Variables

Over 45 global variables are used per operation, as listed in Appendix B. They can be categorized to RTC variables, transition variables, record variables (see Definitions), LCD display state variables, constants and parameters, high-level program state switches, and EEPROM variables. Figure 37 shows a snapshot of the global variable list.

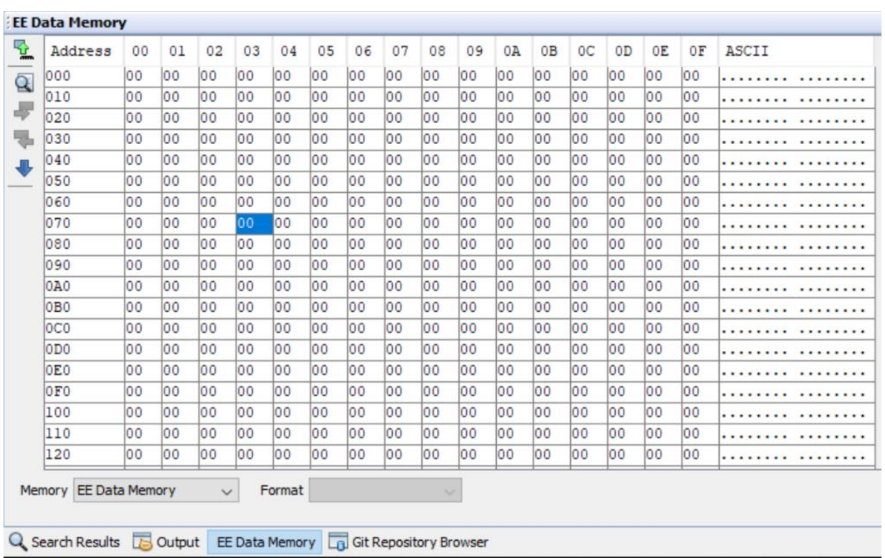
```
unsigned char time[7] = {//declare the data type of real-time array
    0x45, // 45 Seconds
    0x59, // 59 Minutes
    0x23, // 24 hour mode, set to 23:00
    0x00, // Sunday
    0x31, // 31st
    0x12, // December
    0x18  // 2018
};

//"transition variables" that will be computed and then used as inputs in the robot main operating rout
float rotary_counter=0;//instant number of ticks traveled
float turns_counter=0;// instant number of cycles traveled
float rotary_accum = 0;// total number of ticks traveled in the straight line
float accum_straight_distance=0;// total number of cycles traveled in the straight line
long start_sec=0;// real start time converted to second form
long end_sec=0;// real current time converted to second form
long operation_sec=0;// time difference between start_sec and end_sec in the unit of seconds
unsigned int a = 0;//left detection sensor reading
unsigned int b = 0;//middle detection sensor reading
unsigned int c = 0;//right detection sensor reading
float last_problem_bool[2] = {0,0};//last dispense information: {'crack' -> 0 or 'hole' -> 1, distance
bool completion_bool = true;// switch of standby or running
int last_sensed = 0; //1-crack, 2-hole, 0-none identified yet
float drop_position[12] = {0,0,0,0,0,0,0,0,0,0,0,0};//a queue of next positions that the robot supposes
```

Figure 37.

8.10 PC Interface

This feature allows history operation data to be readily downloaded and displayed on a computer. As MPLAB X IDE can view EEPROM memory of the microcontroller connected to it, the user can open this window and copy the data into an extra Python program we wrote to interpret the hexadecimal numbers stored as normal language. (see Appendix B for this program code). Figure 39 shows an example of the output of the Python program.


EE Data Memory

Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	ASCII
000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....

Figure 38.

```
Permanent Memory used 16.60 percent
 170
At 04:30am
Operated 0 hours,1 minutes,24 seconds
Deployed 8 cones
Detected 3 holes
Their distances from Start Line(cm):
 299 325 343
Detected 7 cracks
Their distances from Start Line(cm):
 35 105 150 209 301 334 341
At 04:41am
Operated 0 hours,1 minutes,21 seconds
Deployed 8 cones
Detected 4 holes
Their distances from Start Line(cm):
 112 190 311 338
Detected 7 cracks
Their distances from Start Line(cm):
 38 110 158 219 314 348 355
At 04:43am
Operated 0 hours,0 minutes,42 seconds
Deployed 1 cones
Detected 6 holes
Their distances from Start Line(cm):
 94 156 186 218 309 335
Detected 5 cracks
Their distances from Start Line(cm):
 34 105 217 345 352
```

Figure 39.

8.11 Line Following

The two leftmost sensors at the front separate with each other by the width of the left border of the lane. Going straight, the robot will read “high” for both sensor inputs. However, a difference in the two sensor readings indicates the deviation of (the front of) the robot from the correct track. When moving forward, the program constantly polls on the reading and adjusts the direction of movement.

A previous version of design checks the movement much less frequently (1 check per 10 cm). Through testing, we found that the robot will deviate too much during the time lag and can’t go back to the correct track smoothly. So, we then decided to integrate checking and sensor feedback all the time with straight movement. In addition, a previous method of adjusting is to slide the robot horizontally. But with the rapid checking, a less abrupt turning will be enough for smooth following, not to mention that sideway motions take more current from the circuit.

8.12 Indicator Flag DC Motor Control

Indicator Flag has three different states: hole, crack, flat. Flat occurs at the beginning of an operation until any obstacle is detected. Besides, the flag is turning flat when operation is complete. With the physical shape of the flag, we should send signal to allow the motor to rotate 90 degrees and 180 degrees. This is achieved by calibrating the time to rotate. When the motor needs to rotate in an opposite direction, the two signals sent to it swap.

8.13 Dispense Routine Function

One function called “drop_bool_function” reads the current obstacle and compare with deployment information of last obstacle to return if cone(s) should be deployed on this obstacle. If cone(s) should be deployed at this location, the obstacle’s identity and location will be recorded in an array specifically for deployment data. Since sensors and in the main function, deployment queue is read in every iteration so that when the deployment location is reached, the robot can stop and execute the deployment.

8.14 Future Improvement for Microcontroller

A few colleagues were invited to help us test on the user interface. The overall clarity is appreciated. The key feedback for improvement is on controllability. As we design the LCD to rotate automatically, people who want more time to read each page may find it troublesome waiting the next iteration to read the page again and would hope to have the control of when to flip pages. In the future, if possible, the engineers can potentially design a new data structure that allows more controllable interface.

8.15 Simulation Results and Takeaways

Figure 40 shows a test of RTC display. As seen from the picture, it didn’t work properly. After that we read the datasheet of the RTC (Fig 41), and found out the right way to read time from its registers. At the end of testing, the RTC works well(Fig 42).

Figure 40.

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	FUNCTION	RANGE
00h	CH	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12	10 Hour	10 Hour	Hours				Hours	1–12 +AM/PM
		24	PM/ AM							00–23
03h	0	0	0	0	0	DAY			Day	01–07
04h	0	0	10 Date		Date				Date	01–31
05h	0	0	0	10 Month	Month				Month	01–12
06h	10 Year				Year				Year	00–99
07h	OUT	0	0	SQWE	0	0	RS1	RS0	Control	—
08h–3Fh									RAM 56 x 8	00h–FFh

0 = Always reads back as 0.

Figure 41.

Figure 42.

The encoder is tested when connecting to RB0 interrupt port. For every 10 ticks rotated manually, the data is recorded. As can be seen from Fig 43, uncertainty is around 2 ticks. This is within our tolerance.

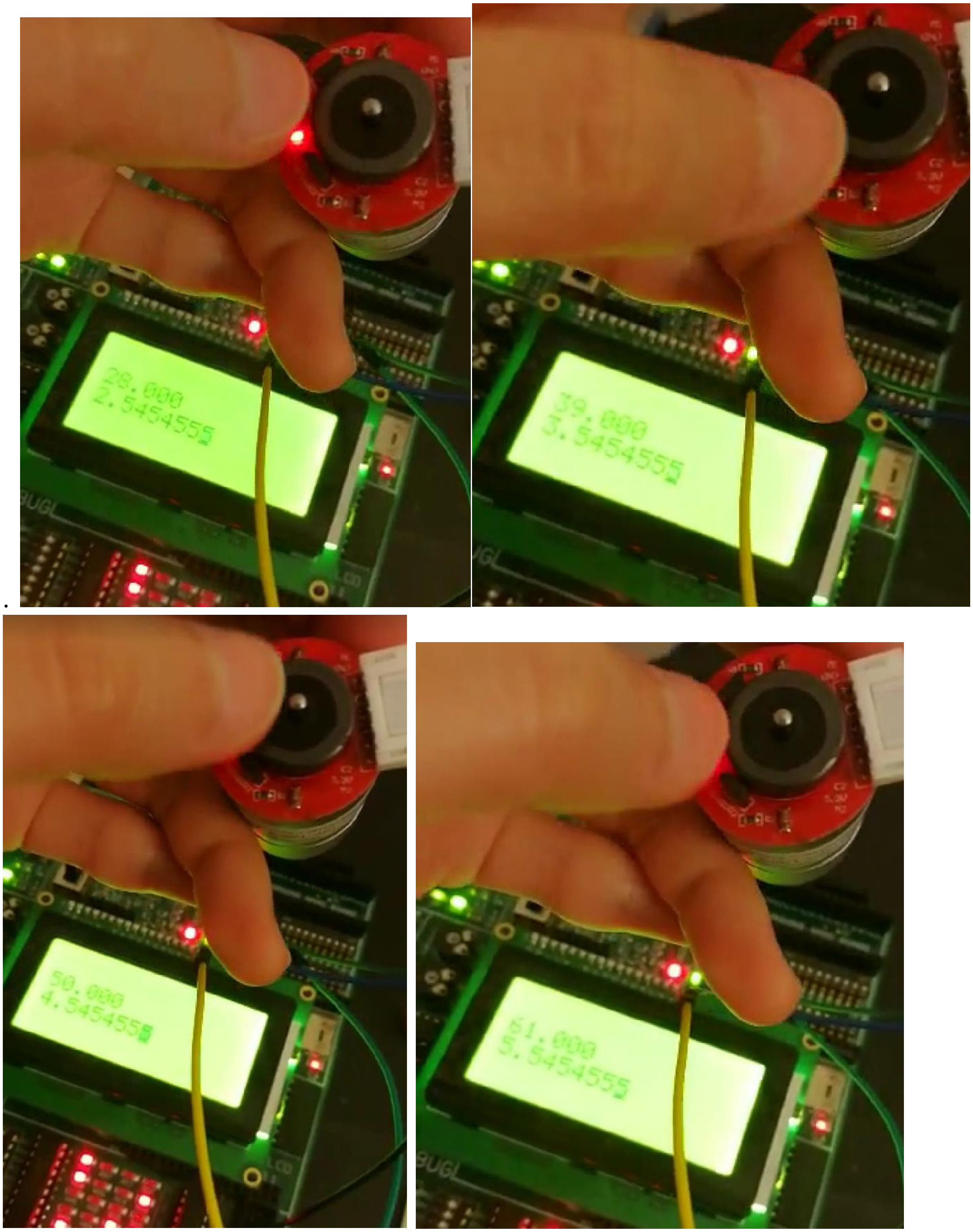


Figure 43.

9. Integration

The first eight weeks of this course focus on completion of subsystem: electromechanical, circuits and microcontroller. The following weeks focus on the integration of systems for the machine to function automatedly.

The overall integration could be divided into two phases. During the first phase, most of the integration is done through physical connection of actuators, circuits and microcontroller. During the second phase, the construction and integration are mostly completed and finalized, the focus in this stage is the calibration of the machine through programming, and also some adjustments on the actuators or arrangement of the robot.

9.1 Phase 1: Physical Integration/ Functional Calibration

- Microcontroller tested the code with actuators such as DC motors, motor with encoders, servo motors to check the response to signal from PIC. Circuit member tested on the protoboard to check if the arrangement of the circuit elements is correct.
- All connections between electrical components and the power supply are securely soldered together for stronger physical and electrical connections.
- Designing specific layout and installing PIC, PCB board, battery, motor driver board and emergency stop on the robot
- Arranging and Securing the wires and actuator

9.2 Phase 2: Accuracy Calibration

During the two phases of integration, specifically during the calibration in the second phase, several problems arose, thus some adjustments are made to the design as well as the constructions:

- Reduce the length of the 90-degree aluminum pieces at the back of the robot such that the cone can move freely below the cone holder without touching it.

Figure 44. New Appearance of the Cone Holder After Cutting the Back Pillars

- Installment of foam around the side of the cone holder to ensure the cones are in position without any rotations.

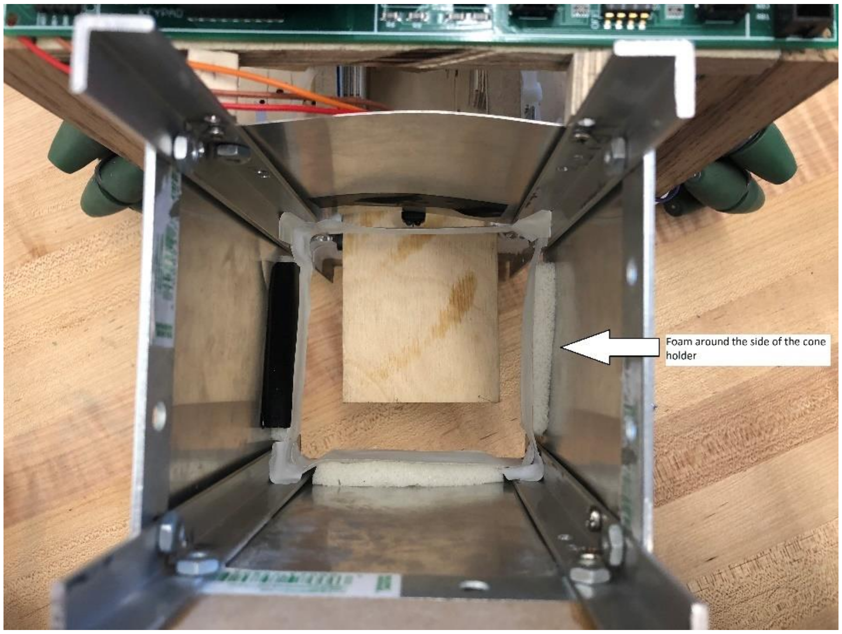


Figure 45. Foam Installed Around the Cone Holder to Hold the Cones in Place

- Replace the detection sensors/ line following sensor used for detecting Hole/ Crack with IR sensors with greater range due to the fact that lighting and external environment affect detection on SF floor.
- Using zip ties to ensure the DC motors attached to the wheels are aligned straightly thereby reducing the vibration while moving

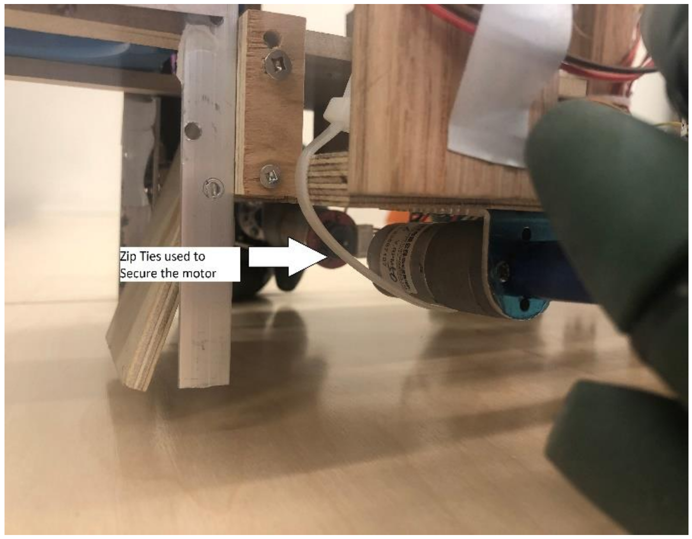


Figure 46. Zip Ties Added for Securing the Motor

- Sanding the edge of the sliding door for it to easily slide in between two stacked cones.

Figure 47. Illustration of How Sliding Door Hold Up the Upper Cone

- Adjusted the angle of the rotating door for more accurate drop (modified to 85 degrees instead of 90 degrees)
- While calibrating the parameters of encoder, servo, flag rotating time, instead of using perimeter since it is hard to measure. Instead, use cycles directly, and measure critical

distance to keep the parameters. In addition, keep a record of the testing results to squeeze and get precise parameter

9.3 Future Improvements and Suggestions

1. Use keypad to adjust a small parameter, instead of changing it on PC and reload everything. -> boost efficiency.
2. Protection for exposed circuits and wires
3. Risk management on some components. For example, adding another row of sensor for hole/crack detection to reduce uncertainties

*Note: detailed future improvement is listed in each subsystem section above

4. Potential to extend to larger capacitance. For example, cone holder can be extended higher to hold more cones once; EEPROM can be modified to store more information than 4 operations; width/length of the lane for the robot to operate can be extended.

10. Time Management

Initial and Accomplished Schedule (Gantt Charts)

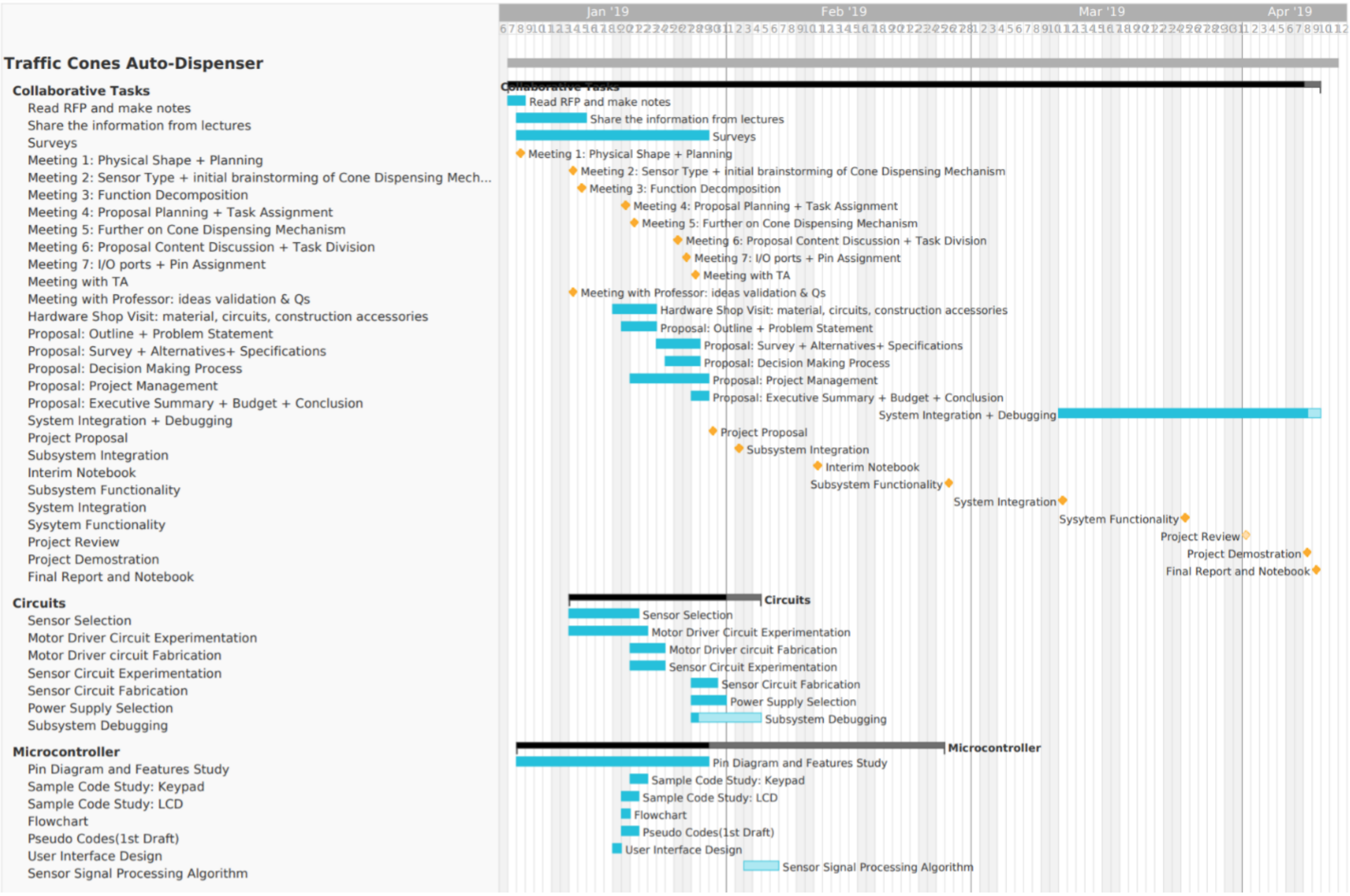


Figure. 48-1

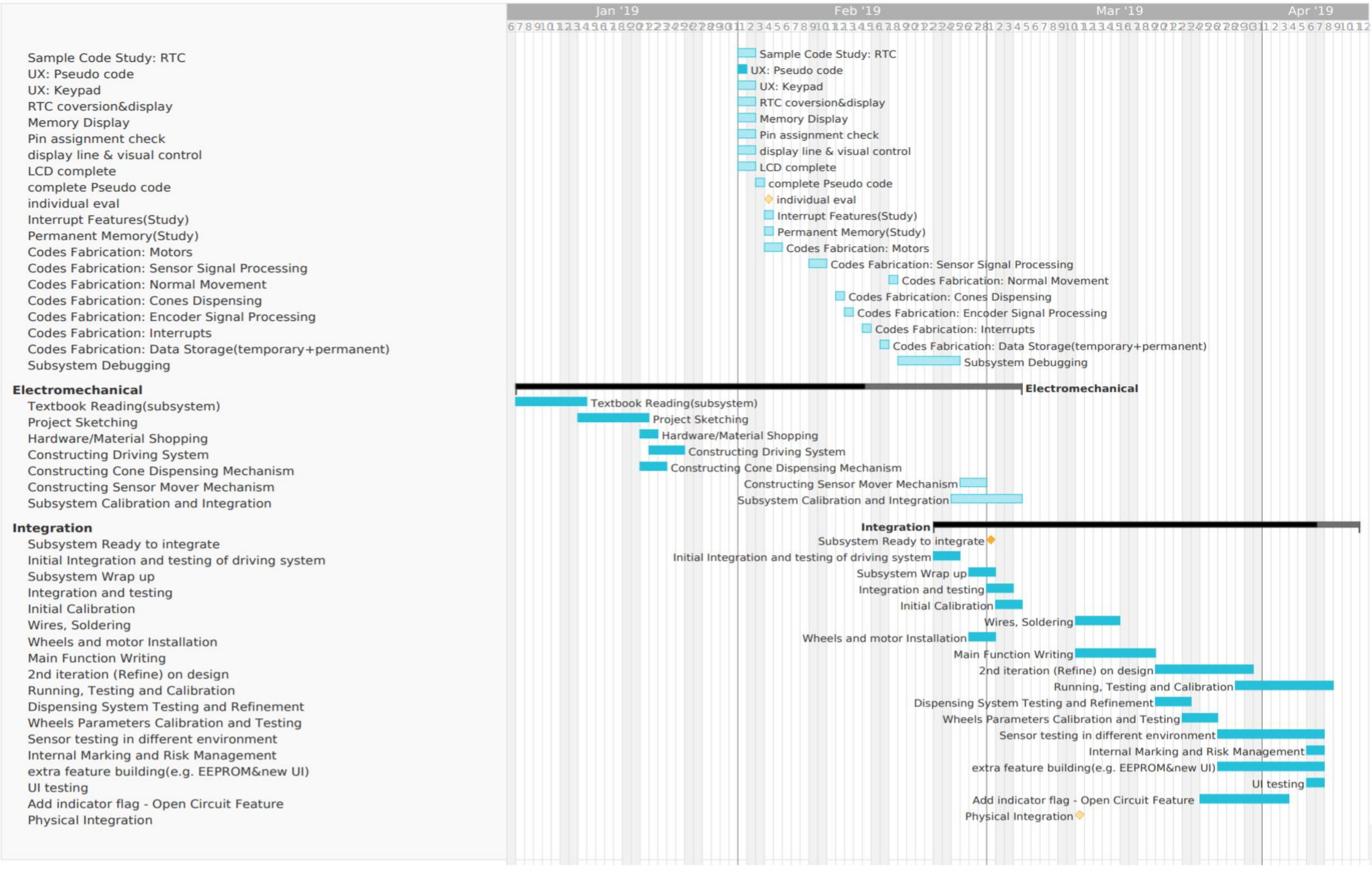


Figure. 48-2

Figure. 48-3

In the Gannt Chart above, pre-set milestones/goals are labelled as orange points while blue blocks represent detailed tasks that have done, and the width represents their duration.

From the chart, we noticed that a few goals are not met on time. For example, wiring and soldering was finished later than Physical Integration milestone. In addition, not all risks are addressed properly before the accuracy debugging deadline, which influenced the result in final competition. Apart from that, the team generally maintained a good progress and everyone in the team was on the same page, with a clear vision on the priorities at various stages of the project. This enabled the team to finish functional integration ahead of time. For future improvement, risk management should be a critical point the team need to pay attention to before the presentation/demonstration. For example, the stability of the robot performance influences the results a lot. The team may consider strategies like redundancy in the future to make sure the robot works.

11. Conclusion

In conclusion, the robot, named after Mr. Krabs, can complete a detection and deployment operation over 4-meter lane within 1.5 minutes. Its omni directional wheels allow it to drive stably and flexibly on a variety of road conditions. Permanent data memory of its microcontroller also extends the functionality and sustainability of the robot. To increase the usability, user interface (keypad and display content) is carefully designed so that it is accessible for a wide range of users. In addition, the permanent logs can be easily downloaded to a personal computer and automatically interpreted from computer languages to human languages.

To improve the performance of the robot, future engineers can work on detection and deployment accuracy. The current concern is that the type of sensors used, in combination with their physical arrangement, vary readings readily when external lighting conditions change. Secondly, current deployment method relies on gravity of the cones themselves. Extra cares should be taken to ensure the cones to be deployed at a precise position. If new mechanism is to be implemented, existing cone holder can also be readily dissembled and adapted to a potential new design.

The project is a valuable experience for student engineers to gain hands-on experience as it stretches their electromechanical, circuitry and programming expertise, as well as knowledge of driving systems, sensing systems and software data processing. Furthermore, the simple but effective conceptual design can be applied to many industry cases. Therefore, perspective engineers are encouraged to take on this opportunity and keep working on the project.

12. Final Design

12.1 Description of Overall Machine

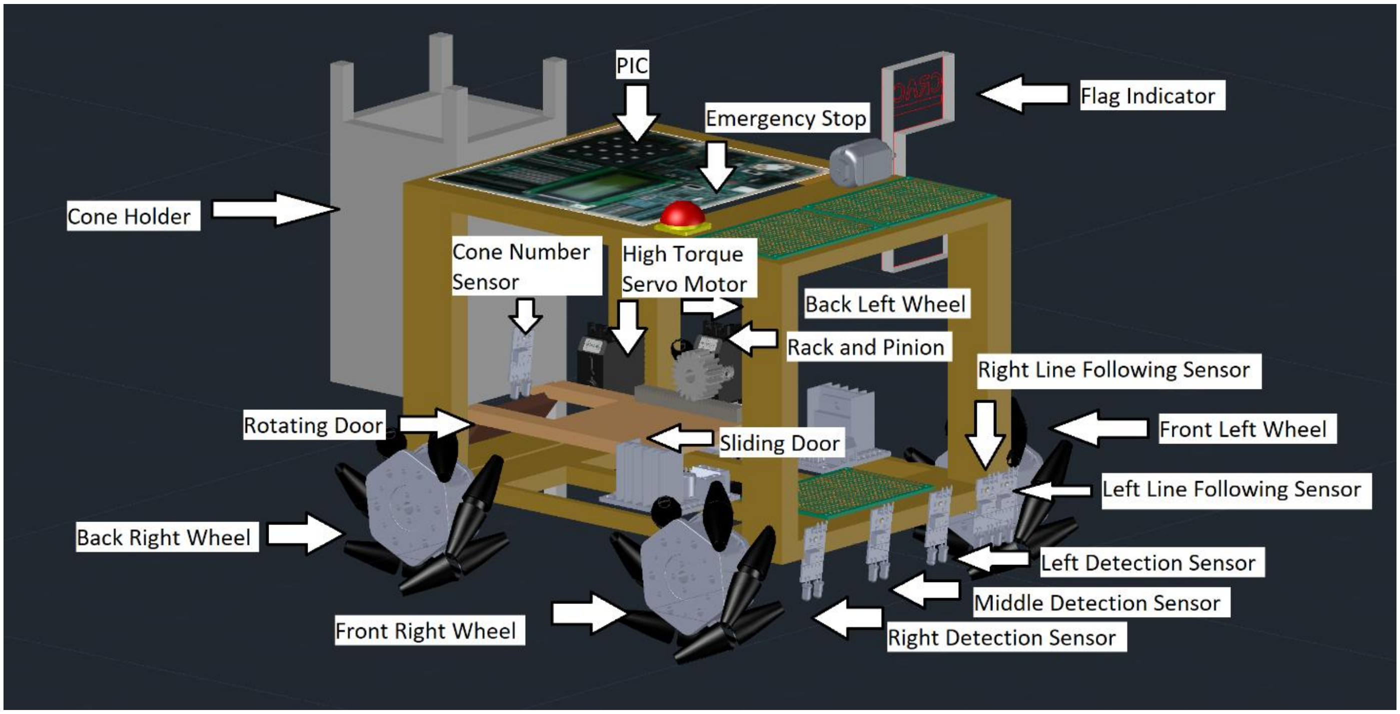


Figure 49. 3D Model Illustration of the Integrated Robot

As shown in Figure 49. The overall design consists of five systems: sensing system, driving system, cone dispensing system, indicator system and user interface.

The Driving system is the omni directional driving system consists of four Mecanum wheels, which allows the robot to move sideways to deploy two cones on the crack consecutively. Each wheel is connected to a DC motor controlled by signal from PIC. One with encoder to record the distance.

Cone dispensing system consists of three components: sliding door controlled by rack and pinion; rotating door controlled by high torque servo motor and an aluminum cone holder. While operating, the rotating door will initially position at 180 degrees to hold the cone in the holder; then the sliding door slides out through the rotation of gears, that lets all cones sitting on the rotating door; it then goes in to hold up the upper cones except the last one; Then the rotating door open at a degrees of 85 to drop one cone.

In the sensing system, five IR sensors are positioned at the front of the robot with specific distances (5cm RDS -7.5cm MDS -7.5cm LDS) for detecting crack or hole on the lane. The middle sensor is for detecting hole while the left and right ones are for cracks. The adjacent two sensors at the left are for lane following.

The Indicator of our robot is a rotating two-sided flag with hollow word “crack” and “hole” on each end, which is controlled by DC motor. H-Bridge circuit allows it to rotate to hold the right “word” up based on the obstacle (hole or crack) detected.

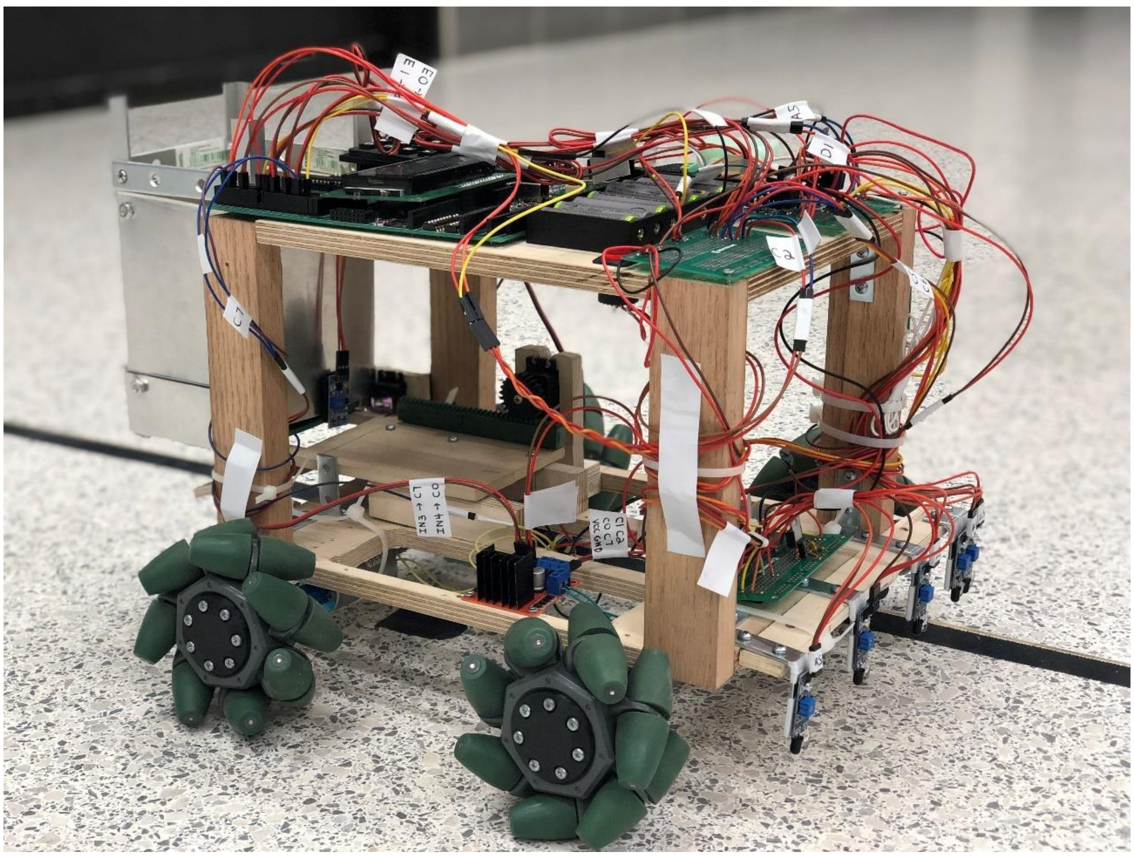


Figure 50. Physical Appearance of the Robot on the Lane

12.2 Standard Operating Procedure

The robot will be placed on the lane with the two line-following sensors facing the black areas. When power is given, the robot is in the standby mode. A user can choose to:

1. start a new operation,
2. read history operation data, or
3. clear history data

through keypad commands (see section 8.1 for details). Assume this is the first operation since last time the history is cleared, the user will now want to run a new operation. The user is responsible for loading as many cones as they desire, ideally within 12.

When 'A' is pressed on the keypad, the robot moves forward following the lane. At the same time, it keeps an eye on the road obstacles (holes or cracks mimicked by black hockey tapes of different shape). When an obstacle is sensed, the robot shows the identity of the obstacle(hole/crack) through a wood indicator flag. After that, it keeps going forward and detecting, but remembering the deployment task if the obstacle needs one (see section 3.2 for the criteria). Besides regular moving and sensing, if the cone holder reaches the deployment location, it stops and executes the deployment. It deploys one cone directly onto a hole and moves left and right to deploy two cones on a crack based on the instruction (see section 6.4 Table 5 for details). If cones are used up, the robot will still complete detection over the 4-meter lane but will not execute deployment routine. This allows the robot to use the most out of its functionality and gives the user most information. At the same time, it saves a lot of time to skip the attempt to move sideways and deploy cones when no cones are left.

When the robot reaches the destination, it slides to the right of the lane then drives backwards until the entire robot is behind the start line. It also slides to the left at the end to return to the standby position. Both the deployment sliding door and the indicator flag return to initial position.

When the robot is in standby mode, the user can do the next operation as instructed by LCD display.

13. Appendix

Appendix A: References

[1] "INTERSTATE RUBBER PROD. CORP. v. RADIATOR SPECIALTY CO". *United States Court of Appeals, Fourth Circuit. 214 F.2d 546 (1954)*. Retrieved 11 April 2019.

[2] "The Surprising History of the Traffic Cone: Spring is here, Chicago!", *Brancato Snow & Ice Management*, 2018. [Online]. Available: https://www.brancatosnowremoval.com/the-surprising-history-of-the-traffic-cone-spring-is-here-chicago/. [Accessed: 11- Apr- 2019].

[3] *Inderscienceonline.com*, 2019. [Online]. Available: https://www.inderscienceonline.com/doi/abs/10.1504/IJHVS.2004.004038. [Accessed: 30- Jan-2019].

[4] S. ARIFFIN, *AN AUTOMATIC TRAFFIC CONES DISPENSER AND COLLECTOR SYSTEM*. 2014.

[5] "AER201 - Historical Projects", *Aer201.aerospace.utoronto.ca*, 2019. [Online]. Available: http://aer201.aerospace.utoronto.ca/History/team.aspx?ID=1&Team=1&Project=Project1&Year=2011. [Accessed: 30- Jan- 2019].

[6] E. Luoma, "Highway cone dispenser and collector", US5054648A, 1990.

[7] Larguier, F. (1996). *DEVICE FOR THE PIACEMENT AND IF DESIRED THE COECTION OF TRAFFIC CONES*. 5,525,021.

[8] Royal Truck & Equipment. (2017). *Cone Placement & Cone Retrieval System - Royal Truck & Equipment*. [online] Available at: https://royaltruckandequipment.com/cone-placement-retrieval-system/ [Accessed 30 Jan. 2019].

[9] R. Emami, *Request for Proposal#1: The Traffic Cone Deployment Machine*. 2019.

Appendix B: Complete Code for Microcontroller

1. <prototypes.h>

```
/*
 * File:   prototypes.h
 * Author: Chen
 *
 * Created on February 25, 2019, 2:04 AM
 */

#ifndef PROTOTYPES_H
#definePROTOTYPES_H

#ifdef  __cplusplus
extern "C" {
#endif

#ifdef  __cplusplus
}
#endif

//functions

void configureports(void);
void readRTC(void);
void rtc_set_time(void);
```

```
unsigned int digit0(unsigned int f);
unsigned char nextLine(unsigned char g);
void printintarray2(unsigned int* a,int b);
void ot(unsigned int* print_data);
void Cones(unsigned int* print_data);
void Holes2(unsigned int* a,int b);
void Cracks(unsigned int* a,int b);
void sensingH(void);
void sensingC(void);
void page1(void);
void page2(void);
void page3(void);
void page4(void);
void page5(void);
void page6(void);
void data1(void);
void data2(void);
void data3(void);
void data4(void);
void initialize_func(void);
void initmoving_disp(void);
void high_priority interrupt interruptHandler(void);
void read_encoder(void);
int distinguish_H_C_function(void);
int hole_drop_bool_function(void);
int crack_drop_bool_function(void);
```

```
void hole_dispense_function(void);
void crack_dispense_function(void);
void sensed_function_3(void);
void normal_updater(void);

void standby_rotating(void);
void middle_crack_drop(void);
void left_crack_drop(void);
void right_crack_drop(void);
void record(char corh);

void L_I(void);
void R_I(void);
void backw(void);

void Stop(void);
void straight(void);
void turn_left(void);
void turn_right(void);

void moving(void);
void move_to_hole(void);

void drop_record(int a);
void drop_function(void);
void completion_return(void);
```

```
void clear_mem(void);
void update_pointer(void);
void completion_write(void);
void read_pointer(void);
void pntr_head_read(void);
void subtract(void);
char one_byte_reader(char r, char rh);

void data_disp(void);
void clear_select(void);
void clear_finish(void);
void data_select(void);
int read_total(void);
#endif  /* PROTOTYPES_H */
```

2. <global_variable.h>

```
/*
 * File:   global_variable.h
 * Author: Chen
 *
 * Created on February 24, 2019, 9:51 PM
 */
#include <xc.h>
#include <stdio.h>
#include <stdbool.h>
```

```
#ifndef GLOBAL_VARIABLE_H
#defineGLOBAL_VARIABLE_H

#ifdef  __cplusplus
extern "C" {
#endif
#ifdef  __cplusplus
}
#endif

//RTC variables/////
unsigned char happynewyear[7] = {//current local time, for initializing RTC
   0x20, // 20 Seconds
   0x20, // 20 Minutes
   0x20, // 24 hour mode, set to 20:00
   0x03, // Wednesday
   0x03, // 03st
   0x19,  // 2019
   0x04, // April
};
unsigned char time[7] = {//declare the data type of real-time array
   0x45, // 45 Seconds
   0x59, // 59 Minutes
   0x23, // 24 hour mode, set to 23:00
   0x00, // Sunday
   0x31, // 31st
```

```
    0x12, // December
    0x18  // 2018
};

//"transition variables" that will be computed and then used as inputs in the robot main operating routine///////
float rotary_counter=0;//instant number of ticks traveled
float turns_counter=0;// instant number of cycles traveled
float rotary_accum = 0;// total number of ticks traveled in the straight line
float accum_straight_distance=0;// total number of cycles traveled in the straight line
long start_sec=0;// real start time converted to second form
long end_sec=0;// real current time converted to second form
long operation_sec=0;// time difference between start_sec and end_sec in the unit of seconds
unsigned int a = 0;//left detection sensor reading
unsigned int b = 0;//middle detection sensor reading
unsigned int c = 0;//right detection sensor reading
float last_problem_bool[2] = {0,0};//last dispense information: {'crack' -> 0 or 'hole' -> 1, distance from the start line}
bool completion_bool = true;// switch of standby or running
int last_sensed = 0; //1-crack, 2-hole, 0-none identified yet
float drop_position[12] = {0,0,0,0,0,0,0,0,0,0,0,0};//a queue of next positions that the robot supposes to stop and dispense cones
int drop_identity[12] = {0,0,0,0,0,0,0,0,0,0,0,0};//identity of the obstacle the cone(s) should cover: 0-none, 1-left crack,2-middle crack,3-right crack,4-hole
int drop_index = 0;//pointer to the next drop task in the drop queue
int add_index = 0;//where to add the new drop task in the drop queue
bool last_dropped = false;//last obstacle has/will have cone(s) deployed
```

```
bool no_cone = false;//switch of dispense functions

char times[8] = "00000000";//temporary list of start time(hours and minutes) of four latest operations


//"record variables" that will be sent to permanent log/////

int hole_counter =0;//number of holes detected

int crack_counter=0;//number of cracks detected

int array_holes_distance[12] = {0,0,0,0,0,0,0,0,0,0,0,0};//distance between each hole and the start line

int array_cracks_distance[12] = {0,0,0,0,0,0,0,0,0,0,0,0};//distance between each crack and the start line

char start_time[2] = "00";//minute, hour(24)

unsigned char operation_time[3]={1,2,3};//sec, min, hour(24)

unsigned int cones_deployed = 0;


//LCD display state variables/////

int disp_standby_page = 0;//level 1 rotating counter

int ddp = 0;//level 2 rotating counter

int reg =0;//level variable

int display_repeat = 0;//data selection page counter (rotates twice)

int clear_waiter = 3;//wait 3 iterations for user command to confirm clearing history


//constants and parameters/////

const char keys[] = "123A456B789C*0#D";//keypad array

const float car_length = 1.45;

const float mc_L = 0.2;//see usage

const float car_width = 0.45;
```

```
const float half_lane_width = 1.5;

const float lc_R = 0.1;//see usage

const float whole_distance = 15;//distance between destination and the start line

int wety_deg = 600;//delay time for indicator flag to rotate 180 degrees

int ninty_deg = 260;//delay time for indicator flag to rotate 90 degrees


//high-level program state switches//////

bool key = false;//switch of setup mode

int planB = 0;//switch between deployment strategies. 0-1: at plan A, 2-3 at plan B %4 every RESTART make it plan A.


//EEPROM variables//////

char addr =0;//lower 8 bits of the address to write

char addrh = 0;//higher 8 bits of the address to write

char read_pntr = 0;//temporary variable for computing the right address to read a specific type of data

int total_op =0;//total number of operations recorded in EEPROM

int print_data[13];//temporary display content

#endif  /* GLOBAL_VARIABLE_H */
```

3. <configBits.h>

```
/**

 * @file

 * @author Tyler Gamvrelis

 *

 * Created on July 10, 2017, 10:54 AM

 *
```

```
 * @ingroup Config_18F4620
 */

#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 = SBORDIS  // 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 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 */
```

4. <I2C.h>

```
/**
 * @file
 * @author Michael Ding
 * @author Tyler Gamvrelis
 *
 * Created summer 2016
 *
 * @defgroup I2C
 * @brief I2C driver
 * @{
```

```
 */

#ifndef I2C_H
#define I2C_H

/********************************* Includes *********************************/
#include <xc.h>
#include "configBits.h"

/********************************** Macros **********************************/
// These mean different things depending on the context, see "Understanding the
// I2C bus" by Texas Instruments for more details
#define ACK  0 /**< Acknowledge     */
#define NACK 1 /**< Not acknowledge */

/*********************** Public Function Prototypes ***********************/
/**
 * @brief Initializes the MSSP module for I2C mode. All configuration register
 *        bits are written to because operating in SPI mode could change them
 * @param clockFreq The frequency at which data is to be transferred via the
 *        I2C bus
 * @note The argument is used to generate the baud rate according to the
 *       formula clock = FOSC / (4 * (SSPADD + 1)). Because the argument
 *       sets the 7 bits of control signals in the SSPADD register, the
 *       following are the limitations on the value of clockFreq for
 *       FOSC = 40 MHz: Minimum: 78125, Maximum: 10000000
```

```
 */
void I2C_Master_Init(const unsigned long clockFreq);

/**
 * @brief Initiates Start condition on SDA and SCL pins. Automatically cleared
 *        by hardware
 */
void I2C_Master_Start(void);

/**
 * @brief Initiates Repeated Start condition on SDA and SCL pins. Automatically
 *        cleared by hardware
 */
void I2C_Master_RepeatedStart(void);

/**
 * @brief Initiates Stop condition on SDA and SCL pins. Automatically cleared
 *        by hardware
 */
void I2C_Master_Stop(void);

/** @brief Writes a byte to the slave device currently being addressed */
void I2C_Master_Write(unsigned byteToWrite);

/**
 * @brief Reads a byte from the slave device currently being addressed
```

```
 * @param ackBit The acknowledge bit
 *       -# ackBit == 0 --> acknowledge bit sent; ready for next bit
 *       -# ackBit == 1 --> no acknowledge bit (NACK); done reading data
 * @return The byte received
 */
unsigned char I2C_Master_Read(unsigned char ackBit);

/**
 * @}
 */

#endif /* I2C_H */
```

5. <lcd.h>

```
/**
 * @file
 * @author Michael Ding
 * @author Tyler Gamvrelis
 *
 * Created on August 12, 2016, 4:24 PM
 *
 * @defgroup CharacterLCD
 * @brief Driver for Hitachi HD44780-based character LCD
 * @{
 */

#ifndef LCD_H
```

```c
#define LCD_H

/********************************** Includes ***********************************/
#include <xc.h>
#include <stdio.h>
#include <stdbool.h>
#include "configBits.h"

/*********************************** Macros ************************************/
#define RS LATDbits.LATD2
#define E  LATDbits.LATD3

/** @brief Clears both LCD lines */
#define lcd_clear(){\
        lcdInst(0x01);\
        __delay_ms(5);\
}

/** @brief Sets cursor position to start of first line */
#define lcd_home(){\
        lcdInst(0x80);\
        __delay_ms(2);\
}

/**
 * @brief Sets the cursor's position to a specific display data RAM (DDRAM)
```

```
 *       address
 * @param addr The DDRAM address to move the cursor to (min: 0, max: 127)
 * @note The cursor will not be visible at all addresses
 */
#define lcd_set_ddram_addr(addr){\
    lcdInst(0x80 | addr);\
}

/**
 * @brief Backlight and cursor control
 * @param display_on Turns on the backlight if true, otherwise turns it off
 * @param cursor_on Turns on cursor if true, otherwise turns it off
 * @param blink_cursor Blinks the cursor if true, otherwise cursor is static
 */
#define lcd_display_control(\
            display_on,\
            cursor_on,\
            blink_cursor\
)\
{\
            lcdInst(\
        (unsigned char)(8 | (display_on << 2) | (cursor_on << 1) | blink_cursor)\
    );\
}

/******************************** Constants **********************************/
```

```
// Display dimensions as seen in the real world (before you use these in your
// code, double-check that they match the size of your LCD)
extern const unsigned char LCD_SIZE_HORZ; /**< Number of visible columns */
extern const unsigned char LCD_SIZE_VERT; /**< Number of visible rows    */

extern const unsigned char LCD_LINE1_ADDR; /**< Address of first line */
extern const unsigned char LCD_LINE2_ADDR; /**< Address of 2nd line   */
extern const unsigned char LCD_LINE3_ADDR; /**< Address of 3rd line   */
extern const unsigned char LCD_LINE4_ADDR; /**< Address of 4th line   */
/******************************** 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;

/********************************** Types ************************************/
/** @brief The directions the display contents and cursor can be shifted */
typedef enum{
	LCD_SHIFT_LEFT = 0, /**< Left shift  */
	LCD_SHIFT_RIGHT = 1 /**< Right shift */
}lcd_direction_e;

/************************* Public Function Prototypes *************************/
```

```
/**
 * @brief Sends a command to a display control register
 * @param data The command byte for the Hitachi controller
 */
void lcdInst(char data);

/** @brief Performs the initial setup of the LCD */
void initLCD(void);

/**
 * @brief Moves the cursor in a given direction by numChars characters
 * @param numChars The number of character positions by which the cursor is to
 *       be moved (min: 0, max: 127)
 * @param direction The direction for which the shift is to occur
 */
void lcd_shift_cursor(unsigned char numChars, lcd_direction_e direction);

/**
 * @brief Shifts the display in a given direction by numChars characters
 * @param numChars The number of character positions by which the display
 *       contents are to be shifted (min: 0, max: 127)
 * @param direction The direction for which the shift is to occur
 */
void lcd_shift_display(unsigned char numChars, lcd_direction_e direction);

/**
```

```
 * @brief Sends a character to the display for printing
 * @details The familiar C function printf internally calls a function named
 *          "putch" (put character) whenever a character is to be written to
 *          the screen. Here we have chosen to implement putch so that it
 *          sends the character to the LCD, but you can choose to implement it
 *          however you'd like (e.g. send the character over UART, etc.)
 * @param data The character (byte) to be displayed
 */
void putch(char data);


/**
 * @}
 */


#endif  /* LCD_H */
```

6. <main.c>

```
#include <xc.h>
#include "configBits.h"
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include "I2C.h"
#include "lcd.h"
#include "global_variable.h"
#include "prototypes.h"
```

```
void main(void){
    configureports();
    initLCD();
    lcd_display_control(true, false, false);
    unsigned long ticks = 0;
    total_op = read_total();
    while(1){
        if(!key){
            a = PORTBbits.RB2;
            b = PORTBbits.RB3;
            c = PORTAbits.RA1;
            read_encoder();
            if ((completion_bool == false)&&((accum_straight_distance >=
whole_distance)||(operation_sec >= 170))){//continue to adjust time limit
                Stop();
                completion_return();
            }
            else if((no_cone == false)&&(completion_bool ==
false)&&(drop_identity[drop_index]!=0)&&(accum_straight_distance >=
drop_position[drop_index])){
                Stop();
                drop_function();
            }
            else if((completion_bool==false)&&(a+b+c > 0)){
                Stop();
                sensed_function_3();
```

```
    initmoving_disp();
}
else if(completion_bool == false){
    moving();
    if (ticks%1000 == 0){
        normal_updater();
    }
}
else if ((completion_bool == true)&&(ticks%3000 == 0)) {
    lcd_clear();
    if (reg == 0){
        standby_rotating();
    }
    else if ((reg == 10) ||(reg == 20)||(reg == 30)||(reg ==40)){
        data_select();
        if ((ddp ==0)&&(reg%10==0)){
            display_repeat +=1;
        }
        if (display_repeat > 1){
            lcd_clear();
            printf("Returning to");
            lcd_set_ddram_addr(LCD_LINE2_ADDR);
            printf("the Main Menu...");
            __delay_ms(500);
            display_repeat = 0;
            reg = 0;}
```

```
        }
        else if ((reg/10 > 0)&&(reg/10 < 5)&&(reg%10 > 0)){
            data_disp();
            reg = (reg/10)*10;
            display_repeat = 0;
        }
        else if (reg ==50){
            clear_select();
            lcd_set_ddram_addr(LCD_LINE3_ADDR);
            printf("exit in %d sec",clear_waiter);
            clear_waiter = clear_waiter - 1;
            if (clear_waiter ==0){
                clear_waiter = 3;
                reg = 0;
            }
        }
        else if(reg ==51){
            clear_finish();
            reg =0;
            clear_waiter =3;
        }
    }
ticks++;
}
else{//set up: adjust indicator flag position
    LATDbits.LATD0 = 1;
```

```
            LATDbits.LATD1 = 0;

            __delay_ms(1);

            LATDbits.LATD0 = 0;

            __delay_ms(0.8);

        }

    }

}


void configureports(void){

    //configure I/O pins

    TRISD = 0;//D0-D7 LCD display

    //D0 D1 = flag motor

    TRISBbits.RB4 = 1;//keypad

    TRISBbits.RB5 = 1;

    TRISBbits.RB6 = 1;

    TRISBbits.RB7 = 1;

    TRISBbits.RB1 = 1;


    TRISCbits.RC3 = 1;//RTC

    TRISCbits.RC4 = 1;


    TRISBbits.RB0 = 1;//encoder of wheels


    TRISBbits.RB2 = 1;//middle sensor 1

    TRISBbits.RB3 = 1;//middle sensor 2

    TRISAbits.RA1 = 1;//middle sensor 3
```

```
TRISAbits.RA3 = 1;//cone weight sensor

//outputs
TRISAbits.RA4 = 0;//gate 1 servo
TRISAbits.RA5 = 0;//gate 2 servo
LATAbits.LATA4 = 0;
LATAbits.LATA5 = 0;



TRISEbits.RE0 = 1;//1st sensor
TRISEbits.RE1 = 1;//2nd sensor

TRISCbits.RC1 = 0;//wheel w/o encoder
TRISCbits.RC2 = 0;//wheel w/o encoder
TRISCbits.RC5 = 0;//wheel connected
TRISCbits.RC6 = 0;//wheel connected
TRISCbits.RC7 = 0;//wheel w/ encoder
TRISCbits.RC0 = 0;//wheel w/ encoder
LATC =0;
LATD =0;
// Set all A/D ports to digital (pg. 222)
ADCON1 = 0b00001111;
//enable all interrupts
INT0IE = 1;
INTEDG0 = 1;
```

```
    //INT2IE = 1;
    INT1IE = 1;
    ei();

}

void normal_updater(void){//regular time counter updater
    readRTC();
    end_sec = time[2]*3600+time[1]* 60 + time[0];
    operation_sec = end_sec - start_sec;
    operation_time[2] = operation_sec/3600;
    operation_time[1] = (operation_sec%3600)/60;
    operation_time[0] = (operation_sec%3600)%60;
    }
void completion_return(void){
    //communicate with the user about the state
    lcd_clear();
    printf("returning");
    //put down indicator flag
    if (last_sensed == 1){
        LATDbits.LATD0 = 0;
        LATDbits.LATD1 = 1;
        __delay_ms(ninty_deg);//90 deg
        LATDbits.LATD1 = 0;
    }
    else if(last_sensed == 2){
```

```
    LATDbits.LATD0 = 1;
    LATDbits.LATD1 = 0;
    __delay_ms(ninty_deg);//180 deg
    LATDbits.LATD0 = 0;
}
//initiate and prepare distance counter for returning
INT0IF = 0;
INT0IE = 1;//encoder input
INTEDG0 = 1;
turns_counter=0;
rotary_counter=0;
//Slide to the right
while (turns_counter < (car_width/2 + half_lane_width)){
    R_I();
}
Stop();
//Move backward
turns_counter=0;
rotary_counter=0;
while (turns_counter < accum_straight_distance + 1.8 ){
    backw();
}
Stop();
//Slide to the left
turns_counter=0;
rotary_counter=0;
```

```
while (turns_counter < car_width/2 + half_lane_width){
    L_I();
}
Stop();
//clear the counter
turns_counter=0;
rotary_counter=0;
//switch off wheel encoder interrupt
INT0IF = 0;
INT0IE = 0;
//sliding door close if it was open
unsigned long tick = 0;
if (no_cone == true){
    while(tick<60){//pwm period = 20ms, duty cycle = 2.5ms
        LATAbits.LATA5 = 1;
        __delay_ms(2.5);
        LATAbits.LATA5 = 0;
        __delay_ms(16.5);
        tick ++;
        __delay_ms(1);
    }
    tick = 0;
}
//update the final operation time
readRTC();
end_sec = time[2]*3600+time[1]* 60 + time[0];//convert to seconds for one-step subtraction
```

```
    operation_sec = end_sec - start_sec;
    operation_time[2] = operation_sec/3600;
    operation_time[1] = (operation_sec%3600)/60;
    operation_time[0] = (operation_sec%3600)%60;//convert back to hh,mm,ss format
    //write to permanent memory
    completion_write();
    //set global variables that re-direct the branch for next main loop
    completion_bool = true;
    disp_standby_page =0;
}
void initialize_func(void){
    //Restart time and distance counter
    readRTC();
    start_sec = time[2]*3600 + time[1]* 60 + time[0];
    start_time[1] = time[2];
    start_time[0] = time[1];
    operation_sec = 0;
    rotary_accum = 0;
    accum_straight_distance=0;
    turns_counter=0;
    rotary_counter=0;
    //Initialize record data variables
    cones_deployed=0;
    hole_counter = 0;
    crack_counter =0;
    //Initialize operational global variables
```

```
    last_sensed = 0;
    no_cone = false;
    drop_index = 0;
    add_index = 0;
    reg = 0;
    ddp = 0;
    last_dropped = false;
    last_problem_bool[0] = 0;
    last_problem_bool[1] = 0;
    //Clear global variables
    for (unsigned int g = 0;g<12;g++){
        array_holes_distance[g]= 0;//record data
        array_cracks_distance[g]=0;//record data
        drop_identity[g]=0;//operational global variables
        drop_position[g]=0;//operational global variables
    }
    //Enable encoder interrupt
    INT0IE = 1;
    INTEDG0 = 1;
    ei();

    initmoving_disp();//display "running"
    normal_updater();//start operation time counter
    completion_bool = false;//toggle case variable
}
```

7. <dispense.c>

```c
#include <xc.h>
#include "configBits.h"
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>

#include "lcd.h"
#include "global_variable.h"
#include "I2C.h"
#include "prototypes.h"


void drop_function(void){
    if (planB <2){
        if (drop_identity[drop_index] == 1){
            left_crack_drop();
        }
        else if(drop_identity[drop_index] == 2){
            middle_crack_drop();
        }
        else if(drop_identity[drop_index] == 3){
            right_crack_drop();
        }
        else if(drop_identity[drop_index] == 4){//or just else
            hole_dispense_function();
        }
```

```
    }
    else{
        hole_dispense_function();
    }
    drop_index+=1;
}
void hole_dispense_function(void){
    unsigned long tick = 0;
    if (PORTAbits.RA3 == 1){
        lcd_clear();
        printf("cone used up");
        while(tick<70){//gate 1 close
            LATAbits.LATA4 = 1;
            __delay_ms(1.2);

            LATAbits.LATA4 = 0;
            __delay_ms(17.8);
            //}
            tick ++;
            __delay_ms(1);
        }
        tick = 0;
        while(tick<60){//175//1.5//55  gate 2 open
            LATAbits.LATA5 = 1;
            __delay_ms(0.5);//1
```

```
        LATAbits.LATA5 = 0;

        __delay_ms(18.5);

        //}

        tick ++;

        __delay_ms(1);

    }


    tick =0;

    while(tick<70){//gate 1 open

        LATAbits.LATA4 = 1;

        __delay_ms(1.85);//1.9 = 85


        LATAbits.LATA4 = 0;

        __delay_ms(17.15); //17.1

        //}

        tick ++;

        __delay_ms(1);

    }

    tick = 0;

    __delay_ms(100);

    cones_deployed +=1;

    no_cone = true;

}


else{

    tick = 0;
```

```
while(tick<70){//gate 1 close

    LATAbits.LATA4 = 1;
    __delay_ms(1.2);

    LATAbits.LATA4 = 0;
    __delay_ms(17.8);
    //}
    tick ++;
    __delay_ms(1);
}

tick = 0;
/*lcd_clear();
printf("gate 2 open");*/

while(tick<60){
    LATAbits.LATA5 = 1;
    __delay_ms(0.5);//1

    LATAbits.LATA5 = 0;
    __delay_ms(18.5);
    //}
    tick ++;
    __delay_ms(1);
}
```

```
__delay_ms(100);
tick = 0;
/*lcd_clear();
printf("gate 2 close");*/

while(tick<60){
   LATAbits.LATA5 = 1;
   __delay_ms(2.5);

   LATAbits.LATA5 = 0;
   __delay_ms(16.5);
   //}
   tick ++;
   __delay_ms(1);
}
__delay_ms(100);
tick = 0;
/*lcd_clear();
printf("gate 1 open");*/

while(tick<70){
   LATAbits.LATA4 = 1;
   __delay_ms(1.85);
   LATAbits.LATA4 = 0;
   __delay_ms(17.15);
   tick ++;
```

```
        __delay_ms(1);
    }
    tick = 0;
    __delay_ms(100);
    cones_deployed +=1;
    }
}
void middle_crack_drop(void){
        turns_counter =0;
        rotary_counter = 0;
        float A = turns_counter;
        float B = turns_counter;
        while((B - A) < mc_L){
            L_I();
            B = turns_counter;
        }
        Stop();
        hole_dispense_function();
        //

        turns_counter =0;
        rotary_counter = 0;
        A = turns_counter;
        B = turns_counter;
        while((B - A) < car_width){
            R_I();
```

```
        B = turns_counter;
    }
    Stop();
    if(no_cone == false){
    hole_dispense_function();}
    //
    turns_counter =0;
    rotary_counter = 0;
    A = turns_counter;
    B = turns_counter;
    while((B - A) < car_width - mc_L){
        L_I();
        B = turns_counter;
    }
    Stop();
    turns_counter =0;
    rotary_counter = 0;
}


void left_crack_drop(void){
    turns_counter =0;
    rotary_counter = 0;
    float A = turns_counter;
    float B = turns_counter;
    while((B - A) < lc_R-0.05){
```

```
   R_I();
   B = turns_counter;
}
Stop();
hole_dispense_function();
//
turns_counter =0;
rotary_counter = 0;
A = turns_counter;
B = turns_counter;
while((B - A) < car_width-0.05){
   L_I();
   B = turns_counter;
}
Stop();
if(no_cone == false){
hole_dispense_function();}
//
turns_counter =0;
rotary_counter = 0;
A = turns_counter;
B = turns_counter;
while((B - A) < car_width - lc_R){
   R_I();
   B = turns_counter;
}
```

```
        Stop();
        turns_counter =0;
        rotary_counter = 0;
}

void right_crack_drop(void){
        turns_counter =0;
        rotary_counter = 0;
        float A = turns_counter;
        float B = turns_counter;
        while((B - A) < lc_R){
            L_I();
            B = turns_counter;
        }
        Stop();
        hole_dispense_function();
        //
        turns_counter =0;
        rotary_counter = 0;
        A = turns_counter;
        B = turns_counter;
        while((B - A) < car_width){
            R_I();
            B = turns_counter;
        }
        Stop();
```

```
    if(no_cone == false){
    hole_dispense_function();}
    //
    turns_counter =0;
    rotary_counter = 0;
    A = turns_counter;
    B = turns_counter;
    while((B - A) < car_width - lc_R){
       L_I();
       B = turns_counter;
    }
    Stop();
    turns_counter =0;
    rotary_counter = 0;


}
```

8. <display.c>

```
#include <xc.h>
#include "configBits.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#include "lcd.h"
#include "global_variable.h"
```

```
#include "I2C.h"
#include "prototypes.h"

void data_disp(void){
    read_pointer();
    //compute the start of the requested data
    int datalength = 0;
    if(reg%10 == 1){//ot
        read_pntr = (total_op +1 - reg/10)*56 - 4 + 2;
        datalength = 3;}
    else if (reg%10 == 2){//cones
        read_pntr = (total_op +1- reg/10)*56 - 7 + 2;
        datalength = 1;}
    else if (reg%10 == 3){//hole
        read_pntr = (total_op +1- reg/10)*56 - 8 + 2;
        datalength = 25;}
    else if (reg%10 == 4){//crack
        read_pntr = (total_op +1- reg/10)*56 - 33 + 2;
        datalength = 25;}
    subtract();

    if (datalength == 25){
        print_data[0]= one_byte_reader(addr,addrh);
        addr +=1;
        if (addr == 0){
            addrh+=1;
```

```
    }
    int temp_read = 0;
    for (int length_i = 1;length_i<datalength;length_i++){
        if (length_i%2 == 0){
            temp_read += one_byte_reader(addr,addrh);
            print_data[(length_i / 2)] = temp_read;
            temp_read = 0;
        }
        else{
            temp_read += 256 * one_byte_reader(addr,addrh);
        }
        addr +=1;
        if (addr == 0){
            addrh+=1;
        }
    }
}
else{
    for (int length_i = 0;length_i<datalength;length_i++){
        print_data[length_i]= one_byte_reader(addr,addrh);
        addr +=1;
        if (addr == 0){
            addrh+=1;
        }
    }
}
```

```
        if(reg%10 == 1){//ot
            ot(print_data);}
        else if (reg%10 == 2){//cones
            Cones(print_data);}
        else if (reg%10 == 3){//hole
            Holes2(print_data,print_data[0]);}
        else if (reg%10 == 4){//crack
            Cracks(print_data,print_data[0]);}
}
void ot(unsigned int* print_data){
    printf("Operation Time:");
    lcd_set_ddram_addr(LCD_LINE2_ADDR);
    printf("%02d hour",print_data[0]);
    lcd_set_ddram_addr(LCD_LINE3_ADDR);
    printf("%02d min",print_data[1]);
    lcd_set_ddram_addr(LCD_LINE4_ADDR);
    printf("%02d sec",print_data[2]);
    __delay_ms(2000);
}

void Cones(unsigned int* print_data){
    printf("Number of Cones");
    lcd_set_ddram_addr(LCD_LINE2_ADDR);
    printf("Deployed:");
    lcd_set_ddram_addr(LCD_LINE3_ADDR);
    printf("%d",print_data[0]);
```

```
    __delay_ms(2000);
}



void Holes2(unsigned int* a,int b){
    printf("Number of Holes");
    lcd_set_ddram_addr(LCD_LINE2_ADDR);
    printf("Detected:");
    lcd_set_ddram_addr(LCD_LINE3_ADDR);
    printf("%d",a[0]);
    __delay_ms(2000);
    //Distance from Start Line: (cm)
    printintarray2(a,b);


}


void Cracks(unsigned int* a,int b){
    printf("Number of Cracks");
    lcd_set_ddram_addr(LCD_LINE2_ADDR);
    printf("Detected:");
    lcd_set_ddram_addr(LCD_LINE3_ADDR);
    printf("%d",a[0]);
    __delay_ms(2000);
    //Distance from Start Line: (cm)
    printintarray2(a,b);
}
```

```
void page5(void){
    unsigned char curr_line = LCD_LINE1_ADDR;
    for (int op_counter = 0;op_counter < total_op;op_counter++){
        printf("#%d.",op_counter+1);
        read_pntr = (total_op - op_counter)*56;//latest is the first
        read_pointer();
        subtract();//correct
        times[op_counter*2] = one_byte_reader(addr,addrh);
        addr +=1;
        if (addr == 0){
            addrh+=1;
        }
        times[op_counter*2+1] = one_byte_reader(addr,addrh);
        if (times[op_counter*2] >= 12){
            times[op_counter*2] = times[op_counter*2] - 12;
            printf("%d:%02d pm",times[op_counter*2],times[op_counter*2+1]);
        }
        else{
            printf("%d:%02d am",times[op_counter*2],times[op_counter*2+1]);
        }
        curr_line = nextLine(curr_line);
        lcd_set_ddram_addr(curr_line);
    }


}
///////////////////////////////////////
```

```
void clear_select(void){
    printf("Clear?");
    lcd_set_ddram_addr(LCD_LINE2_ADDR);
    printf("C: confirm");

}
void clear_finish(void){
    clear_mem();
    printf("History Cleared");

}



void data_select(void){
    if (ddp == 0){
        data1();
    }
    else if (ddp ==1){
        data2();
    }
    else if (ddp ==2){
        data3();
    }
    else if (ddp ==3){
        data4();
    }
```

```
    ddp = (ddp+1)%4;
}
void standby_rotating(void){

    if (disp_standby_page == 0){
        page1();
    }
    else if (disp_standby_page ==1){
        page2();
    }
    else if (disp_standby_page ==2){
        page3();
    }
    else if (disp_standby_page ==3){
        page4();
    }
    else if (disp_standby_page ==4){
        if (total_op == 0){
            page3();
        }
        else{
            page5();}
    }else if (disp_standby_page ==5){
        page6();
    }
```

```
    disp_standby_page = (disp_standby_page+1)%6;
}
unsigned int digit0(unsigned int h){
        if (h<10){
                return 1;}
        else if (h<100){
                return 2;}
        else if (h<1000){
                return 3;}
    else if (h<10000){
      return 4;
    }
        else{
                return 0;}
}
unsigned char nextLine(unsigned char g){
        if (g == LCD_LINE1_ADDR){
                return LCD_LINE2_ADDR;
    }
        else if (g ==LCD_LINE2_ADDR){
                return LCD_LINE3_ADDR;
    }
        else if (g == LCD_LINE3_ADDR){
                return LCD_LINE4_ADDR;
    }
        else if (g == LCD_LINE4_ADDR){
```

```
            return LCD_LINE1_ADDR;
    }
}
//display in necessary digits, display one page one time.

void printintarray2(unsigned int* a,int b){
    unsigned char curr_line = LCD_LINE3_ADDR;
    unsigned int dig =0;
    unsigned int counter =0;
    unsigned int left =16;

    lcd_clear();

    printf("Distance from");
    lcd_set_ddram_addr(LCD_LINE2_ADDR);
    printf("Start Line: (cm)");
    lcd_set_ddram_addr(LCD_LINE3_ADDR);
    __delay_ms(200);
    for(unsigned int i = 1;i<b+1;i++){//change
        dig = digit0(a[i]);
        counter = counter + 1 + dig;
        if (counter <= 16){
            printf(" ");
            printf("%d",a[i]);
            left = 16-counter;
        }
```

```
        else if (counter > 16){
            curr_line = nextLine(curr_line);
            if (curr_line == LCD_LINE1_ADDR){
                __delay_ms(3000);
                lcd_clear();}
            lcd_set_ddram_addr(curr_line);
            printf(" ");
            printf("%d",a[i]);
            counter = counter - 16 + left;
            left = 16-counter;
        }
    }
    __delay_ms(3000);
}


void page1(void){
    printf("Thanks for using ");
    lcd_set_ddram_addr(LCD_LINE2_ADDR);
    printf("Traffic Cone");
    lcd_set_ddram_addr(LCD_LINE3_ADDR);
    printf("Dispenser");
}
void page3(void){
    readRTC();
    printf("Date");
```

```
    lcd_set_ddram_addr(LCD_LINE2_ADDR);
    printf("20");
    printf("%02d-%02d-%02d",time[6],time[5],time[4]);
    lcd_set_ddram_addr(LCD_LINE3_ADDR);
    printf("Time");
    lcd_set_ddram_addr(LCD_LINE4_ADDR);
    printf("%02d:%02d:%02d",time[2],time[1],time[0]);
}

void page2(void){
    printf("Press A:");
    lcd_set_ddram_addr(LCD_LINE2_ADDR);
    printf("Start a New");
    lcd_set_ddram_addr(LCD_LINE3_ADDR);
    printf("Operation");
}
void page6(void){
    printf("Press 0:");
    lcd_set_ddram_addr(LCD_LINE2_ADDR);
    printf("Clear History");
}
void page4(void){
    printf("Past Operations:");
    lcd_set_ddram_addr(LCD_LINE2_ADDR);
    total_op = read_total();
    if (total_op == 0){
```

```
        printf("     Empty");
    }
    else if (total_op == 1){
        printf("Choose 1");
    }
    else{
    printf("Choose 1 - %d",total_op);}
    //lcd_set_ddram_addr(LCD_LINE3_ADDR);
    //printf("");

}

void data1(void){
    printf("Press 1:");
    lcd_set_ddram_addr(LCD_LINE2_ADDR);
    printf("Operation Time");
}

void data2(void){
    printf("Press 2:");
    lcd_set_ddram_addr(LCD_LINE2_ADDR);
    printf("Number of");
    lcd_set_ddram_addr(LCD_LINE3_ADDR);
    printf("Cones Deployed");

}
```

```
void data3(void){
   printf("Press 3:");
   lcd_set_ddram_addr(LCD_LINE2_ADDR);
   printf("Number and ");
   lcd_set_ddram_addr(LCD_LINE3_ADDR);
   printf("Location of ");
   lcd_set_ddram_addr(LCD_LINE4_ADDR);
   printf("Holes Detected");
}

void data4(void){
   printf("Press 4:");
   lcd_set_ddram_addr(LCD_LINE2_ADDR);
   printf("Number and ");
   lcd_set_ddram_addr(LCD_LINE3_ADDR);
   printf("Location of ");
   lcd_set_ddram_addr(LCD_LINE4_ADDR);
   printf("Cracks Detected");
}
void initmoving_disp(void){
   lcd_clear();
   printf("Running");
}

//old alternatives methods
//delay all in three digits, one page one time
```

```
/*void printintarray1(int* a,int b){
    unsigned char curr_line = LCD_LINE3_ADDR;
    unsigned int counter=0;
    lcd_clear();
    printf("Distance from");
    lcd_set_ddram_addr(LCD_LINE2_ADDR);
    printf("Start Line: (cm)");

    __delay_ms(500);
    lcd_set_ddram_addr(LCD_LINE3_ADDR);
    for(unsigned int i=0;i<b;i++){
        printf("%03d",a[i]);
        printf(" ");
        counter+=1;
        if ((curr_line == LCD_LINE4_ADDR)&&(counter%4 == 0)){
            __delay_ms(3000);
            lcd_clear();
            }

        if (counter%4==0){
            curr_line = nextLine(curr_line);
            lcd_set_ddram_addr(curr_line);
        }
    }
    __delay_ms(3000);
  }*/
```

```
/*void Holes(int* a,int b){

    //number of Holes Detected
    unsigned char hole[] = "8";

    lcd_clear();

    printf("Number of Holes");
    lcd_set_ddram_addr(LCD_LINE2_ADDR);
    printf("Detected:");
    lcd_set_ddram_addr(LCD_LINE3_ADDR);
    printf("%s",hole);
    __delay_ms(2000);
    //Distance from Start Line: (cm)
    printintarray3(a,b);
}


void Holes1(int* a,int b){

    //number of Holes Detected
    unsigned char hole[] = "8";

    lcd_clear();
```

```
    printf("Number of Holes");
    lcd_set_ddram_addr(LCD_LINE2_ADDR);
    printf("Detected:");
    lcd_set_ddram_addr(LCD_LINE3_ADDR);
    printf("%s",hole);
    __delay_ms(2000);
    //Distance from Start Line: (cm)
    printintarray1(a,b);

}*/
/*// display in necessary digits, display one by one, clear one line by one line
void printintarray3(int* a,int b){
    unsigned char curr_line = LCD_LINE3_ADDR;
    unsigned int dig =0;
    unsigned int counter =0;
    unsigned int left =16;

    lcd_clear();

    printf("Distance from");
    lcd_set_ddram_addr(LCD_LINE2_ADDR);
    printf("Start Line: (cm)");
    lcd_set_ddram_addr(LCD_LINE3_ADDR);
    __delay_ms(500);
    //printf("size of a");
```

```
    //printf("%d",sizeof(a)); /fixed by b;
    for(unsigned int i = 0;i<b;i++){
        dig = digit0(a[i]);
        counter = counter + 1 + dig;
        if (counter <=16){
            printf(" ");
            printf("%d",a[i]);
            left = 16-counter;
        }
        else if (counter > 16){
            curr_line = nextLine(curr_line);
            lcd_set_ddram_addr(curr_line);
            printf("                ");
            __delay_ms(500);
            lcd_set_ddram_addr(curr_line);
            printf(" ");
            printf("%d",a[i]);
            counter = counter - 16 + left;
            left = 16-counter;
        }
        //printf("%u",counter);
        //printf("%d",curr_line);
        __delay_ms(1000);
    }
    __delay_ms(2000);
}*/
```

```
/*
void sensingH(void){
//sensed, hole

   lcd_clear();

   //while(1){
   lcd_set_ddram_addr(LCD_LINE1_ADDR);
   printf("Detected: Hole");
   lcd_set_ddram_addr(LCD_LINE3_ADDR);
   printf("   Operating");
   //__delay_ms(500);
   //}

}

void sensingC(void){
//sensed, crack

   lcd_clear();

   //while(1){
   lcd_set_ddram_addr(LCD_LINE1_ADDR);
   printf("Detected: Crack");
   lcd_set_ddram_addr(LCD_LINE3_ADDR);
```

```
    printf("   Operating");


}
*/
```

9.<EEPROM.c>

```
#include <xc.h>

#include "configBits.h"

#include <stdlib.h>

#include <stdio.h>

#include <stdbool.h>


#include "lcd.h"

#include "global_variable.h"

#include "I2C.h"

#include "prototypes.h"


void clear_mem(void){

//set first two data to 02.

    read_pointer();


    while (1){

        EEADRH = addrh;

        EEADR = addr;

        EEDATA = 0;

        EECON1bits.EEPGD = 0;

        EECON1bits.CFGS = 0;
```

```
    EECON1bits.WREN = 1;
    GIE =0;

    EECON2 = 0x55;
    EECON2 = 0xAA;

    EECON1bits.WR =1;

    while(EECON1bits.WR==1){
    }//wait writing to complete

    GIE =1;
    EECON1bits.WREN = 0;

    addr = addr-1;
    if (addr == 0xFF){
        addrh = addrh -1;
    }
    if (addrh == 0xFF){
        break;
    }
}
EEADRH = 0;
EEADR = 0;
EEDATA = 0;
```

```
EECON1bits.EEPGD = 0;
EECON1bits.CFGS = 0;
EECON1bits.WREN = 1;
GIE =0;

EECON2 = 0x55;
EECON2 = 0xAA;

EECON1bits.WR =1;

while(EECON1bits.WR==1){
}//wait writing to complete
GIE =1;
EECON1bits.WREN = 0;

EEADRH = 0;
EEADR = 1;
EEDATA = 2;

EECON1bits.EEPGD = 0;
EECON1bits.CFGS = 0;
EECON1bits.WREN = 1;
GIE =0;

EECON2 = 0x55;
EECON2 = 0xAA;
```

```
    EECON1bits.WR =1;

    while(EECON1bits.WR==1){
    }//wait writing to complete

    GIE =1;
    EECON1bits.WREN = 0;

}

void completion_write(void){
    read_pointer();

    int value = 0;
    while(value <56){

        EEADRH = addrh;
        EEADR = addr;

        if (value <2){
            EEDATA = start_time[1-value];}
        else if (value <5){
            EEDATA = operation_time[4-value];}
        else if (value == 5){
            EEDATA = cones_deployed;}
```

```
else if (value == 6){
    EEDATA = hole_counter;}
else if ((value <31)&&(value%2 == 1)){
    EEDATA = array_holes_distance[(value-7)/2]/256;}
else if ((value <31)&&(value%2 == 0)){
    EEDATA = array_holes_distance[(value-8)/2]%256;}
else if (value == 31){
    EEDATA = crack_counter;}
else if ((value <56)&&(value%2 == 0)){
    EEDATA = array_cracks_distance[(value-32)/2]/256;
}
else if ((value <56)&&(value%2 == 1)){
    EEDATA = array_cracks_distance[(value-33)/2]%256;
}

EECON1bits.EEPGD = 0;
EECON1bits.CFGS = 0;
EECON1bits.WREN = 1;
GIE =0;

EECON2 = 0x55;
EECON2 = 0xAA;

EECON1bits.WR =1;

while(EECON1bits.WR==1){
```

```
        }
        GIE =1;
        EECON1bits.WREN = 0;

        addr +=1;
        if (addr == 0){
            addrh+=1;
        }
        if (addrh == 0x03){//actually not complete since it will not be here
            //EECON1bits.WREN =0;
            printf("memory full");
            break;
        }
        value +=1;
    }
    update_pointer();
}

void update_pointer(void){

    EEADRH = 0;
    EEADR = 0;

    EEDATA = addrh;

    EECON1bits.EEPGD = 0;
```

```
EECON1bits.CFGS = 0;
EECON1bits.WREN = 1;
GIE = 0;

EECON2 = 0x55;
EECON2 = 0xAA;

EECON1bits.WR =1;

while(EECON1bits.WR==1){
}

GIE =1;
EECON1bits.WREN = 0;
//////////////////////////////////
EEADRH = 0;
EEADR = 1;

EEDATA = addr;

EECON1bits.EEPGD = 0;
EECON1bits.CFGS = 0;
EECON1bits.WREN = 1;
GIE = 0;

EECON2 = 0x55;
```

```
    EECON2 = 0xAA;

    EECON1bits.WR =1;

    while(EECON1bits.WR==1){
    }//wait writing to complete

    GIE =1;
    EECON1bits.WREN = 0;
}

void read_pointer(void){
    EECON1bits.EEPGD = 0;
    EECON1bits.CFGS = 0;

    EEADRH = 0;
    EEADR = 0;

    EECON1bits.RD = 1;
    addrh = EEDATA;

    EECON1bits.EEPGD = 0;
    EECON1bits.CFGS = 0;

    EEADRH = 0;
    EEADR = 1;
```

```
    EECON1bits.RD = 1;
    addr = EEDATA;

}

void subtract(void){
    if (addr >= read_pntr){
        addr = addr - read_pntr;}
    else{
        addrh = addrh - 1;
        addr = addr - read_pntr;}
}
char one_byte_reader(char r, char rh){
    EECON1bits.EEPGD = 0;
    EECON1bits.CFGS = 0;
    EEADRH = rh;
    EEADR = r;
    EECON1bits.RD = 1;
    char out= EEDATA;
    return out;
}

int read_total(void){
    read_pointer();
    /*printf(",%d,%d,",addrh,addr);
```

```
    __delay_ms(1000);*/
    int total = ((addrh*256 + addr)-2)/56;
    /*printf("%d",total);
    __delay_ms(1000);*/
    if (total > 4){
        total = 4;
    }
    return total;
}
```

10. <encoder_motor.c>

```
#include <xc.h>
#include "configBits.h"
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>

#include "lcd.h"
#include "global_variable.h"
#include "I2C.h"
#include "prototypes.h"



void read_encoder(void){
    rotary_accum+=rotary_counter;
    accum_straight_distance= rotary_accum / 334;
    turns_counter = 0;
```

```
    rotary_counter = 0;
}

void moving(void){
    if ((PORTEbits.RE0 && PORTEbits.RE1) ==1){
        straight();
    }
    if(PORTEbits.RE1 ==0){
        turn_left();
    }
    else if (PORTEbits.RE0 ==0){
        turn_right();
    }
}
void straight(void){
    LATCbits.LATC5 = 0;
    LATCbits.LATC1 = 0;
    LATCbits.LATC7 = 0;

    LATCbits.LATC2 = 1;
    LATCbits.LATC6 = 1;
    LATCbits.LATC0 = 1;
    __delay_ms(7);
    LATCbits.LATC2 = 0;
    LATCbits.LATC6 = 1;
    LATCbits.LATC0 = 1;
```

```
    __delay_ms(1);
    LATCbits.LATC2 = 0;
    LATCbits.LATC6 = 0;
    LATCbits.LATC0 = 0;
    __delay_ms(2.5);
}
void turn_left(void){//10ms
    LATCbits.LATC5 = 0;
    LATCbits.LATC1 = 0;
    LATCbits.LATC7 = 0;

    LATCbits.LATC2 = 1;
    LATCbits.LATC6 = 1;
    LATCbits.LATC0 = 1;
    __delay_ms(8);
    LATCbits.LATC2 = 1;
    LATCbits.LATC6 = 0;
    LATCbits.LATC0 = 0;
    __delay_ms(1);
    LATCbits.LATC2 = 0;
    LATCbits.LATC6 = 0;
    LATCbits.LATC0 = 0;
    __delay_ms(1.5);
}
void turn_right(void){//10ms 8 1 1
    LATCbits.LATC5 = 0;
```

```
    LATCbits.LATC1 = 0;
    LATCbits.LATC7 = 0;

    LATCbits.LATC2 = 1;
    LATCbits.LATC6 = 1;
    LATCbits.LATC0 = 1;
    __delay_ms(5);
    LATCbits.LATC2 = 0;
    LATCbits.LATC6 = 1;
    LATCbits.LATC0 = 1;
    __delay_ms(4);
    LATCbits.LATC2 = 0;
    LATCbits.LATC6 = 0;
    LATCbits.LATC0 = 0;
    __delay_ms(1.5);
}

void L_I(void){
    LATCbits.LATC6 = 0;
    LATCbits.LATC1 = 0;
    LATCbits.LATC7 = 0;

    LATCbits.LATC0 = 1;
    LATCbits.LATC5 = 1;
    LATCbits.LATC2 = 1;
    __delay_ms(4.5);
```

```
    LATCbits.LATC5 = 0;
    __delay_ms(1);
    LATCbits.LATC0 = 1;
    LATCbits.LATC2 = 0;
    __delay_us(700);
}
void Stop(void){
    LATCbits.LATC6 = 0;
    LATCbits.LATC1 = 0;
    LATCbits.LATC7 = 0;
    LATCbits.LATC0 = 0;
    LATCbits.LATC5 = 0;
    LATCbits.LATC2 = 0;
}
void R_I(void){
    LATCbits.LATC0 = 0;
    LATCbits.LATC5 = 0;
    LATCbits.LATC2 = 0;

    LATCbits.LATC6 = 1;
    LATCbits.LATC1 = 1;
    LATCbits.LATC7 = 1;
    __delay_ms(5);
    LATCbits.LATC6 = 0;
    LATCbits.LATC1 = 0;
    __delay_ms(1.2);
```

```
    LATCbits.LATC1 = 1;
    __delay_ms(0.5);
    LATCbits.LATC6 = 0;
    LATCbits.LATC1 = 0;
    LATCbits.LATC7 = 1;
    __delay_us(700);
}
void backw(void){
    LATCbits.LATC6 = 0;
    LATCbits.LATC2 = 0;
    LATCbits.LATC0 = 0;

    LATCbits.LATC1 = 1;
    LATCbits.LATC5 = 1;
    LATCbits.LATC7 = 1;
    __delay_us(100);
}
```

11.<I2C.c>

```
/**
 * @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;
}
```

12.<interrupt handler.c>

```
#include <xc.h>
#include "configBits.h"
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>

#include "lcd.h"
#include "global_variable.h"
#include "I2C.h"
#include "prototypes.h"

void high_priority interrupt interruptHandler(void){
    // Interrupt on change handler for RB1
    if(INT1IF){
        // Notice how we keep the interrupt processing very short by simply
        // setting a "flag" which the main program loop checks
        INT1IF = 0; // Clear interrupt flag bit to signify it's been handled
        char keypress = (PORTB & 0xF0) >> 4;
        char command = keys[keypress];
        //lcd_clear();
```

```
//0-4 :48 49 50 51 52;
if((command == '1') ||(command == '2')||(command == '3')||(command =='4') ){

   if (reg == 0){
      if ((command - 48) > total_op){
         lcd_clear();
         printf("Oops. I don't get");
         lcd_set_ddram_addr(LCD_LINE2_ADDR);
         printf("it... Please");
         lcd_set_ddram_addr(LCD_LINE3_ADDR);
         printf("Review the Menu.");
         __delay_ms(500);
      }
      else{
      reg = (command - 48)*10;
      display_repeat = 0;}
   }
   else if ((reg == 10)||(reg == 20)||(reg == 30)||(reg == 40)){
      reg = reg + command - 48;
   }
}
else if (command == 48){

   if (reg == 0){
      reg =50;
   }
```

```
}
else if (command == 'C'){

    if(reg == 50){
        reg = 51;
        clear_waiter = 3;
    }
}
else if(command =='A'){//start command

    initialize_func();
}
else if(command == 'D'){
    key = !key;
    LATDbits.LATD0 = 0;
    LATDbits.LATD1 = 0;
}
else if (command == 'B'){
    lcd_clear();
    planB +=1;
    planB = planB%4;
    if (planB%4 == 1){
        printf("planB?");
        }//pressed once
    else if (planB%4 == 2){
        printf("planB set");
```

```
        }
    else if (planB%4 == 3){
        printf("planA?");
    }
    else if (planB%4 == 0){
        printf("planA set");
    }
    __delay_ms(800);
}

else{

    if (reg == 50){
        reg = 0;
    }
    else{
        lcd_clear();
        printf("Oops. I don't get");
        lcd_set_ddram_addr(LCD_LINE2_ADDR);
        printf("it... Please");
        lcd_set_ddram_addr(LCD_LINE3_ADDR);
        printf("Review the Menu.");
        __delay_ms(500);
    }
}
```

```
    }
    else if(INT0IF){
        int b0 = PORTBbits.RB0;
        if (b0==1){//read high inputs
            rotary_counter +=1;
            turns_counter = rotary_counter / 334;
        }
        INT0IF= 0;
    }
}
```

13.<lcd.c>

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

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


/***************************** 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();
}
```

```c
/**
 * @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);
}
```

14.<RTC.c>

```
#include <xc.h>
#include "configBits.h"
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>

#include "lcd.h"
#include "global_variable.h"
#include "I2C.h"
#include "prototypes.h"



void readRTC(void){
    I2C_Master_Init(100000);
    // Reset RTC memory pointer
        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
        char t_data = 0;
```

```
        I2C_Master_Start(); // Start condition
        I2C_Master_Write(0b11010001); // 7 bit RTC address + Read
        for(unsigned char i = 0; i < 6; i++){
            t_data = I2C_Master_Read(ACK);
            time[i] = (t_data>>4)*10+(t_data&0x0F);
        }
        t_data = I2C_Master_Read(NACK);
        time[6] = (t_data>>4)*10+(t_data&0x0F); // Final Read with NACK
        I2C_Master_Stop(); // Stop condition
}

void rtc_set_time(void){
    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(happynewyear[i]);
    }

    I2C_Master_Stop(); //Stop condition
}
```

15.<sensor.c>

```
#include <xc.h>
#include "configBits.h"
```

```
#include <stdlib.h>

#include <stdio.h>

#include <stdbool.h>


#include "lcd.h"

#include "global_variable.h"

#include "I2C.h"

#include "prototypes.h"


int hole_drop_bool_function(void){

    if (last_dropped == false){

        last_dropped = true;return 1;

    }

    else{

    float x = last_problem_bool[0];

    float y = last_problem_bool[1];

    //bool last_droped = false;or true

    //if false, drop

    //if true, decide.


    if (x + y ==0){

        last_dropped = true;return 1;

    }

    else if (last_problem_bool[0] == 1) {//hole

        if (accum_straight_distance - y >= 0.55){ // 15//unit in cm

        last_dropped = true;return 1;}
```

```
        else{
            last_dropped = false;
            return 0;}
    }
    else{
        if (accum_straight_distance - y >= 0.75){//20 ~ 0.75 cycles ish
            last_dropped = true;return 1;}
        else{last_dropped = false;return 0;}
    }
}
}
int crack_drop_bool_function(void){
    if (last_dropped == false){
        last_dropped = true;
        return 1;
    }
    else{

    float x = last_problem_bool[0];
    float y = last_problem_bool[1];
    if (x+y ==0){
        last_dropped = true;
        return 1;
    }
    else if (last_problem_bool[0] == 1) {//last is hole
        if (accum_straight_distance - y >= 0.75){  //unit in cm
```

```
            last_dropped = true;
            return 1;}
        else{
            last_dropped = false;
            return 0;}
    }
    else{
        if (accum_straight_distance - y >= 0.36){//10 ~ 0.5
            last_dropped = true;
            return 1;
        }
        else{last_dropped = false;return 0;
        }
    }
}
}

void drop_record(int aa){
    drop_identity[add_index] = aa;
    drop_position[add_index] = accum_straight_distance + car_length;
    add_index +=1;
    /*printf("hhhhh,%d,%.2f",aa,accum_straight_distance + car_length);
    __delay_ms(2000);*/
}
void sensed_function_3(void){
    read_encoder();
```

```
if (a+c == 2){
    int sth = crack_drop_bool_function();
    if (sth){
        drop_record(2);
        //middle_crack_drop();
        last_problem_bool[0] = 0;
        last_problem_bool[1] = accum_straight_distance;
    }
    record('c');
}
else if (a == 1){
    /*printf("left crack");
    __delay_ms(2000);*/
    int sth = crack_drop_bool_function();
    if (sth){
        drop_record(1);
        //left_crack_drop();
        last_problem_bool[0] = 0;
        last_problem_bool[1] = accum_straight_distance;
    }
    record('c');
}
else if (c ==1){
    /*printf("right crack");
    __delay_ms(2000);*/
    int sth = crack_drop_bool_function();
```

```
        if (sth){
            drop_record(3);
            //right_crack_drop();
            last_problem_bool[0] = 0;
            last_problem_bool[1] = accum_straight_distance;
        }
        record('c');
    }
    else{
        /*printf("hole");
        __delay_ms(2000);*/
        int sth = hole_drop_bool_function();
        if (sth ==1){
            drop_record(4);
            //hole_dispense_function();
            last_problem_bool[0] = 1;
            last_problem_bool[1] = accum_straight_distance;
        }
        record('h');
    }
    while(PORTBbits.RB3 + PORTAbits.RA1 + PORTBbits.RB2 > 0){
        moving();
    }
    Stop();
}
void record(char corh){
```

```
if (corh == 'c'){
    array_cracks_distance[crack_counter] = (int) (accum_straight_distance*28.3);//27.5
    crack_counter +=1;
    if (last_sensed == 2){
        LATDbits.LATD0 = 1;
        LATDbits.LATD1 = 0;
        __delay_ms(wety_deg);//180 deg
    }
    else if(last_sensed == 0){
        LATDbits.LATD0 = 1;
        LATDbits.LATD1 = 0;
        __delay_ms(ninty_deg);//90 deg
    }
    else{
        LATDbits.LATD0 = 1;
        LATDbits.LATD1 = 0;
        __delay_ms(50);//shake
        LATDbits.LATD0 = 0;
        LATDbits.LATD1 = 0;
        __delay_ms(50);//shake
        LATDbits.LATD0 = 0;//
        LATDbits.LATD1 = 1;
        __delay_ms(50);//shake
    }
    LATDbits.LATD0 = 0;
    LATDbits.LATD1 = 0;
```

```
    last_sensed = 1;

}
else if (corh == 'h'){
    array_holes_distance[hole_counter] = (int) (accum_straight_distance*28.3);
    hole_counter +=1;
    if (last_sensed == 1){
        LATDbits.LATD0 = 1;
        LATDbits.LATD1 = 0;
        __delay_ms(wety_deg);//180 deg
    }
    else if (last_sensed == 0){
        LATDbits.LATD0 = 0;
        LATDbits.LATD1 = 1;
        __delay_ms(ninty_deg);//90 deg
    }
    else{
        LATDbits.LATD0 = 1;
        LATDbits.LATD1 = 0;
        __delay_ms(50);//shake
        LATDbits.LATD0 = 0;
        LATDbits.LATD1 = 0;
        __delay_ms(50);//shake
        LATDbits.LATD0 = 0;
        LATDbits.LATD1 = 1;
```

```
        __delay_ms(50);//shake
    }
    LATDbits.LATD0 = 0;
    LATDbits.LATD1 = 0;
    last_sensed = 2;

  }
  else{
    printf("wrong");
    __delay_ms(10000);
  }

}

//original idea of moving sensor
/*void sensed_function_1(void){
    lcd_clear();
    printf("sensed");
    //update straight distance,then stop counting
    accum_straight_distance+=read_encoder_function(Perimeter_wheels);
    LATC = Stop;
    INT0IF = 0;
    INT0IE = 0;
    //TRISBbits.RB0 = 0;

    shape_distance=0;
```

```
sensor_return_distance=0;
turns_counter =0;
rotary_counter = 0;
//enable sensor encoder
TRISBbits.RB2 = 1;
INT2IF = 0;
INT2IE = 1;
INTEDG2 = 1;
while (PORTAbits.RA0 == 0){
   LATEbits.LATE0 = 0;
   LATEbits.LATE1 = 1;//left
}
   printf("moving left");
   __delay_ms(1000);
   LATEbits.LATE0 = 0;
   LATEbits.LATE1 = 0;//Stop
sensor_return_distance+= read_encoder_function(Perimeter_sensor_mover);
printf("moved %0.1f",sensor_return_distance);
__delay_ms(1000);
while(PORTAbits.RA0 !=0){// adjust sensor position to enter the tape area again
   LATEbits.LATE0 = 1;
   LATEbits.LATE1 = 0;//Right

}
   printf("moving right null");
   __delay_ms(1000);
```

```
turns_counter =0;
rotary_counter = 0;

while(PORTAbits.RA0 == 0){
   LATEbits.LATE0 = 1;
   LATEbits.LATE1 = 0;//Right
   }
printf("moving right");
   __delay_ms(1000);

LATEbits.LATE0 = 0;
LATEbits.LATE1 = 0;//Stop
shape_distance+= read_encoder_function(Perimeter_sensor_mover);
printf("shape distance %0.1f",shape_distance);
__delay_ms(1000);
sensor_return_distance =shape_distance-sensor_return_distance;
printf("return %0.1f cm",sensor_return_distance);
__delay_ms(1000);
turns_counter =0;
rotary_counter = 0;
while(move_encoder_function(Perimeter_sensor_mover,sensor_return_distance)){
   LATEbits.LATE0 = 0;
   LATEbits.LATE1 = 1; //left

}
   printf("sensor returning");
```

```
    __delay_ms(1000);
//INT2IF = 0;
//INT2IE = 0;
TRISBbits.RB2 = 0;


LATEbits.LATE0 = 0;
LATEbits.LATE1 = 0;//Stop
if (distinguish_H_C_function() == 1){
   sensingH();
   TRISCbits.RC0=0;
   LATCbits.LATC0 = 1;
   if (hole_drop_bool_function() ==1){
      hole_dispense_function();
      cones_deployed+=1;
   }
   hole_counter +=1;
   //last_problem_bool = 1;
   array_holes_distance[hole_counter] = (int)(accum_straight_distance);
   while(PORTAbits.RA0 ==0){
   LATC = forw;}
   LATC = Stop;}
else if (distinguish_H_C_function()==0){
   sensingC();
   TRISCbits.RC0=0;
   LATCbits.LATC0 = 0;
   if (crack_drop_bool_function() ==1){
```

```
            crack_dispense_function();
            cones_deployed+=1;}
          crack_counter+=1;
          //last_problem_bool =0;
          array_cracks_distance[crack_counter] =(int) (accum_straight_distance);
          while(PORTAbits.RA0 ==0){
            LATC = forw;} //moved over the solved problem. distance count
          LATC = Stop;
        }
        //end of the external interrupt handler


}
int distinguish_H_C_function(void){
    if ((shape_distance < 6) && (shape_distance >2)){
        return 1;}//hole
    else if ((shape_distance >13)&&(shape_distance <17)){
        return 0;}//crack
    else{
        return 3;}
}*/
```

16. PC Interface Program

```
import numpy as np
raw = np.loadtxt(open("/Users/Chen/Desktop/Microcontroller/pcinterface.txt","rt"), delimiter="
        ",dtype = "int32",converters={_:lambda s: int(s, 16) for _ in range(16)})
#print(raw)
print("-"*20,"start of report","-"*20,"\n")
```

```
nlength = (256*raw[0][0]+raw[0][1])
print("Permanent Memory used %.2f percent" % float(nlength/1024*100))
row = int(nlength/16)
remaining = row%16
column = 16
nlist = []
for i in range(row):
    for j in range(column):
        nlist.append(raw[i][j])
for i in range(remaining):
    nlist.append(raw[row][i])
#print(nlist)
temp = 0
for i in range(0,nlength):
    if (i-2)%56 == 0:
        print("\nAt %02d:"%nlist[i],end = "")
    elif (i-2)%56 == 1:
        print("%02d"%nlist[i],end = "")
        if nlist[i-1]<12:
            print("am")
        else:
            print("pm")
    elif (i-2)%56 == 2:
        print("Operated %d hours,"%nlist[i],end = "")
    elif (i-2)%56 == 3:
        print("%d minutes,"%nlist[i],end = "")
```

```
elif (i-2)%56 == 4:
    print("%d seconds"%nlist[i])
elif (i-2)%56 == 5:
    print("Deployed %d cones"%nlist[i])
elif (i-2)%56 == 6:
    print("Detected %d holes"%nlist[i])
    print("Their distances from Start Line(cm):")
elif (((i-2)%56 < 31)and(((i-2)%56) %2 ==1)):
    temp= nlist[i]*256
elif (((i-2)%56 < 31)and(((i-2)%56) %2 ==0)):
    temp+=nlist[i]
    if temp!=0:
        print(" %d"%temp,end = "")
elif (i-2)%56 == 31:
    print("\nDetected %d cracks"%nlist[i])
    print("Their distances from Start Line(cm):")
elif (((i-2)%56 < 56)and(((i-2)%56) %2 ==0)):
    temp= nlist[i]*256
elif (((i-2)%56 < 56)and(((i-2)%56) %2 ==1)):
    temp+=nlist[i]
    if temp!=0:
        print(" %d"%temp,end = "")
```

Appendix C: Additional Diagrams and Datasheets

1. U.S. Patent: Highway Cone Dispenser and Collector

United States Patent [19]
Luoma

[11] **Patent Number: 5,054,648**
[45] **Date of Patent: Oct. 8, 1991**

[54] **HIGHWAY CONE DISPENSER AND COLLECTOR**

[76] Inventor: **Eugene H. Luoma,** 1325 Normanna Rd., Duluth, Minn. 55803

[21] Appl. No.: **468,569**

[22] Filed: **Jan. 23, 1990**

[51] **Int. Cl.**[5] **B65G 65/18**
[52] **U.S. Cl.** **221/185;** 414/439; 414/507; 404/6; 198/512; 198/518
[58] **Field of Search** 198/512, 518, 624; 414/437, 439, 440, 507, 529; 221/185; 404/6, 9

[56] **References Cited**

U.S. PATENT DOCUMENTS

2,632,411	3/1953	Cordes	198/624
3,082,908	3/1963	Ingham, Jr. .	
3,096,868	7/1963	Hendrikson et al.	198/512
3,124,231	3/1964	Ott	198/624
3,157,267	11/1964	Asbury .	
3,232,408	2/1966	Asbury	198/510
3,603,479	9/1971	Kuster et al.	198/624
4,193,512	3/1980	Buxton et al. .	
4,747,515	5/1988	Kasher et al. .	
4,792,271	12/1988	Akel	414/440
4,936,485	6/1990	Downing	221/185

FOREIGN PATENT DOCUMENTS

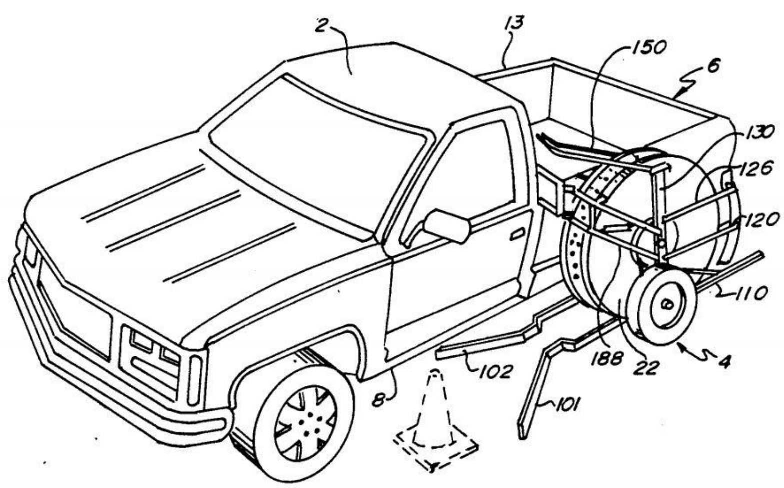
118757	6/1958	U.S.S.R.	414/437

Primary Examiner—Joseph E. Valenza
Attorney, Agent, or Firm—Palmatier & Sjoquist

[57] **ABSTRACT**

A highway marker cone dispenser and collector is provided. A rotatable cone conveyor moves the cones between upper and lower locations, the lower location being near the ground, and has a cone stripper for removing cones from the conveyor at either the upper or lower location. The device may be vehicle mounted for simultaneous rotational and translational movement.

41 Claims, 8 Drawing Sheets



2. U.S. Patent: Device for the Placement and If Desired the Collection of Traffic Cone

US005525021A

United States Patent [19]
Larguier

[11] **Patent Number:** **5,525,021**
[45] **Date of Patent:** **Jun. 11, 1996**

[54] **DEVICE FOR THE PLACEMENT AND IF DESIRED THE COLLECTION OF TRAFFIC CONES**

[75] Inventor: **Frederic Larguier**, Lavalette, France

[73] Assignee: **Baltic Ingenierie**, Carquefou Cedex, France

[21] Appl. No.: **301,706**

[22] Filed: **Sep. 7, 1994**

[30] **Foreign Application Priority Data**

Sep. 7, 1993 [FR] France 93 10696

[51] **Int. Cl.**^{6} **B60P 1/40**
[52] **U.S. Cl.** **414/551**; 414/788.2; 414/795.6
[58] **Field of Search** 414/551, 788.2, 414/789.7, 795.6

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,597,706 7/1986 Michit 414/798.7

FOREIGN PATENT DOCUMENTS

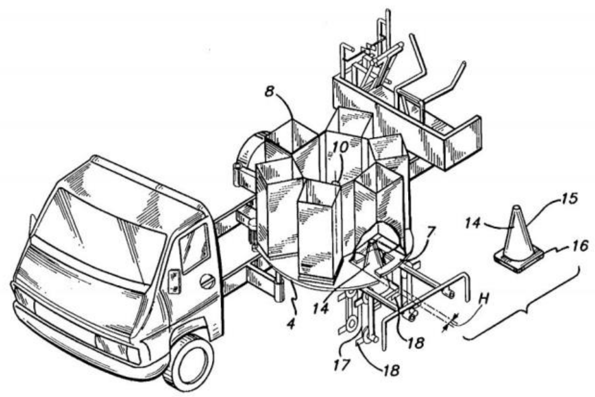
0501106 9/1992 European Pat. Off. .
2556378 5/1986 France .
2657313 7/1991 France .
2747183 4/1978 Germany .

Primary Examiner—David A. Bucci
Assistant Examiner—Janice L. Krizek
Attorney, Agent, or Firm—Young & Thompson

[57] **ABSTRACT**

A vehicle-mounted device for positioning traffic cones (**14**) on the street and for collecting them from the street. Each cone has a conical hollow body (**15**) and a base (**16**). The cones are stacked and nested in a supply (**8**) comprising a hollow vertical body open at least at its base (**28**) to permit the reception of the base (**16**) of a cone. A platform (**4**) is disposed about a vertical axis (**10**) and disposed below the base (**28**) of the supply (**8**). The platform (**4**) has at least one opening (**6**) and a fork formed of two fingers (**7**) which are substantially horizontal and are spaced from each other in a direction radially of the platform so as to let pass only the body (**15**) of a cone and not the base (**16**) of the cone. These fingers are disposed at a height (H) above the platform (**4**) at least equal to the thickness of the base (**16**) of a cone such that in the course of movement of the supply (**8**) and the platform (**4**) relative to each other in rotation about the vertical axis (**10**) in a direction such that the fingers of the fork enter the stack of cones contained in the supply, the opening (**28**) of the supply will pass above the opening (**6**) provided in the platform (**4**) and the fingers (**7**) of the fork will penetrate between the bases (**16**) of the last cone (**14**) of the stack in the supply and the next-to-last cone, thereby detaching from the stack during such relative displacement the last cone which then descends to the street and retaining the next-to-last cone. Much the same structure can be used during collection of the cones from the street.

11 Claims, 9 Drawing Sheets



3. L298N Motor Driver Board Datasheet

L298

DUAL FULL-BRIDGE DRIVER

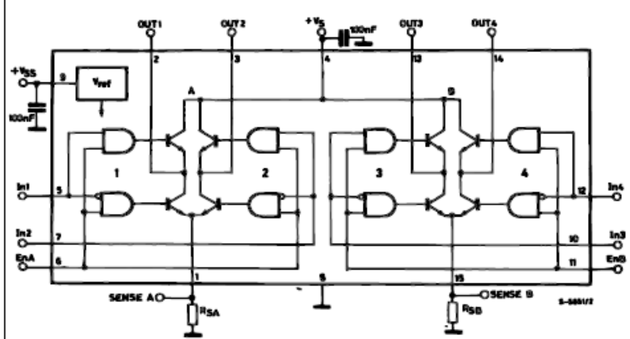
- OPERATING SUPPLY VOLTAGE UP TO 46 V
- TOTAL DC CURRENT UP TO 4 A
- LOW SATURATION VOLTAGE
- OVERTEMPERATURE PROTECTION
- LOGICAL "0" INPUT VOLTAGE UP TO 1.5 V (HIGH NOISE IMMUNITY)

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

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.

BLOCK DIAGRAM



January 2000 1/13

4. LM338 Voltage Regulator Datasheet

LM138 and LM338 5-Amp Adjustable Regulators

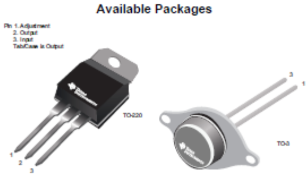
1 Features

- Specified 7-A Peak Output Current
- Specified 5-A Output Current
- Adjustable Output Down to 1.2 V
- Specified Thermal Regulation
- Current Limit Constant With Temperature
- P + Product Enhancement Tested
- Output is Short-Circuit Protected

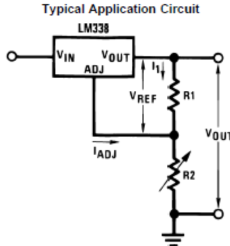
2 Applications

- Adjustable Power Supplies
- Constant Current Regulators
- Battery Chargers

Available Packages



Typical Application Circuit



3 Description

The LM138 series of adjustable 3-terminal positive voltage regulators is capable of supplying in excess of 5 A over a 1.2-V to 32-V output range. They are exceptionally easy to use and require only 2 resistors to set the output voltage. Careful circuit design has resulted in outstanding load and line regulation, comparable to many commercial power supplies. The LM138 family is supplied in a standard 3-lead transistor package.

A unique feature of the LM138 family is time-dependent current limiting. The current limit circuitry allows peak currents of up to 12 A to be drawn from the regulator for short periods of time. This allows the LM138 to be used with heavy transient loads and speeds start-up under full-load conditions. Under sustained loading conditions, the current limit decreases to a safe value protecting the regulator. Also included on the chip are thermal overload protection and safe area protection for the power transistor. Overload protection remains functional even if the adjustment (ADJ) pin is accidentally disconnected.

Normally, no capacitors are needed unless the device is situated more than 6 inches from the input filter capacitors in which case an input bypass is needed. An output capacitor can be added to improve transient response, while bypassing the adjustment pin increases the ripple rejection of the regulator.

Besides replacing fixed regulators or discrete designs, the LM138 is useful in a wide variety of other applications. Because the regulator is floating and receives only the input-to-output differential voltage, supplies of several hundred volts can be regulated as long as the maximum input to output differential is not exceeded; do not short-circuit output to ground. The part numbers in the LM138 series which have a *K* suffix are packaged in a standard steel TO-CAN package, while those with a *T* suffix are packaged in a TO-220 plastic package. The LM138 is rated for T_J = –55°C to 150°C, and the LM338 is rated for T_J = 0°C to 125°C.

Device Information(1)

PART NUMBER	PACKAGE	BODY SIZE (NOM)
LM138	TO-CAN (2)	25.40 mm × 38.94 mm
LM338	TO-220 (3)	10.16 mm × 14.986 mm
	TO-CAN (2)	25.40 mm × 38.94 mm

(1) For all available packages, see the orderable addendum at the end of the data sheet.

8 Application and Implementation

NOTE

Information in the following applications sections is not part of the TI component specification, and TI does not warrant its accuracy or completeness. TI's customers are responsible for determining suitability of components for their purposes. Customers should validate and test their design implementation to confirm system functionality.

8.1 Application Information

In operation, the LM138 develops a nominal 1.25-V reference voltage (V_{REF}) between the output and adjustment terminal. The reference voltage is impressed across program resistor R_1 and, since the voltage is constant, a constant current I_1 then flows through the output set resistor R_2, giving an output voltage calculated with Equation 1.

$$V_{OUT} = V_{REF}\left(1 + \frac{R2}{R2}\right) + I_{ADJ}R2 \quad (1)$$

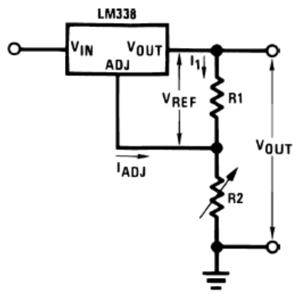
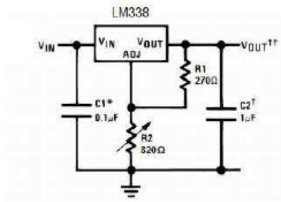


Figure 15. Typical Application Circuit

Because the 50-µA current from the adjustment terminal represents an error term, the LM138 was designed to minimize I_{ADJ} and make it very constant with line and load changes. To do this, all quiescent operating current is returned to the output establishing a minimum load current requirement. If there is insufficient load on the output, the output rises.

8.2 Typical Applications

8.2.1 Constant 5-V Regulator



*Needed if device is more than 6 inches from filter capacitors.

†Optional—improves transient response

†† $V_{OUT} = 1.25\,V\left(1+\frac{R2}{R1}\right)+I_{ADJ}\,(R_2)$

Figure 16. Constant 5-V Regulator

8.2.1.1 Design Requirements

R1: Because the LM138 produces a typical 1.24 V potential between the OUTPUT and ADJUST pins, placing a 270-Ω resistor between them causes 4.6 mA to flow through R1 and R2.

R2: To achieve a 5-V output, the sum of the voltages across R1 and R2 must equal 5 V. Therefore, Vr2 must equal 3.76 V when 4.6 mA is flowing through it. R2 = Vr2 / I = 3.76 V / 4.6 mA = ~820 Ω.

C_{IN}: 0.1 μF of input capacitance helps filter out unwanted noise, especially if the regulator is located far from the power supply filter capacitors.

C_{OUT}: The regulator is stable without any output capacitance, but adding a 1-μF capacitor improves the transient response.

C_{ADJ}: A 10-μF capacitor bypassing the ADJUST pin to ground improves the regulators ripple rejection.

D1: Protection diode D1 is recommended if C_{OUT} is used. The diode provides a low-impedance discharge path to prevent the capacitor from discharging into the output of the regulator (see *Protection Diodes*).

D2: Protection diode D2 is recommended if C_{ADJ} is used. The diode provides a low-impedance discharge path to prevent the capacitor from discharging into the output of the regulator (see *Protection Diodes*).

Table 1 lists the design parameters for this typical application.

Table 1. Design Parameters

PARAMETER	VALUE
Feedback resistor 1 (R1)	270 Ω
Feedback resistor 2 (R2)	820 Ω
Input capacitor (C_{IN})	0.1 µF
Output capacitor (C_{OUT})	1 µF
Adjust capacitor(C_{ADJ})	10 µF

8.2.1.2 Detailed Design Procedure

8.2.1.2.1 External Capacitors

An input bypass capacitor is recommended. A 0.1-µF disc or 1-µF solid tantalum on the input is suitable input bypassing for almost all applications. The device is more sensitive to the absence of input bypassing when adjustment or output capacitors are used but the above values eliminate the possibility of problems.

The adjustment terminal can be bypassed to ground on the LM138 to improve ripple rejection. This bypass capacitor prevents ripple from being amplified as the output voltage is increased. With a 10-µF bypass capacitor, 75-dB ripple rejection is obtainable at any output level. Increases over 20 µF do not appreciably improve the ripple rejection at frequencies above 120 Hz. If the bypass capacitor is used, it is sometimes necessary to include protection diodes to prevent the capacitor from discharging through internal low current paths and damaging the device.

In general, the best type of capacitors to use are solid tantalum. Solid tantalum capacitors have low impedance even at high frequencies. Depending upon capacitor construction, it takes about 25 µF in aluminum electrolytic to equal 1-µF solid tantalum at high frequencies. Ceramic capacitors are also good at high frequencies; but some types have a large decrease in capacitance at frequencies around 0.5 MHz. For this reason, 0.01-µF disc may seem to work better than a 0.1-µF disc as a bypass.

Although the LM138 is stable with no output capacitors, like any feedback circuit, certain values of external capacitance can cause excessive ringing. This occurs with values between 500 pF and 5000 pF. A 1-µF solid tantalum (or 25-µF aluminum electrolytic) on the output swamps this effect and insures stability.

8.2.1.2.2 Load Regulation

The LM138 is capable of providing extremely good load regulation but a few precautions are needed to obtain maximum performance. The current set resistor connected between the adjustment terminal and the output terminal (usually 240 Ω) must be tied directly to the output of the regulator (case) rather than near the load. This eliminates line drops from appearing effectively in series with the reference and degrading regulation. For example, a 15-V regulator with 0.05-Ω resistance between the regulator and load has a load regulation due to line resistance of 0.05 Ω × I_L. If the set resistor is connected near the load, the effective line resistance is 0.05 Ω (1 + R2/R1) or in this case, 11.5 times worse.

Figure 17 shows the effect of resistance between the regulator and 240-Ω set resistor.

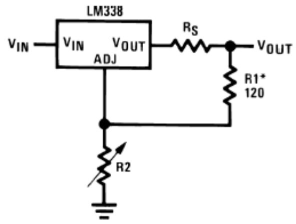


Figure 17. Regulator With Line Resistance in Output Lead

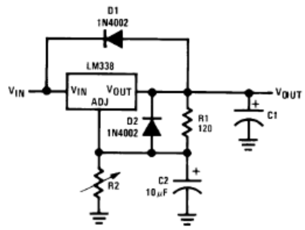
With the TO-3 package, it is easy to minimize the resistance from the case to the set resistor, by using 2 separate leads to the case. The ground of R2 can be returned near the ground of the load to provide remote ground sensing and improve load regulation.

8.2.1.2.3 Protection Diodes

When external capacitors are used with *any* IC regulator it is sometimes necessary to add protection diodes to prevent the capacitors from discharging through low current points into the regulator. Most 20-µF capacitors have low enough internal series resistance to deliver 20-A spikes when shorted. Although the surge is short, there is enough energy to damage parts of the IC.

When an output capacitor is connected to a regulator and the input is shorted, the output capacitor discharges into the output of the regulator. The discharge current depends on the value of the capacitor, the output voltage of the regulator, and the rate of decrease of V_{IN}. In the LM138 this discharge path is through a large junction that is able to sustain 25-A surge with no problem. This is not true of other types of positive regulators. For output capacitors of 100 µF or less at output of 15 V or less, there is no need to use diodes.

The bypass capacitor on the adjustment terminal can discharge through a low current junction. Discharge occurs when *either* the input or output is shorted. Internal to the LM138 is a 50-Ω resistor which limits the peak discharge current. No protection is needed for output voltages of 25-V or less and 10-µF capacitance. Figure 18 shows an LM138 with protection diodes included for use with outputs greater than 25 V and high values of output capacitance.



$$V_{OUT} = 1.25V\left(1 + \frac{R2}{R1}\right) + I_{ADJ}R2$$

D1 protects against C1
D2 protects against C2

Figure 18. Regulator With Protection Diodes

8.2.1.3 Application Curves

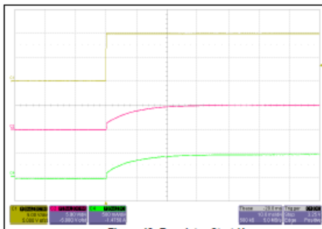


Figure 19. Regulator Start-Up

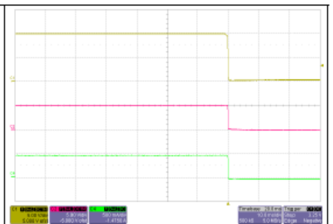


Figure 20. Regulator Shutdown

5. LM7805 Voltage Regulator Datasheet

Electrical Characteristics (LM7805)

Refer to the test circuits. -40°C < T_J < 125°C, I_O = 500mA, V_I = 10V, C_I = 0.1µF, unless otherwise specified.

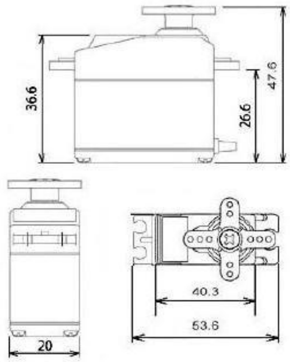
Symbol	Parameter	Conditions		Min.	Typ.	Max.	Unit
V_O	Output Voltage	T_J = +25°C		4.8	5.0	5.2	V
		5mA ≤ I_O ≤ 1A, P_O ≤ 15W, V_I = 7V to 20V		4.75	5.0	5.25	
Regline	Line Regulation(1)	T_J = +25°C	V_O = 7V to 25V	–	4.0	100	mV
			V_I = 8V to 12V	–	1.6	50.0	
Regload	Load Regulation(1)	T_J = +25°C	I_O = 5mA to 1.5A	–	9.0	100	mV
			I_O = 250mA to 750mA	–	4.0	50.0	
I_Q	Quiescent Current	T_J = +25°C		–	5.0	8.0	mA
ΔI_Q	Quiescent Current Change	I_O = 5mA to 1A		–	0.03	0.5	mA
		V_I = 7V to 25V		–	0.3	1.3	
$\Delta V_O/\Delta T$	Output Voltage Drift(2)	I_O = 5mA		–	-0.8	–	mV/°C
V_N	Output Noise Voltage	f = 10Hz to 100kHz, T_A = +25°C		–	42.0	–	µV/V_O
RR	Ripple Rejection(2)	f = 120Hz, V_O = 8V to 18V		62.0	73.0	–	dB
V_{DROP}	Dropout Voltage	I_O = 1A, T_J = +25°C		–	2.0	–	V
r_O	Output Resistance(2)	f = 1kHz		–	15.0	–	mΩ
I_{SC}	Short Circuit Current	V_I = 35V, T_A = +25°C		–	230	–	mA
I_{PK}	Peak Current(2)	T_J = +25°C		–	2.2	–	A

Notes:

1. Load and line regulation are specified at constant junction temperature. Changes in V_O due to heating effects must be taken into account separately. Pulse testing with low duty is used.
2. These parameters, although guaranteed, are not 100% tested in production.

6. MG996R Servo Motor Datasheet

MG996R High Torque
Metal Gear Dual Ball Bearing Servo



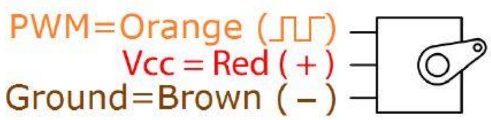
This High-Torque MG996R Digital Servo features metal gearing resulting in extra high 10kg stalling torque in a tiny package. The MG996R is essentially an upgraded version of the famous MG995 servo, and features upgraded shock-proofing and a redesigned PCB and IC control system that make it much more accurate than its predecessor. The gearing and motor have also been upgraded to improve dead bandwith and centering. The unit comes complete with 30cm wire and 3 pin 'S' type female header connector that fits most receivers, including Futaba, JR, GWS, Cirrus, Blue Bird, Blue Arrow, Corona, Berg, Spektrum and Hitec.

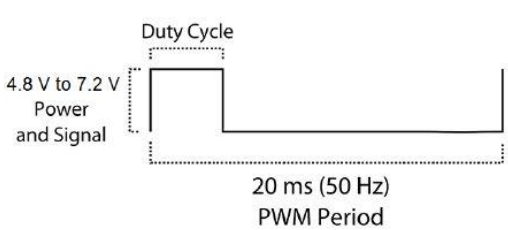
This high-torque standard servo can rotate approximately 120 degrees (60 in each direction). You can use any servo code, hardware or library to control these servos, so it's great for beginners who want to make stuff move without building a motor controller with feedback & gear box, especially since it will fit in small places. The MG996R Metal Gear Servo also comes with a selection of arms and hardware to get you set up nice and fast!

Specifications

- Weight: 55 g
- Dimension: 40.7 x 19.7 x 42.9 mm approx.
- Stall torque: 9.4 kgf·cm (4.8 V), 11 kgf·cm (6 V)
- Operating speed: 0.17 s/60º (4.8 V), 0.14 s/60º (6 V)

- Operating voltage: 4.8 V a 7.2 V
- Running Current 500 mA – 900 mA (6V)
- Stall Current 2.5 A (6V)
- Dead band width: 5 µs
- Stable and shock proof double ball bearing design
- Temperature range: 0 ºC – 55 ºC





7. SM-S4306R Servo Motor Datasheet

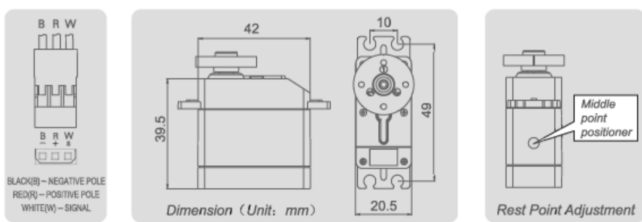
SPRINGRC®
www.springrc.com

43R Servo(360° Rotation) Specification

Thank you for choosing Spring Model's product

MODEL	TYPE	WEIGHT		4.8V			6V			DESCRIPTION	
		g	oz	SPEED	TORQUE		SPEED	TORQUE		GEAR	BEARING
				r/min	kg.cm	oz.in	r/min	kg.cm	oz.in		
SM-S4303R	Analog	44	1.55	60	3.3	45.8	70	4.8	66.7	1Metal Gear+ 4Plastic Gear	2
SM-S4306R		44	1.55	60	5.0	69.4	50	6.2	86.1	1Metal Gear+ 4Plastic Gear	2
SM-S4309R		60	2.12	58	7.9	109.7	49	8.7	120.8	Metal Gear	2
SM-S4315R		60	2.12	62	14.5	201.4	53	15.4	213.9	Metal Gear	2

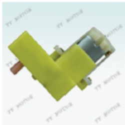
▲ 43R Robot series servo controled via analog signai(PWM),stopped via middle point positiner.
▲ Standard interface(like JR)with 30cm wire.
▲ Rotation and Rest Point Adjustment:when analog signal inputs,servo chooses orientation according to impulse width.when intermediatevalue of impluse width is above 1.5ms, servo is clockwise rotation,conversely,anticlockwise.Rest point need use slotted screwdriver to adjust the positioner carefully.Servo stopped rotation when the input signal is equivalent to impluse width.
▲ Please choose correct model for your application.
Caution: Torque over-loaded will damage the servo's mechanism.
▲ Keep the servo clean and away from dust, corrosive gas and humid air.
▲ Without further notification when some parameters slightly amend for improving quality.



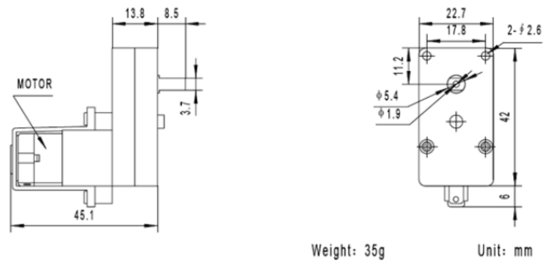
SPRING MODEL ELECTRONICS Co.,LTD. MADE IN CHINA XBBS4306R119 V1.0 RoHS CE

8. TGP01S-A130 Angled DC Motor Datasheet

Model：TGP02S-A130
Gear Motor



Applications
Electric Tools Robot Medical System Home Appliances Automobiles
Description
Reduction Ratio: 1/48、1/120、1/180、1/220、1/288



MODEL		VOLTAGE		NO LOAD		AT LOAD					STALL		
		PERATING RANGE	OMINAL	SPEED	URRENT	SPEED	URRENT	TORQUE		UTPUT	TORQUE		URRENT
			V	rpm	A	rpm	A	N.m	Kg.cm	w	N.m	Kg.cm	A
GP02S-A130	8100-220	3.0-12.0	3.0	50	0.20	40	0.60	0.120	1.233	0.60	.393	4.003	2.15
	4150-120	3.0-12.0	3.0	65	0.16	52	0.30	0.035	0.362	0.29	.200	2.055	0.87

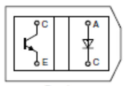
9. TCRT5000L IR Reflective Sensor Datasheet

TCRT5000, TCRT5000L

Vishay Semiconductors

Reflective Optical Sensor with Transistor Output

19156_2



Top view 19156_1

FEATURES

- Package type: leaded
- Detector type: phototransistor
- Dimensions (L x W x H in mm): 10.2 x 5.8 x 7
- Peak operating distance: 2.5 mm
- Operating range within > 20 % relative collector current: 0.2 mm to 15 mm
- Typical output current under test: I_C = 1 mA
- Daylight blocking filter
- Emitter wavelength: 950 nm
- Lead (Pb)-free soldering released
- Compliant to RoHS directive 2002/95/EC and in accordance to WEEE 2002/96/EC

RoHS COMPLIANT

DESCRIPTION

The TCRT5000 and TCRT5000L are reflective sensors which include an infrared emitter and phototransistor in a leaded package which blocks visible light. The package includes two mounting clips. TCRT5000L is the long lead version.

APPLICATIONS

- Position sensor for shaft encoder
- Detection of reflective material such as paper, IBM cards, magnetic tapes etc.
- Limit switch for mechanical motions in VCR
- General purpose - wherever the space is limited

PRODUCT SUMMARY

PART NUMBER	DISTANCE FOR MAXIMUM CTR_{rel} (1) (mm)	DISTANCE RANGE FOR RELATIVE I_{out} > 20 % (mm)	TYPICAL OUTPUT CURRENT UNDER TEST (2) (mA)	DAYLIGHT BLOCKING FILTER INTEGRATED
TCRT5000	2.5	0.2 to 15	1	Yes
TCRT5000L	2.5	0.2 to 15	1	Yes

Notes

(1) CTR: current transfere ratio, I_{out}/I_{in}

(2) Conditions like in table basic charactristics/sensors

ORDERING INFORMATION

ORDERING CODE	PACKAGING	VOLUME (1)	REMARKS
TCRT5000	Tube	MOQ: 4500 pcs, 50 pcs/tube	3.5 mm lead length
TCRT5000L	Tube	MOQ: 2400 pcs, 48 pcs/tube	15 mm lead length

Note

(1) MOQ: minimum order quantity

ABSOLUTE MAXIMUM RATINGS (1)

PARAMETER	TEST CONDITION	SYMBOL	VALUE	UNIT
INPUT (EMITTER)				
Reverse voltage		V_R	5	V
Forward current		I_F	60	mA
Forward surge current	$t_p \leq 10\ \mu s$	I_{FSM}	3	A
Power dissipation	$T_{amb} \leq 25\ °C$	P_V	100	mW
Junction temperature		T_j	100	°C

Document Number: 83760 For technical questions, contact: sensorstechsupport@vishay.com www.vishay.com
Rev. 1.7, 17-Aug-09 1

10. EK1254x5C IR Reflective Sensor Datasheet

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.

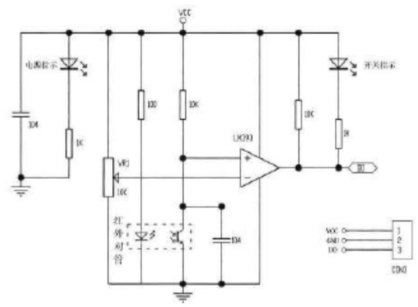
Specification

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)



11. Material Performance Index

Material	Process Used In	Chemical Resistance	Cost cents per cubic in.	Tensile Strength kpsi	Flexural Strength kpsi	Tensile Modulus kpsi	Flexural Modulus kpsi	Hard-ness-Rockwell	Impact Izod-Notch ft-lb/in	Density lb/in^3	Thermal Conductivity, 68 deg. F (BTU-ft/hrft^2 F)	Coef. of Thermal Expansion 68-212 deg. F (10^{-6} in/in/F)	Typical Uses
Thermoplastics													
ABS Medium Impact	Injection molding and extrusion.	High to acqueous acids, alkilis and salt.	3.4	6.3 - 8.0	9.9 - 11.8	340 - 400	350 - 400	80	24 - 4.0	0.038	0.96 - 2.16	3.2 - 4.8	Appliance parts; office, lawn and garden equipment; toys.
Acetal	Injection molding, extrusion, blow molding, and rotational molding.	Excellent to most. Poor for strong acids and alkalis.	6.4	10	14.1	520	410 - 450	80	1.5	0.052	1.56	4.5	Appliance parts, gears, bushings, auto and plumbing parts.
Acetal - 20 % glass			8.4	8.5	16.5	1300	800	110	0.8	0.056		2.0 - 4.5	
Nylon 6		Resists weak acids, alcohol, and common solvents.	5.9	5.5 - 13	10.0 -11.6	200 - 500			0.8 - 3.0	0.039	1.2	1.6 - 8.3	Bearings, gears, bushings, rod, tubing.
Nylon 6 - 30 % glass			7.7	22 - 26	26 - 34	1000 - 1450			2.3 - 3.0	0.05	1.2 - 1.7	1.2 - 3.0	General purpose parts requiring stiffness.
Nylon 6/6		Attacked by strong concentrations of mineral acids.	6.5	11.8		385 - 475	410		1.0	0.041	1.7	1.7	Bearings, gears, bushings, rod tubing.
Nylon 6/6 -. 30% glass			9.8		26 - 35	1400 - 2000	1300		2.2	0.05	1.5	1.5	Bearings, gears, bushings, rod tubing.
Polycar-bonate	Injection molding and extrusion.	Resists weak acids and alkilis, oils and grease.	6.7	8.5 - 9.0	12 - 14.2	325 - 340	310 - 350	63	12 - 18	0.04	1.35 - 1.41	3.75	Electrical parts, portable tool housings, lenses, sporting goods, impellers, and auto parts.
Polycar-bonate - 40% glass	Blow molding and thermo-forming.		10.4	23	27	1680	1400	50	2.5	0.055	1.53	0.93	
Thermosets													
Alkyd	Compression and transfer molding.	Resistant to weak acids. Unattacked by organic liquids (alcohol, fatty acids, and hydrocarbons).	5.5	7 - 8	19 - 20	1950	2500	70 - 75	2.2	0.079	4.2 - 7.2	1 - 3	Encapsulation of resistors, coils and small electronic parts, switches, relays, connectors, sockets, circuit breakers, parts for transformers, motor controllers, and auto ignition systems.
Alkyd and glass				5 - 9	12 - 17	2250	2500	70 - 80	8 - 12	0.073	2.4 - 3.6	1 - 3	
Epoxy and glass		Highly resis-tant to water and bases.		8 - 11	19 - 22		1500 - 2500	75 - 80	0.4 - 0.5	0.069	1.2 - 6	1 - 2	Electrical molding such as condensers, resistors, coils, etc.

Properties of Selected Plastics
(1 psi = 6895 Pa; 1 kpsi + 6.895 MPa; Example: 10 kpsi (6.895) = 69 MPa)

Material	Process Used In	Chemical Resistance	Cost cents per cubic in.	Tensile Strength kpsi	Flexural Strength kpsi	Tensile Modulus kpsi	Flexural Modulus kpsi	Hard-ness-Rockwell	Impact Izod-Notch ft-lb/in	Density lb/in^3	Thermal Conductivity, 68 deg. F (BTU-ft/hrft^2·F)	Coef. of Thermal Expansion 68-212 deg. F (10^{-6} in/in/F)	Typical Uses
						Thermoplastics							
ABS Medium Impact	Injection molding and extrusion.	High to acqueous acids, alkilis and salt.	3.4	6.3 - 8.0	9.9 - 11.8	340 - 400	350 - 400	80	24 - 4.0	0.038	0.96 - 2.16	3.2 - 4.8	Appliance parts; office, lawn and garden equipment; toys.
Acetal	Injection molding, extrusion, blow molding, and rotational molding.	Excellent to most. Poor for strong acids and alkalis.	6.4	10	14.1	520	410 - 450	80	1.5	0.052	1.56	4.5	Appliance parts, gears, bushings, auto and plumbing parts.
Acetal - 20 % glass			8.4	8.5	16.5	1300	800	110	0.8	0.056		2.0 - 4.5	
Nylon 6		Resists weak acids, alcohol, and common solvents.	5.9	5.5 - 13	10.0 -11.6	200 - 500			0.8 - 3.0	0.039	1.2	1.6 - 8.3	Bearings, gears, bushings, rod, tubing.
Nylon 6 - 30 % glass			7.7	22 - 26	26 - 34	1000 - 1450			2.3 - 3.0	0.05	1.2 - 1.7	1.2 - 3.0	General purpose parts requiring stiffness.
Nylon 6/6		Attacked by strong concentrations of mineral acids.	6.5	11.8		385 - 475	410		1.0	0.041	1.7	1.7	Bearings, gears, bushings, rod tubing.
Nylon 6/6 - 30% glass			9.8		26 - 35	1400 - 2000	1300		2.2	0.05	1.5	1.5	Bearings, gears, bushings, rod tubing.
Polycar-bonate	Injection molding and extrusion.	Resists weak acids and alkilis, oils and grease.	6.7	8.5 - 9.0	12 - 14.2	325 - 340	310 - 350	63	12 - 18	0.04	1.35 - 1.41	3.75	Electrical parts, portable tool housings, lenses, sporting goods, impellers, and auto parts.
Polycar-bonate - 40% glass	Blow molding and thermo-forming.		10.4	23	27	1680	1400	50	2.5	0.055	1.53	0.93	
						Thermosets							
Alkyd	Compression and transfer molding.	Resistant to weak acids. Unattacked by organic liquids (alcohol, fatty acids, and hydrocarbons).	5.5	7 - 8	19 - 20	1950	2500	70 - 75	2.2	0.079	4.2 - 7.2	1 - 3	Encapsulation of resistors, coils and small electronic parts, switches, relays, connectors, sockets, circuit breakers, parts for transformers, motor controllers, and auto ignition systems.
Alkyd and glass				5 - 9	12 - 17	2250	2500	70 - 80	8 - 12	0.073	2.4 - 3.6	1 - 3	
Epoxy and glass		Highly resis-tant to water and bases.		8 - 11	19 - 22		1500 - 2500	75 - 80	0.4 - 0.5	0.069	1.2 - 6	1 - 2	Electrical molding such as condensers, resistors, coils, etc.

Properties of Selected Plastics
(1 psi = 6895 Pa; 1 kpsi + 6.895 MPa; Example: 10 kpsi (6.895) = 69 MPa)

Typical Properties of Some Engineering Materials

Material	Density (_) (g/cc)	Tensile Modulus (E) (GPa)	Tensile Strength (_) (GPa)	Specific Modulus (E/_)	Specific Strength (_/_)	Max. Service Temp. (°C)
Metals						
Cast iron, grade 20	7.0	100	0.14	14.3	0.02	230-300
Steel, AISI 1045 hot rolled	7.8	205	0.57	26.3	0.073	500-650
Aluminum 2024-T4	2.7	73	0.45	27.0	0.17	150-250
Aluminum 6061-T6	2.7	69	0.27	25.5	0.10	150-250
Plastics						
Nylon 6/6	1.15	2.9	0.082	2.52	0.071	75-100
Polypropylene	0.9	1.4	0.033	1.55	0.037	50-80
Epoxy	1.25	3.5	0.069	2.8	0.055	80-215
Phenolic	1.35	3.0	0.006	2.22	0.004	70-120
Ceramics						
Alumina	3.8	350	0.17	92.1	0.045	1425-1540
MgO	3.6	205	0.06	56.9	0.017	900-1000
Short fiber composites						
Glass-filled epoxy (35%)	1.90	25	0.30	8.26	0.16	80-200
Glass-filled polyester (35%)	2.00	15.7	0.13	7.25	0.065	80-125
Glass-filled nylon (35%)	1.62	14.5	0.20	8.95	0.12	75-110
Glass-filled nylon (60%)	1.95	21.8	0.29	11.18	0.149	75-110
Unidirectional composites						
S-glass/epoxy (45%)	1.81	39.5	0.87	21.8	0.48	80-215
Carbon/epoxy (61%)	1.59	142	1.73	89.3	1.08	80-215
Kevlar/epoxy (53%)	1.35	63.6	1.1	47.1	0.81	80-215

12. LCD datasheet

HD44780U (LCD-II)

(Dot Matrix Liquid Crystal Display Controller/Driver)

HITACHI

ADE-207-272(Z)
'99.9
Rev. 0.0

Description

The HD44780U dot-matrix liquid crystal display controller and driver LSI displays alphanumerics, Japanese kana characters, and symbols. It can be configured to drive a dot-matrix liquid crystal display under the control of a 4- or 8-bit microprocessor. Since all the functions such as display RAM, character generator, and liquid crystal driver, required for driving a dot-matrix liquid crystal display are internally provided on one chip, a minimal system can be interfaced with this controller/driver.

A single HD44780U can display up to one 8-character line or two 8-character lines.

The HD44780U has pin function compatibility with the HD44780S which allows the user to easily replace an LCD-II with an HD44780U. The HD44780U character generator ROM is extended to generate 208 5 × 8 dot character fonts and 32 5 × 10 dot character fonts for a total of 240 different character fonts.

The low power supply (2.7V to 5.5V) of the HD44780U is suitable for any portable battery-driven product requiring low power dissipation.

Features

- 5 × 8 and 5 × 10 dot matrix possible
- Low power operation support:
  — 2.7 to 5.5V
- Wide range of liquid crystal display driver power
  — 3.0 to 11V
- Liquid crystal drive waveform
  — A (One line frequency AC waveform)
- Correspond to high speed MPU bus interface
  — 2 MHz (when V_{CC} = 5V)
- 4-bit or 8-bit MPU interface enabled
- 80 × 8-bit display RAM (80 characters max.)
- 9,920-bit character generator ROM for a total of 240 character fonts
  — 208 character fonts (5 × 8 dot)
  — 32 character fonts (5 × 10 dot)

13. RTC datasheet

CLOCK AND CALENDAR

The time and calendar information is obtained by reading the appropriate register bytes. Table 2 shows the RTC registers. The time and calendar are set or initialized by writing the appropriate register bytes. The contents of the time and calendar registers are in the BCD format. The day-of-week register increments at midnight. Values that correspond to the day of week are user-defined but must be sequential (i.e., if 1 equals Sunday, then 2 equals Monday, and so on.) Illogical time and date entries result in undefined operation. Bit 7 of Register 0 is the clock halt (CH) bit. When this bit is set to 1, the oscillator is disabled. When cleared to 0, the oscillator is enabled. On first application of power to the device the time and date registers are typically reset to 01/01/00 01 00:00:00 (MM/DD/YY DOW HH:MM:SS). The CH bit in the seconds register will be set to a 1. The clock can be halted whenever the timekeeping functions are not required, which minimizes current (I_{BATDR}).

The DS1307 can be run in either 12-hour or 24-hour mode. Bit 6 of the hours register is defined as the 12-hour or 24-hour mode-select bit. When high, the 12-hour mode is selected. In the 12-hour mode, bit 5 is the AM/PM bit with logic high being PM. In the 24-hour mode, bit 5 is the second 10-hour bit (20 to 23 hours). The hours value must be re-entered whenever the 12/24-hour mode bit is changed.

When reading or writing the time and date registers, secondary (user) buffers are used to prevent errors when the internal registers update. When reading the time and date registers, the user buffers are synchronized to the internal registers on any I^2C START. The time information is read from these secondary registers while the clock continues to run. This eliminates the need to re-read the registers in case the internal registers update during a read. The divider chain is reset whenever the seconds register is written. Write transfers occur on the I^2C acknowledge from the DS1307. Once the divider chain is reset, to avoid rollover issues, the remaining time and date registers must be written within one second.

Table 2. Timekeeper Registers

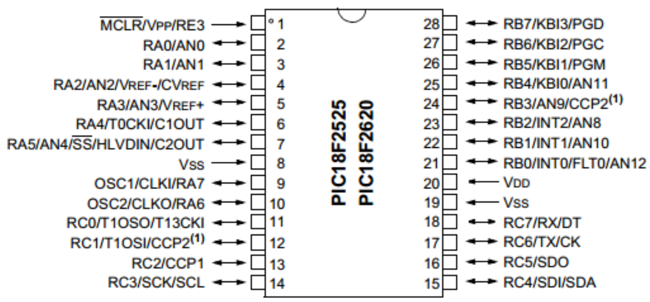
ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	FUNCTION	RANGE
00h	CH	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12 24	10 Hour PM/AM	10 Hour	Hours				Hours	1–12 +AM/PM 00–23
03h	0	0	0	0	0	DAY			Day	01–07
04h	0	0	10 Date		Date				Date	01–31
05h	0	0	0	10 Month	Month				Month	01–12
06h	10 Year				Year				Year	00–99
07h	OUT	0	0	SQWE	0	0	RS1	RS0	Control	—
08h–3Fh									RAM 56 x 8	00h–FFh

0 = Always reads back as 0.

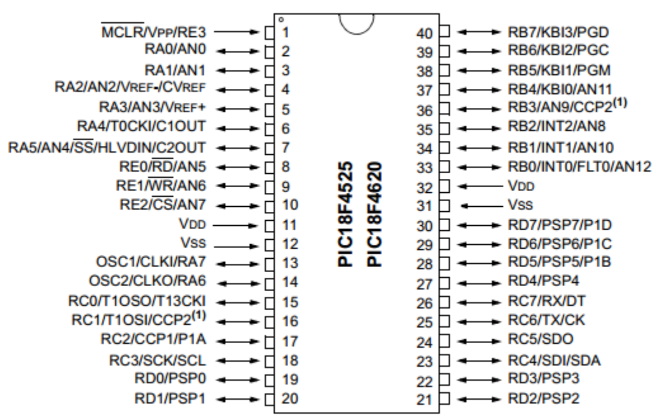
PIC18F2525/2620/4525/4620

Pin Diagrams

28-Pin SPDIP, SOIC



40-Pin PDIP



Note 1: RB3 is the alternate pin for CCP2 multiplexing.

Appendix D: Sections of Proposal

5. SPECIFICATION

This section presents the proposed design in detail through the three different subsystems: electromechanical, circuit and microcontroller. The overall design will be presented in the design overview section, which includes the general description of the design choice and final model.

The design is likely to change during the construction process and testing process, but the overall design should remain as what is described in this report.

5.1 Description Overview

The cone dispensing machine will be structured in a hybrid arrangement, with the cone dispensing mechanism independent of the driving and sensing systems. The cone holder is mounted at the back of the cart. The advantage of using this structure is to achieve system separation as both systems do not necessarily interact with each other (the machine will stop while dispensing the cone).

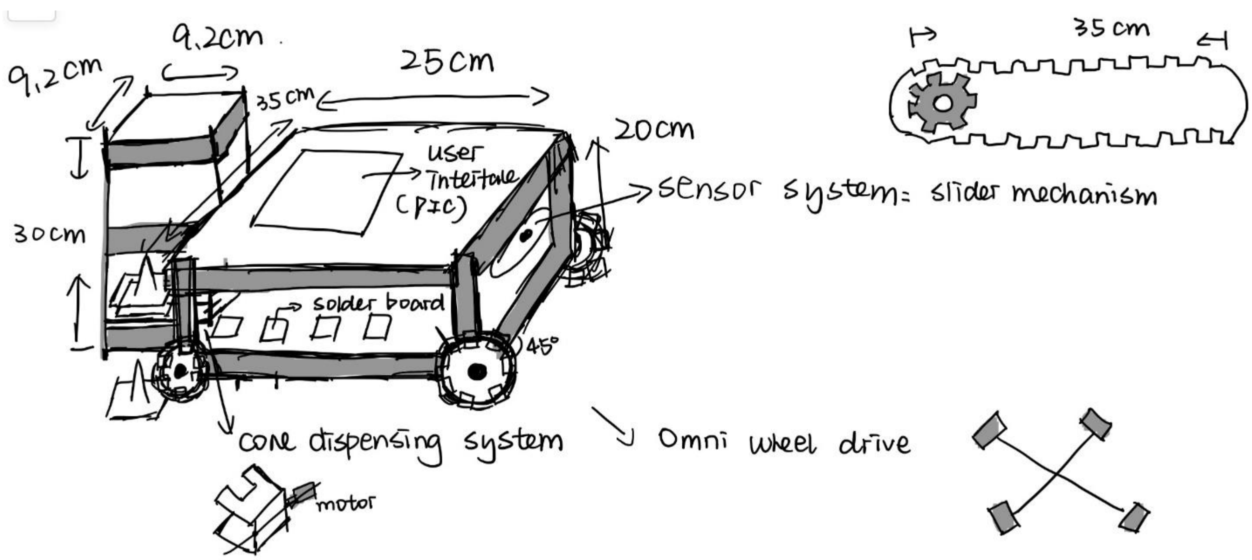


Figure 12: Illustration of Proposed Design

5.1.1 Size

<u>Measurements:</u>

Cone: 9cm*9cm*9cm (5cm diameter circle in the middle)

Distance Between Two Stacked Cones: $1cm \pm 1mm$

PIC board: 17.5cm $*$ 18.7cm $*$ 1.3cm $\pm\ 1mm$

According to the constraints that the dimension must be within 50cm*50cm*50cm, with also the width of the lane is 25cm, the width of the design can be ranging from 25cm to 50cm if the design wants to achieve the purpose of covering the lane. Height of the cone dispensing holder must be higher than 30cm (shown below in calculation (1)) measuring from the ground in order to hold 12 cones in place. The length of the design should cover the length of a PIC board as it is designed to be mount on the top of the robot body, in which the length of the robot can be ranging from 17.5cm to 50 cm. The width of the cone holder is considered ranging from 9.0 cm to 9.5 cm, as during the testing of our prototype, the length within this range provides enough space for the cone to freely drop while not move to other directions.

$H = 12 * 1 + 2 * 9 = 30\ cm \pm 1mm$ (1)

With the hybrid arrangement, the overall machine is chosen within the dimension of 34.2cm* 35cm*30cm, which consists of a rectangular holder of 9.2cm*9.2cm*30cm and a rectangular shaped body of 25cm*35cm*20cm. The length of the holder is chosen to be the average value between 9cm and 9.5cm.

Both the dispensing holder and the body will be constructed using hollow aluminum sheet. The cone dispensing holder will have four aluminum stands at each corner and surrounded with aluminum stripes (approximately 3 stripes distributed in an equal distance).

5.1.2 Cone Dispensing System

The cone dispensing system uses the alternating doors mechanism described in conceptualization section, which is constructed using a wooden board with a shaft connected to one servo motor.

5.1.3 Sensor Moving System

The sensor moving system utilizes the slider mechanism described in conceptualization section. The sensor will achieve its movement by moving with the slider while the gear is rotating.

5.1.4 Driving System

The drive system chosen for our machine is the omni wheel drive with 4 omni wheels, which is described in the conceptualization section.

5.1.5 Operation

The machine is positioned at the beginning line of the lane and will operate after the setup on the user interface. It will then go along the straight lane until it detects a crack or hole. Once a hole or crack is detected, the machine will stop and move its sensor to measure its length to determine the placement of the cones. Then, the machine will move horizontally to place either one cone for a hole or two cones for a crack.

5.2 Subsystem Decomposition

5.2.1 Electromechanical

5.2.1.1 Actuator Selection

5.2.1.1.1 Driving System Motors

The motor must work continuously excluding the operation of deployment of cones. The torque must be high to carry weight.

To quantitively analyze the task performed by the motor, the following calculations are performed:

5.2.1.1.1.1 Moment of Inertia

The wheel is made of plastic with density of 0.92g/m^3

$$I = {}^{1}/_{2}\, mr^2 = \frac{1}{2} \times 0.00092 \times pi \times 0.04^2 \times 0.02 \times 0.04^2 = 7.40 \times 10^{-11} kgm^2$$

5.2.1.1.1.2 Angular velocity

Consider the total operation time is 3 minutes and assume that the remaining 20 seconds is for returning.

$$v = 4 \div 20 = 0.2\, m/s$$

$$P = 8 \times pi = 25 \frac{cm}{rotation} = 0.25\, m/rotation$$

$$0.2 \frac{m}{s} \div 0.25 \frac{m}{rotation} \times 60\, s/\min = 48 \text{ rpm}$$

$$\omega = \omega_{rpm} \times \left(\frac{2pi}{60}\right) = 5.02\, rad/s$$

5.2.1.1.1.3 Angular Acceleration

Assume the machine needs 2s to accelerate

$$\alpha = \frac{\Delta\omega}{\Delta t} = \frac{5.02}{2} = 2.5\, rad/s$$

5.2.1.1.1.4 Torque

$$\tau = I\alpha = 7.40 \times 10^{-11} \times 2.5 = 1.85 \times 10^{-10} N/m$$

5.2.1.1.1.5 Power

$P = \tau \times \omega_{rad} = 9.29 \times 10^{-10} w$

Apply a factor of safety of 2 to account to sources of errors:

$P_{safe} = 1.857 \times 10^{-9} w$

Based on the calculations of the torque and power needed, the section of DC motor TGP01S-A13014150-120 is made according to datasheet in Appendix C-3.

5.2.1.1.2 Sensor Moving Mechanism Motor

Assumption is made that the motor rotates approximately 30 revolutions per minute.

Regard the gear as an aluminum cylinder with r=1cm

The density of aluminum is 2.7 g/m^3

5.2.1.1.2.1 Moment of Inertia

$$I = \frac{1}{2} mr^2 = \frac{1}{2} \times 0.0027 \times pi \times 0.01^2 \times 0.002 \times 0.01^2 = 8.48 \times 10^{-14} kgm^2$$

5.2.1.1.2.2 Angular Acceleration

$$\omega = \omega_{rpm} \ \times \left(\frac{2pi}{60}\right) = 3.14\, rad/s$$

Assume the acceleration time is approximately 3 s

$$\alpha = \frac{\Delta\omega}{\Delta t} = \frac{3.14}{3} = 1\, rad/m^2$$

5.2.1.1.2.3 Torque

$$\tau = I\alpha = 8.48 \times 10^{-14} \times 1 = 8.48 \times 10^{-14} Nm$$

5.2.1.1.2.4 Power

$$P = \tau \times \omega_{rad} = 2.66 \times 10^{-13} w$$

Based on the calculated results and data sheet in Appendix C-3, the DC motor TGP01S-A13014150-120 is selected for the sensor moving mechanism.

5.2.1.1.3 Cone Dispensing System Motor

The board is of dimension 9.2 cm* 6cm*0.8cm, the wood has a density of 650 kg/m^3

5.2.1.1.3.1 Moment of Inertia

$$I = \frac{1}{12} m(D^2 + H^2) = \frac{1}{12} \times 650 \times (0.092 \times 0.06 \times 0.008) \times (0.0092^2 + 0.06^2)$$
$$= 8.814 \times 10^{-6} kgm^2$$

5.2.1.1.3.2 Torque

$\tau = I\alpha = 8.814 \times 10^{-6} \times 1 = 8.814 \times 10^{-6} Nm$

5.2.1.1.3.3 Power

$P = \tau \times \omega_{rad} = 2.77 \times 10^{-5} w$

According to the calculated torque and power, a choice of MG996R high torque servo motor is selected. (Details of the motor in Appendix C-4)

5.2.1.2 Drive System

The proposed driving system used in our design is holonomic drive with four omni wheels constructed 45 degree clockwise with respect to the general orientation with wheels tangential at the two sides. The following figures show how the omni wheels are controlled through microcontroller signals to achieve all directional movement.

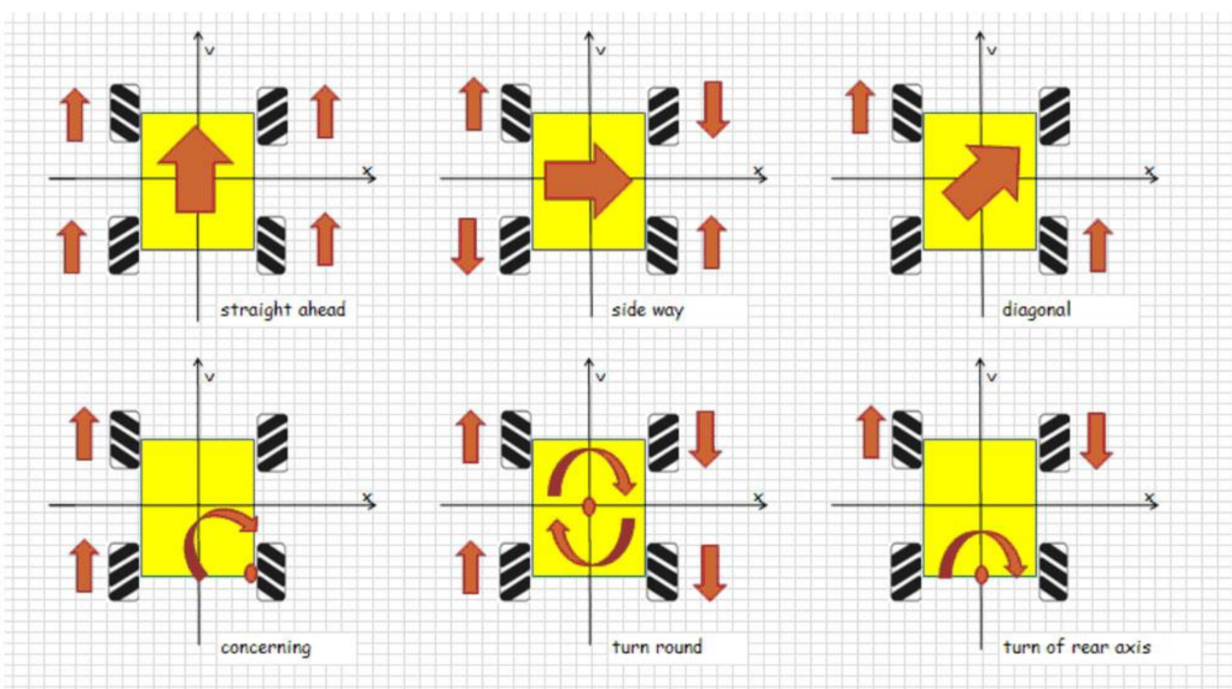


Figure 13: Directional Control of Mecanum Wheels Mounted Parallel to the Main Body

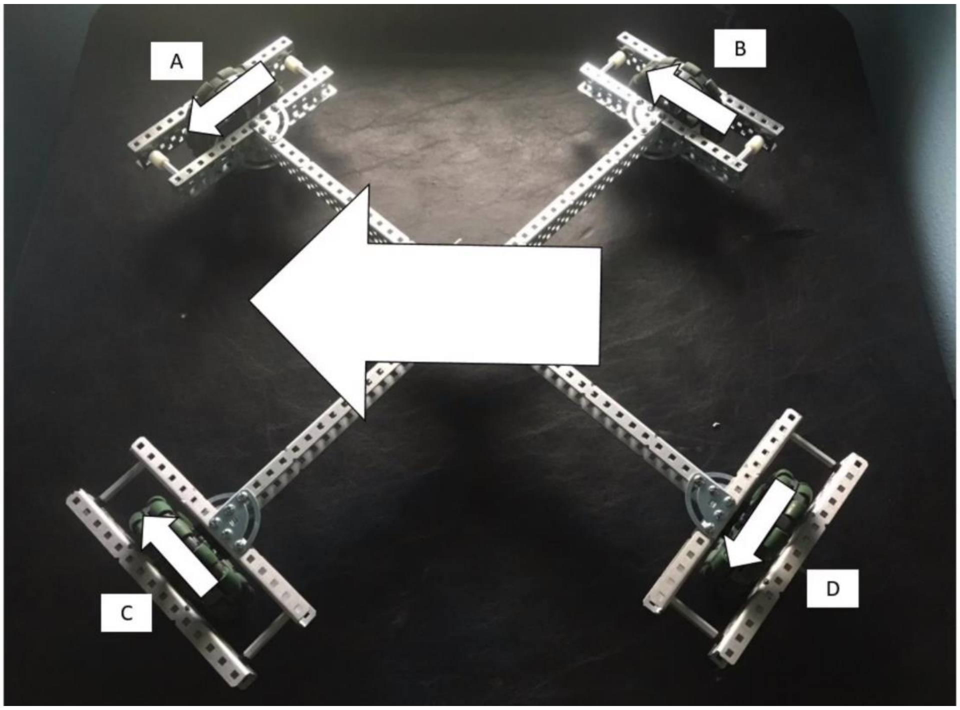


Figure 14: Omni Wheels Mounted at 45 Degree Angles to the Main Body Follows the Same Directional Control Protocols as the Mecanum Wheels.

5.2.1.3 Materials

According to the materials index (See Appendix C-5), several materials are selected for different mechanisms through careful consideration:

- Structure: aluminum (robust, light)
- Base: aluminum (robust, light)
- Cone Dispensing Mechanism: wood (easy for construction)
- Wire: copper (conductor)

5.2.2 Circuit Design

5.2.2.1 Driving Motors

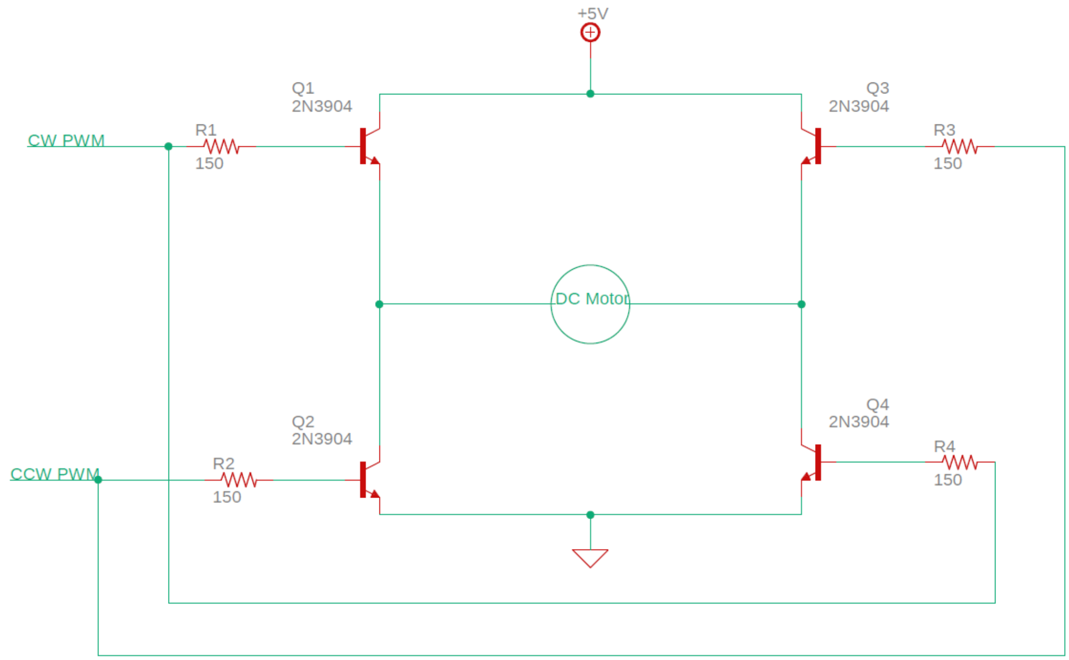


Figure 15: H – Bridge Motor Driver Circuit

The driving DC motors will be controlled using a simple h-bridge circuit as shown in Figure 15. This circuit will provide the necessary directional control of the driving motors based on signals given by the microcontroller in the form of PWM. When a signal is given to transistors Q1 and Q4, the motor will be powered to rotate in the clockwise direction. Similarly, when a signal is given to transistors Q2 and Q3, the motor will rotate in the counterclockwise direction. Furthermore, when none of the transistors are receiving a signal, the motor will be turned off. Since the rotation of diagonal driving wheels will be the same, this means that the PWM signal for the two diagonal wheels can be shared. As a result, the PWM signals indicated in Figure 15 will also be connected to another motor with a similar h-bridge circuit, which reduces the amount of microcontroller pins needed by half. This circuit can also be used for speed control via the duty cycle. A higher duty cycle would result in a higher speed because the motors are exposed to the current flow for a longer period of time.

5.2.2.2 Sensor Rack Motor

Since the motor used to provide translational movement for the IR sensors will also be a DC motor. The same h-bridge circuit can be applied to the DC motor controlling the movement of the sensor rack.

5.2.2.3 Cone Dispensing Mechanism Motor

Figure 16: Connection Diagram of the MG996R Servo Motor

As shown in Figure 16, the connection to the servo motor is does not require an external driver board. The PWM signal from the PIC board can be directly sent to the servo motor to control its speed and direction, while the VCC and Ground pins are used to power the motor through the power supply.

5.2.2.4 IR Sensors

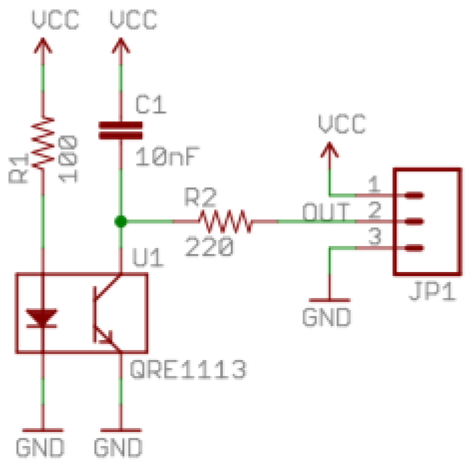


Figure 17: Circuit Diagram for QRE1113 IR Sensors

The sensors for crack and hole detection as well as trajectory maintenance will be IR sensors. This type of sensor is the best choice for our project because we do not need detect physical objects, which eliminates the choice of most proximity sensors. The only remaining choice is using light sensors, and out of all of the light sensors in a reasonable price range within the budgeting limit, IR sensors have the best price to performance ratio. The IR sensors that will be used compose of IR LEDs and IR transistor pairs as shown in Figure 17. The IR LED transmits a beam of IR light into one direction, and the IR transistor will have a different discharge response based on the intensity of the IR light bounced backed after contacting a surface. IR light is also less affected by ambient light conditions, which makes calibrations less troublesome. This setup is perfect for crack and hole detection between black surfaces like the tape that will be used to represent cracks and holes reflect a minimum amount of IR light. As a result, the IR sensor will have a very sensitive response in differentiating between the tape and the floor.

5.2.2.5 Rotary Encoders

Another type of sensors that will be used are rotary encoders. Rotary encoders are sensors that can be added onto the shaft of DC motors to record distance travelled. However, these encoders can only output the relative position of the wheel at any given time. As a result, to extract the distance information from these encoders, the arclength between two rotational positions of the encoder must be obtained through the datasheet. Using this arclength, it is then possible to calculate the distance travelled through the PIC board by multiplying the arclength by the number of positions travelled. A rotary encoder will be needed on one of the driving DC motors to record the distance travelled both along the path as well as horizontally when dispensing the cones. Another rotary encoder will be needed on the DC motor controlling the sensor moving mechanism to ensure accurate IR sensor positioning.

5.2.2.6 Power Supply

Figure 18: The Turnigy 2200mAh 2S 25C Lipo Battery Pack

Since an on-board power supply is required, the power supply that will be used is the Turnigy 2200mAh 2S 25C Lipo Pack. It provides a good capacity of 2200mAh, but the voltage that it comes in is at 7.7V. This means that voltage regulators are required for each circuit board to protect the components as most of them take less than 5V in potential.

5.2.3 Microcontroller

5.2.3.1 Microcontroller Selection

The Peripheral Interface Controller (PIC) microcontroller from Microchip Technology Inc. is used in the design as recommended by the instructor (as well as the client). The advantages of PIC microcontrollers are their fast operation, low power, low cost and ease of programming [12]. Figure 18 shows a picture of the PIC model we use.

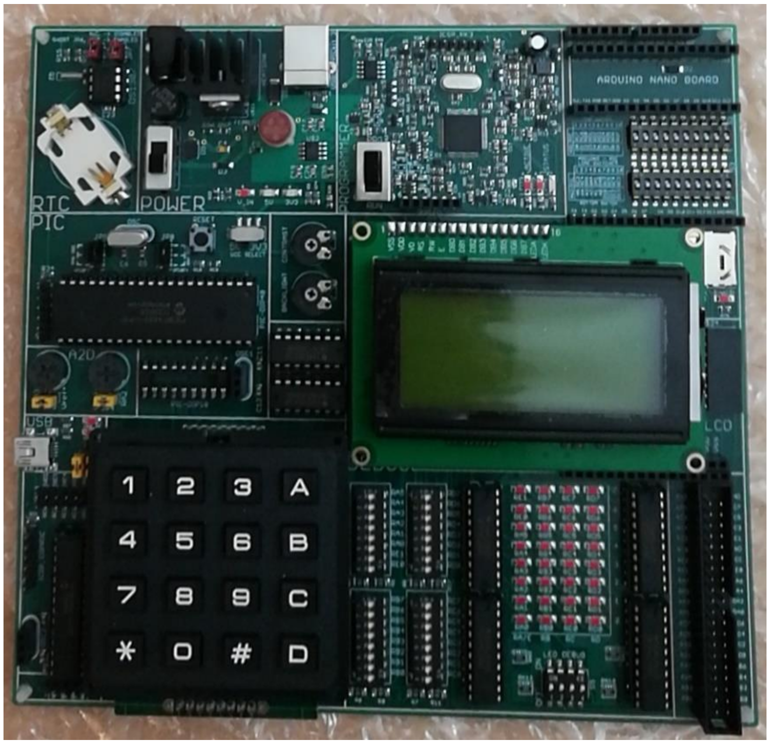


Figure 19: Picture of PIC Microcontroller

5.3.3.2 Pin Assignments

Table 5: Microcontroller Pin Assignments

Pin name	In/Out	Analog(A) or Digital(D)	Description
RD0:RD7	Out	D	LCD Display

RB4:RB7	In	D	Keypad Input
RB1	In	D	Keypad Interrupt
RA0	Out	D	CW signal for first set of diagonal driving DC motors
RA1	Out	D	CCW signal for first set of diagonal driving DC motors
RA5	Out	D	CW signal for second set of diagonal driving DC motors
RE0	Out	D	CCW signal for second set of diagonal driving DC motors
RE1	In	D	Signal from rotary encoder on one of the driving DC motors
RE2	Out	D	Directional signal for cone dispensing mechanism servo motor
RC0	Out	D	CW signal for sensor moving mechanism DC motor
RC1	Out	D	CCW signal for sensor moving mechanism DC motor
RC2	In	D	Signal from rotary encoder on the sensor moving mechanism DC motor
RC6	In	D	Signal from crack/hole detection IR sensor
RC5	In	D	Signal from first lane following IR sensor
RC7	In	D	Signal from second lane following IR sensor
RC3, RC4	In	D	RTC

*CW denotes clockwise, CCW denotes counterclockwise

5.2.3.3 Microcontroller Flowchart and Pseudo Codes

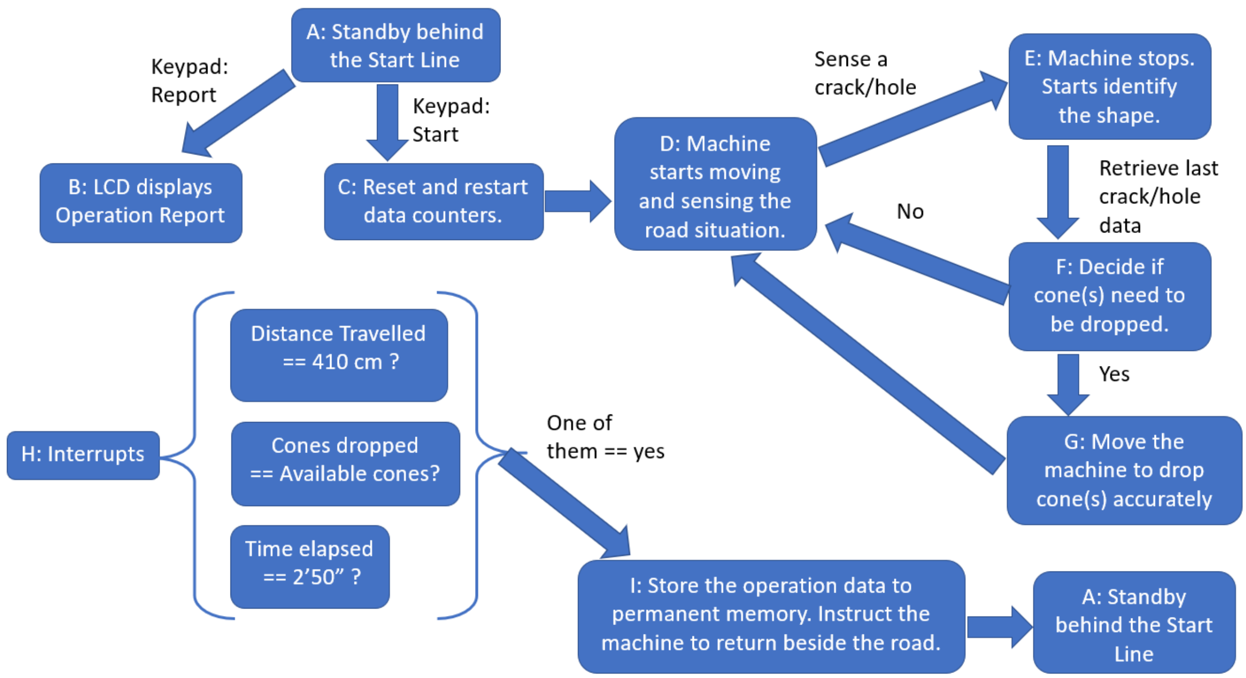


Figure 20: Microcontroller Flowchart

Corresponding to the processes A to I in the main flowchart, pseudo codes for each function are proposed here.

Initial Setup: Configure input and output pins. Enable interrupts. Assign data address.

A: Set outputs to moving parts to zero. Instruct LCD to the standby menu display.

B: if-statements to change LCD displays based on Keypad pins inputs.

C: reset the data in corresponding address for: elapsed time, distances travelled, number of cones dropped, number of holes/cracks detected etc. Start time and distance counters.

D: Set output to wheels and sensors to high.

E: Clear the interrupt. Set output to wheels to zero. Set the output to communication signal to high. A while-loop to instruct the sensor move left and right until both edges of the crack/hole are located. Identify the shape based on the distance travelled by the sensor. Retrieve from memory the shape and location of the last crack/hole. Accumulate the relevant counters.

F: If-statements to compute a 0/1/2 output for cones dispensing based on previous data. Accumulate the relevant counters.

G: Compute the required movement of the machine to drop the cones. Instruct the machine to dispense cones. Accumulate the relevant counters.

H: (in the initial setup) set internal interrupts for the three listed conditions.

I: Clear the interrupt. Move the necessary data to permanent memory. Compute the distance needed to travel to return to the Start line. Instruct to machine to move correspondingly.

5.2.3.4 User Interface

The keypad and LCD module on PIC board serve as the main user interface. If an emergence occurs, the STOP switch on the shell of the machine can cut off the power supply and stop all moving parts. Figure xx shows the appearance of the keypad and LCD. LCD shows 4 lines and 64 characters in total.

The keypad can be used when the machine is at rest and in standby mode behind the start line. At that time, LCD shows "**Completed. Press A: Operation Report / D:Restart**" in the first three lines while the fourth line **rotates** to display the real-time date and time (e.g. "**Date 28/Jan/2019 Time 13:57:34**" . If A is pressed, LCD will display the new instruction and the operation report. The operation report contains the numbers and locations of holes and cracks detected respectively, the number of cones deployed, and overall operation time. If exit is commanded, LCD will show the initial standby page.

Example:

*:Exit #: ->	0: <- #: ->	0: <- #: ->
Use 12 Cones	4 Holes: 123 252	8Cracks:24 75 83
Duration 3'45''	364 381	99 156 189 224 300

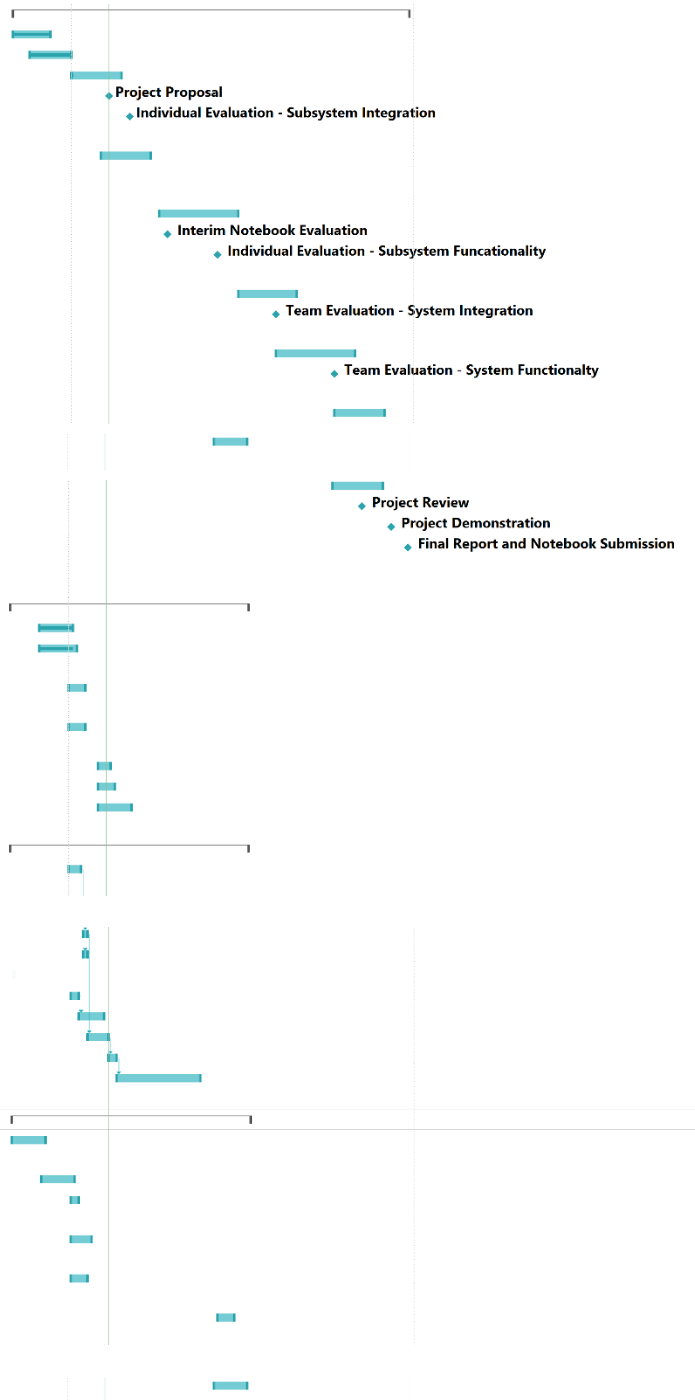
Figure 21: Page 1, 2, and 3 for Operation Report.

6. Project Management

6.1 Task Assignment

6.1.1 Gantt Chart

Task Name	Duration	Start	Finish
<AER201 Cone Placement>	**69 days**	**Mon 1/7/19**	**Fri 4/12/19**
Brainstorming	7 days	Mon 1/7/19	Tue 1/15/19
Project Planning	7 days	Fri 1/11/19	Sun 1/20/19
Subsystem Fabrication	10 days	Mon 1/21/19	Fri 2/1/19
Project Proposal	0 days	Wed 1/30/19	Wed 1/30/19
Individual Evaluation - Subsystem Integration	0 days	Mon 2/4/19	Mon 2/4/19
Subsystem Integration	10 days	Mon 1/28/19	Fri 2/8/19
Subsystem Functionality	15 days	Mon 2/11/19	Fri 3/1/19
Interim Notebook Evaluation	0 days	Wed 2/13/19	Wed 2/13/19
Individual Evaluation - Subsystem Funcationality	0 days	Mon 2/25/19	Mon 2/25/19
System Integration	11 days	Sat 3/2/19	Fri 3/15/19
Team Evaluation - System Integration	0 days	Mon 3/11/19	Mon 3/11/19
System Functionality	15 days	Mon 3/11/19	Fri 3/29/19
Team Evaluation - System Functionalty	0 days	Mon 3/25/19	Mon 3/25/19
System Debugging	10 days	Mon 3/25/19	Fri 4/5/19
Subsystem Calibration and Integration	6 days	Mon 2/25/19	Mon 3/4/19
System Debugging	10 days	Mon 3/25/19	Fri 4/5/19
Project Review	0 days	Mon 4/1/19	Mon 4/1/19
Project Demonstration	0 days	Mon 4/8/19	Mon 4/8/19
Final Report and Notebook Submission	0 days	Fri 4/12/19	Fri 4/12/19
Circuits	**41 days**	**Mon 1/7/19**	**Mon 3/4/19**
Sensor Selection	6 days	Mon 1/14/19	Mon 1/21/19
Motor Driver Circuit Experimentation	7 days	Mon 1/14/19	Tue 1/22/19
Motor Driver Circuit Fabrication	4 days	Mon 1/21/19	Thu 1/24/19
Sensor Circuit Experimentation	4 days	Mon 1/21/19	Thu 1/24/19
Sensor Circuit Fabrication	3 days	Mon 1/28/19	Wed 1/30/19
Power Supply Selection	4 days	Mon 1/28/19	Thu 1/31/19
Debugging	6 days	Mon 1/28/19	Mon 2/4/19
Microcontroller	**41 days**	**Mon 1/7/19**	**Mon 3/4/19**
run and play with sample code	3 days	Mon 1/21/19	Wed 1/23/19
user interface flowchart	1 day	Thu 1/24/19	Thu 1/24/19
pins propertities study	1 day	Thu 1/24/19	Thu 1/24/19
pins assignment	1 day		
overall flowchart	2 days	Mon 1/21/19	Tue 1/22/19
subroutine flow chart	4 days	Wed 1/23/19	Mon 1/28/19
user interface design	3 days	Fri 1/25/19	Tue 1/29/19
user interface peusdo code	2 days	Wed 1/30/19	Thu 1/31/19
actual code writing	14 days	Fri 2/1/19	Wed 2/20/19
Electromechanical	**41 days**	**Mon 1/7/19**	**Mon 3/4/19**
Textbook Reading Electromechanical part	6 days	Mon 1/7/19	Mon 1/14/19
Project Skeching	6 days	Mon 1/14/19	Mon 1/21/19
Hardware/Materials Shopping	2 days	Mon 1/21/19	Tue 1/22/19
Constructing Driving System	5 days	Mon 1/21/19	Fri 1/25/19
Construcing Cone Dispensing Mechanism	4 days	Mon 1/21/19	Thu 1/24/19
Constructing Sensor Mover Mechnism	4 days	Mon 2/25/19	Thu 2/28/19
Subsystem Calibration and Integration	6 days	Mon 2/25/19	Mon 3/4/19



6.1.2 PERT

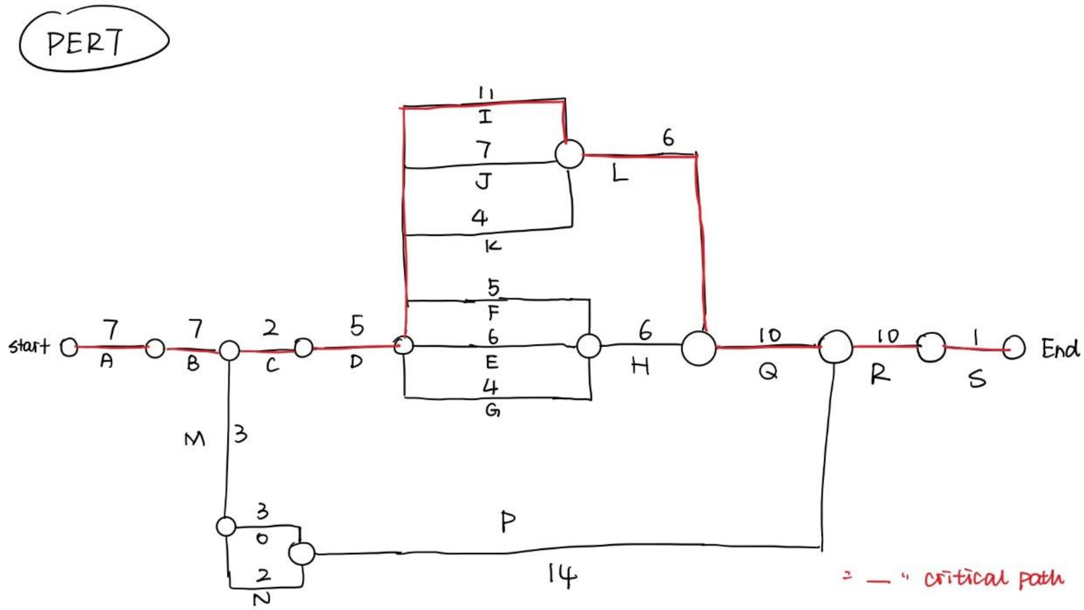


Figure 22: PERT Diagram

Table 6: PERT Activity Correspondence Table

Activity	Duration	Proceed b	ES	LS	TF	Durations	Variance	Expected Time(te)
Brainstorming	7		0	0	0	5-7-8	0.25	6.83
Project Planning	7	A	7	7	0	5-7-9	0.11	7.00
Hardware shopping	2	B	14	14	0	1-2-3	0.11	2.00
Prototyping and testing	5	C	16	16	0	2-4-5	0.25	3.83
Constructing Driving System	5	D	21	21	0	4-5-8	0.44	5.33
Construcing Cone Dispensing Mechanism	6	D	21	21	0	5-6-8	0.25	6.17
Constructing Sensor Mover Mechnism	4	D	21	21	0	3-4-6	0.25	4.17
Integrating machine	6	E,F,G	25	27	2	5-6-10	0.69	6.50
Motor Driver Circuit Experimentation and Fabrication	11	D	21	21	0	9-11-12	0.25	10.83
Sensor Circuit Experimentation and Fabrication	7	D	21	21	0	6-7-8	0.11	7.00
Power supply selection	4	D	21	21	0	3-4-6	0.25	4.16
Debugging	6	I,J,K	25	32	7	5-6-9	0.44	6.33
Running sample code	3	B	14	14	0	1-2-3	0.11	2.00
pin assignment	2	M	17	17	0	1-2-3	0.11	2.00
user interface design	3	M	17	17	0	2-3-5	0.25	3.17
programming code	14	O,N	19	20	1	10-14-15	0.44	13.50
System Integration	10	L,H	31	38	7	8-10-12	0.69	10.00
System Debugging	10	Q,P	33	49	16	8-10-12	0.69	10.00
Project Demonstration	1	R	44	59	15	0-1-2	0.11	1.00

Table 7: Critical Paths Based on PERT

PATH	LENGTH/DAYS
A-B-C-D-E-H-Q-R-S	54
A-B-C-D-F-H-Q-R-S	53
A-B-C-D-G-H-Q-R-S	52
A-B-C-D-I-L-Q-R-S	60
A-B-C-D-J-L-Q-R-S	55
A-B-C-D-K-L-Q-R-S	52
A-B-M-N-P-R-S	43
A-B-M-O-P-R-S	44

The longest path taken is A-B-C-D-I-L-Q-R-S which has a duration of 60 days in total.

Expected Time to Complete the Project:

$$T_e = \sum t_e = 53.49\ Days$$

Variance of the Project Completion Duration:

$$\sigma_e^2 = \sum \sigma^2 = 2.9$$

6.2 Budgeting

Table 8: Budget of Required Materials Sectioned by Subsystem

Subsystem	Item	Qty.	Cost	Source
Electromechanical	8*24*0.25 Aluminum sheet	1	$ 13.81	https://www.homedepot.ca/en/home/p.8-inch-x-24-inch-x-025-inch-aluminum-sheet-
	Shenzhen DC Motor Straight	3	$ 12.00	Project Kit
	Servo Motor	1	$ 12.00	Project Kit
	Paulin 1*4 aluminum tube	1	$ 23.78	https://www.homedepot.ca/en/home/p.papc-1x4-square-alum-tubing.1000170181.htr
	¼*2*4 Ply wood	1	$ 11.76	https://www.homedepot.ca/en/home/p.14-inch-x-2-feet-x-4-feet-birch-plywood-hanc
	Plastic Gear	1	$ 5.99	https://hobbyking.com/en_us/kimbrough-48pitch-73t-spur-gear.html
	Gear Rack	1	$ 18.00	https://www.ebay.com/itm/BOSTON-GEAR-G-579-GEAR-RACK-FOR-CLOCKS-ETC-CNC-
	Hinges	4	$ 0.36	https://www.ebay.com/p/20x-Miniature-Hinges-Nails-Screws-Fits-Dollhouse-1-12-Scal
	Omni wheels	4	$ 31.60	https://item.taobao.com/item.htm?id=571660599800&price=40-50&sourceType=item
Circuits	Turnigy 2200mAh 2S 25C Lipo Battery Pack	1	$ 8.99	https://hobbyking.com/en_us/turnigy-2200mah-2s-25c-lipo-pack-w-xt60.html
	Wires	1	$ 5.63	https://www.robotshop.com/ca/en/el-wire-blue-1m.html
	Solder Board	4	$ 0.40	https://item.taobao.com/item.htm?id=522043872157&price=0.5&original_price=0.5&
	QRE1113 IR Sensor	3	$ 9.39	https://www.robotshop.com/ca/en/sfe-digital-ir-line-sensor-qre1113.html
	L7805 5V 1.5A Voltage Regulator	4	$ 4.36	https://www.robotshop.com/ca/en/l78055v-15a-voltage-regulator.html
	H -Bridge Driver Board	1	$ 3.00	Project Kit
Microcontroller	PIC DevBugger Development Board with AC/DC Adapter and Cable Bus	1	$ 55.00	Project Kit
	Character LCD+Keypad (with the PIC encoder chip)	1	$ 8.00	Project Kit
	Real-time Clock (RTC) Chip and Coin Battery	1	$ 4.00	Project Kit

Total Project Cost of the Proposed Design: $228 CAD

Table 9: Sources for Budgeting

Source
https://www.homedepot.ca/en/home/p.8-inch-x-24-inch-x-025-inch-aluminum-sheet-metal.1000126786.html
Project Kit
Project Kit
https://www.homedepot.ca/en/home/p.papc-1x4-square-alum-tubing.1000170181.html
https://www.homedepot.ca/en/home/p.14-inch-x-2-feet-x-4-feet-birch-plywood-handy-panel.1000114111.html
https://hobbyking.com/en_us/kimbrough-48pitch-73t-spur-gear.html
https://www.ebay.com/itm/BOSTON-GEAR-G-579-GEAR-RACK-FOR-CLOCKS-ETC-CNC-LINEAR-MOTION-12-LONG-BRASS-NOS-/232419124289
https://www.ebay.com/p/20x-Miniature-Hinges-Nails-Screws-Fits-Dollhouse-1-12-Scale-Cabinet-Furniture/2114401531?iid=362504358235
https://item.taobao.com/item.htm?id=571660599800&price=40-50&sourceType=item&sourceType=item&suid=a3879175-8f36-4017-b607-85528cb7aa6d&ut_sk=1.We4NzyOduAQDADzamSQEiLf2_21646297_1548857663376.Copy.1&un=463d822e970cc012a8fb54add16a2d54&share_crt_v=1&sp_tk=77+lUTAzTGJIZEhPa0zvv6U=&cpp=1&shareurl=true&spm=a313p.22.ol.1008561703359&short_name=h.3GL9upy&sm=7a1a5d&app=chrome&price=40-50&sourceType=item&sourceType=item&suid=a3879175-8f36-4017-b607-85528cb7aa6d&ut_sk=1.We4NzyOduAQDADzamSQEiLf2_21646297_1548857663376.Copy.1&un=463d822e970cc012a8fb54add16a2d54&share_crt_v=1&sp_tk=77+lUTAzTGJIZEhPa0zvv6U=&cpp=1&shareurl=true&spm=a313p.22.ol.1008561703359&short_name=h.3GL9upy&sm=7a1a5d&app=chrome
https://hobbyking.com/en_us/turnigy-2200mah-2s-25c-lipo-pack-w-xt60.html
https://www.robotshop.com/ca/en/el-wire-blue-1m.html
https://item.taobao.com/item.htm?id=522043872157&price=0.5&original_price=0.5&sourceType=item&sourceType=item&suid=d82a5dd5-d63d-4047-9c31-388803dca5b1&ut_sk=1.We4NzyOduAQDADzamSQEiLf2_21646297_1548858619995.Copy.1&un=463d822e970cc012a8fb54add16a2d54&share_crt_v=1&sp_tk=77+lemJwdmJIZENBYm7vv6U=&cpp=1&shareurl=true&spm=a313p.22.319.1008922353858&short_name=h.3GyLOd2&sm=de3fc6&app=chrome
https://www.robotshop.com/ca/en/sfe-digital-ir-line-sensor-qre1113.html
https://www.robotshop.com/ca/en/l78055v-15a-voltage-regulator.html
Project Kit

Project Kit
Project Kit
Project Kit