Letter of Transmittal

Arfaz Hossain, Kohen Butler, Manny Dhesi
University of Victoria
3800 Finnerty Rd.
Victoria, BC V8P 5C2

April 7th, 2023

Dr. Dave Riddell
Ocean Networks Canada
University of Victoria, Queenswood Campus
2474 Arbutus Road
Victoria, BC V8N 1V8

Dear Dr. Riddell,

We are writing to transmit the proposed designs for the autonomous underwater vehicle requested by Ocean Networks Canada (ONC). Our report presents a comprehensive overview of three designs that meet the requirements outlined in the request. We are confident that this will provide valuable insight on each of our design's capabilities and help ONC select the best option for their needs.

The primary objective of this report is to provide detailed information on the design and testing of three unique prototypes developed by our team. We also included an in-depth analysis of the results obtained during the test of each design. In addition to the detailed information on the design and testing of each prototype, our report would provide ONC with additional technical information, including the electrical circuit diagrams and programs which will allow ONC to understand the technical aspects of each design and provide insights into the underlying functionality of each.

Finally, our report provides a recommendation based on the testing results. This recommendation summarises the strengths and weaknesses of each prototype, providing ONC with a clear understanding of the team's preferred option for their specific needs. This will enable ONC to make an informed decision, taking into account the results of the testing and the technical aspects of each prototype.

We trust that this report will be an excellent resource for ONC in selecting an autonomous underwater vehicle that aligns with the requirements. We are available to address any questions or concerns you may have regarding this report or the proposed designs.

Most Sincerely,

Kohen Butler, Arfaz Hossain and Manny Dhesi

Final Report

In response to

RFP-VN120-202301

Submitted to:

Gillian Saunders

Submitted by:

Arfaz Hossain, Kohen Butler, and Manny Dhesi

Date:

April 7, 2023

Executive Summary

Problem Statement

Ocean debris and marine life obstruct underwater cameras and sensors from gathering data about the oceans. Ocean Networks Canada (ONC) requested prototypes for an autonomous underwater vehicle to install cleaning devices on the sensors and cameras. These prototypes must simulate installing a cleaning device by placing a ping pong ball atop a target emitting a specific frequency of infrared light in a dry environment.

Prototype Designs

We proposed three designs to meet ONC's needs that each prioritize different potential objectives. All three designs consist of VEX Robotics components and can scale up for use as an autonomous underwater sensor cleaning robot. Yet each design delivers the device using a different strategy to locate and approach the target. *Design 1* promotes efficiency and durability to complete the task quickly and prevent repairs. *Design 2* priortizes durability and reliability to successfully complete the task under many conditions. *Design 3* focuses on mobility and versatility to complete potentially more complex tasks in various terrain.

Prototype Results

Leading up to the final demonstrations, the designs were rigorously tested and adjusted to meet their expectations and requirements. During the final demonstration and tests, each design's results were recorded including the time taken to deliver the ping pong ball, the percentage of successful attempts, the distance from the target the robot can sense the infrared and the number of collisions with the walls. Each design received a score across several categories to determine the capabilities and limitations of each design.

Conclusion and Recommendation

To summarize the findings of these results, each design had clear advantages and disadvantages. *Design 1* remained durable and affordable, though its imperfect movement was unreliable. *Design 2* proved its consistency and durability, but will damage the ocean environment surrounding the sensors and cameras. *Design 3* sacrifices speed and durability to excel in detecting and approaching the target. After some minor adjustments to the design and program we feel that *Design 3* is the optimal design for ONC. Although, we are prepared to continue with either of the three designs if ONC prefers another design.

Table of contents

List of Figures and Tables

1 | Introduction

Clients often request prototypes before the final product to test and evaluate the functionality of various design options. Prototypes help identify potential issues and allow for adjustments before investing time and resources into the final product.

1.1 | Ocean Networks Canada

Ocean Networks Canada (ONC) is a non-profit organization owned by the University of Victoria responsible for collecting data about Canadian oceans, specifically off the coast of British Columbia. ONC collects data to predict weather systems, track marine life, and forecast earthquakes and tsunamis using sensors and cameras on the ocean floor [1]. Unfortunately, activity deep in the ocean including currents and marine life often cover the sensors in sediment and microorganisms that prohibit the sensors from gathering data. Since the sensors are too deep for divers to clean, ONC is requesting prototypes for an autonomous underwater vehicle capable of locating sensors and installing a cleaning device on them [2].

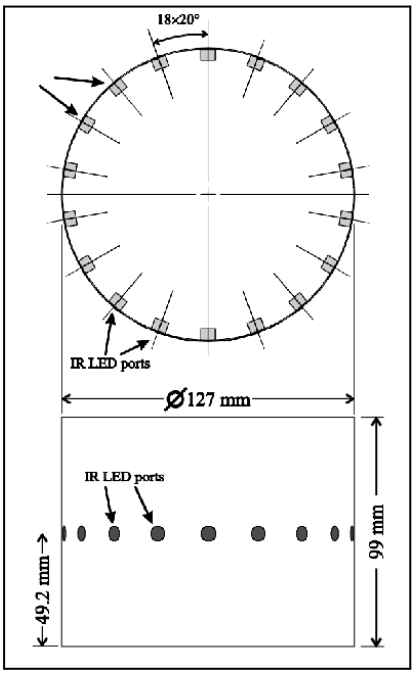

Figure 1.1: Infrared light source dimensions and specifications

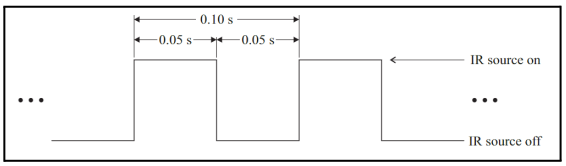

Figure 1.2: 10Hz square wave diagram

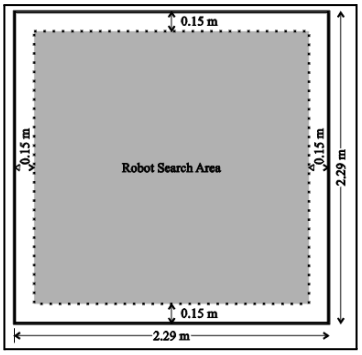

Figure 1.3: Search area dimensions

1.2 | Testing Environment

Before building an autonomous underwater vehicle, ONC requested prototypes for dry environment testing. During these tests, the robots must autonomously place a ping pong ball atop a target to simulate installing a cleaning device on an underwater sensor. The target, shown in Figure 1.1, is a cylinder equipped with infrared (IR) light emitting diodes (LEDs) at 20° intervals around the sides. These IR LEDs synchronously emit IR light in a 10Hz square wave pattern as illustrated in Figure 1.2. During a test, both the target and the robot

will be positioned and oriented randomly within the enclosed search area displayed in Figure 1.3. Neither the robot or the target will be placed within 15cm of the walls, though a robot may drive within 15cm of the walls during a test. The robot must autonomously locate and deliver the ping pong ball to the target, then exit the search area by driving to one of the four walls. [3]

2 | Design Descriptions

Three distinct designs have been proposed each with unique features and capabilities that meet the client's needs. *Design 1* emphasises efficiency and durability, designed to operate for extended periods with minimal maintenance requirements. *Design 2* focuses on durability and affordability to handle many rigorous tests at a low cost. *Design 3* prioritises mobility and versatility, allowing the robot to navigate different terrains and perform a variety of tasks. While each design exhibits distinct priorities and features, all three share a common goal of efficiently locating and reaching an infrared source.

2.1 | Design 1

Design 1 provides an innovative method of positioning objects while navigating through the ocean depths. This prototype utilises a crane-like mechanism to lift and place objects. The crane is connected to a gear that stabilizes the crane while it is lowered and lifted; it ensures that the crane will be fixed in position. *Design 1* adheres to all of the objectives and constraints specified by Ocean Networks Canada. More specifically, the robot possesses the necessary manoeuvrability and sensory skills to perform tasks within a dry lab environment.

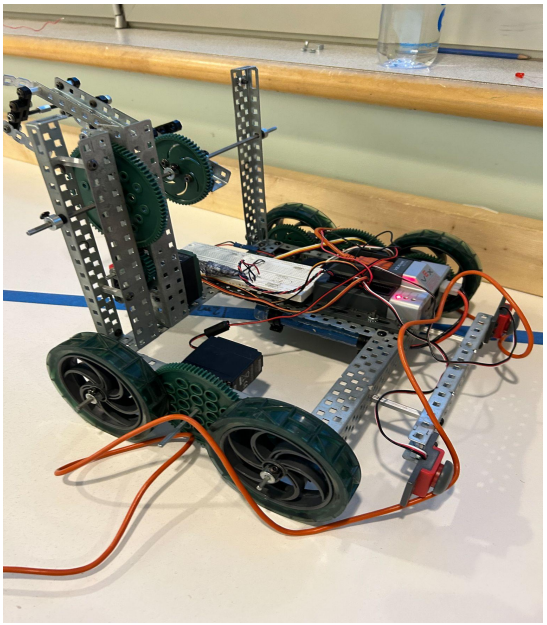

Figure 2.1.1: *Design 1* prototype

Materials and Components

Design 1 (Figure 2.1.1) consists of mechanical parts and an electrical system supplied by VEX Robotics; a trusted manufacturer of robotic parts. VEX Robotics provides a vast range

of robotic parts that can be used in a versatile manner. All mechanical and electrical parts used in Figure 2.1.1 can be found in the VEX Robotics Design System Kit [4] and listed in Table 2.1.1.

Table 2.1.1: *Design 1* components and cost
(Blank cells indicate unknown quantity or cost. Reference *Appendix A* for more details)

Part Description	Cost/Part	Quantity	Cost
Structural Components			
C-Channel Frame Rail	$2.00	10	$20.00
Chassis Bumper Angle	$2.00	2	$4.00
Bar	$2.00	2	$4.00
Wheel	$2.50	4	$10.00
Shaft	$2.00	8	$16.00
Gear		10	
Collection of Small Parts (Nuts, Bolts, etc.)	$1.00		
Electrical Components			
Microcontroller		1	
Battery		1	
Limit Switch	$2.00	2	$4.00
Button	$2.00	2	$4.00
Infrared Phototransistor	$0.50	1	$0.50
LED	$0.50	1	$0.50
Wire and Connectors	$0.25	6	$1.50
Resistor		1	
Bread Board		1	
Ultrasonic Sensor	$15.00	1	$15.00
Motor without Encoder	$10.00	3	$30.00
Motor with Encoder	$15.00	0	$0.00
Motor Controller		1	
Total			$109.50

Frame and Movement

This design prototype (Figure 2.1.1) features a movement system composed of four wheels which is powered by two motors connected to the back two wheels; a two wheel drive system. The motors are programmed to follow a linear path, and fully rotate a minimum of 360 degrees. This design (Figure 2.1.1) also features a gear based drive system. Previous versions of *Design 1* experienced difficulty in being able to rotate due to the gear ratio exerting an extreme downwards force on the left front wheel. However, *Design 1* successfully manages to rotate due to the implementation of the gear based drive system producing a greater torque. *Design 1* has a third and final motor connected to the arm mechanism. This allows for the arm to be lowered far enough so that objects can be lifted or placed, as well as raising the arm back consistently to the original position.

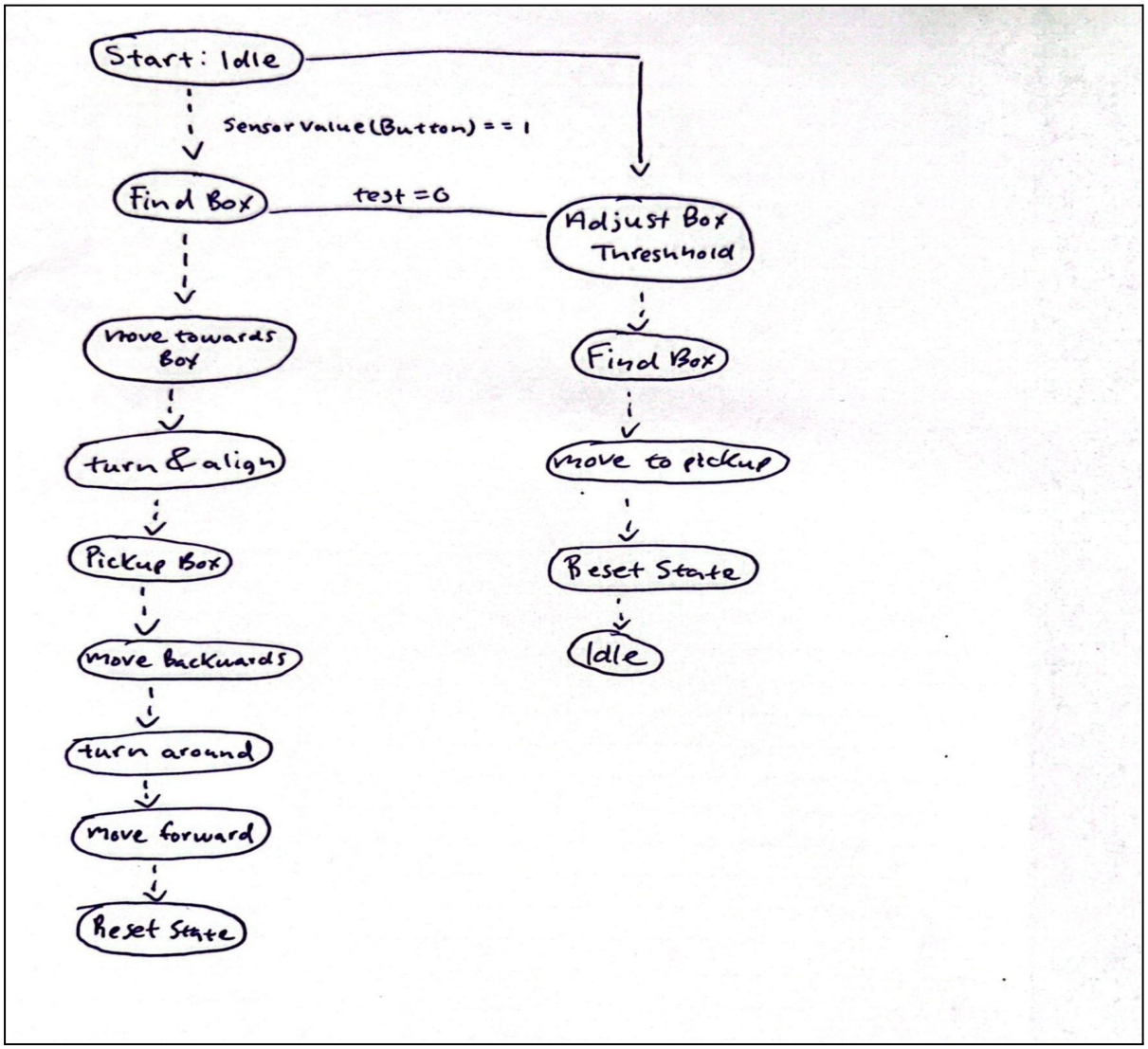

Figure 2.1.2: *Design 1* finite state machine diagram

Electrical Systems

The following list contains the parts found in Table 2.1.1 that are used for the electrical system of *Design 1*, along with their quantities:

- VEX Motor 2-Wire (3)
- VEX Microcontroller Bumper Switch (2)
- VEX Breadboard (1)
- VEX Infrared Sensor (1)
- VEX Ultrasonic Sensor (1)
- VEX Microcontroller (1)
- 7.2V Nickel-Metal Hydride NiMH 3000 mAh Battery (1)

All of these parts priorly listed are responsible for the navigational and sensory skills displayed by *Design 1*. The motors are used to physically move the robot in a desired way as

previously explained. The Bumper switches are responsible for detecting whether the robot has made contact with another object on either side. If the bumper switches are activated the robot will immediately rotate and steer away from the location of the object. This is due to the sensitive ecology of the ocean depths, it is best for the robot to have the least amount of interaction with the surrounding environment. The breadboard provides a platform for the connection of the sensors used in *Design 1.* The IR sensor is used to detect IR signals emitted by an object up to 30 feet away. The code implemented into *Design 1* makes the robot follow the IR signal to its source. The Ultrasonic sensor allows the robot to detect the distance between the front of the robot and the closest target. This allows the robot to not collide with objects while travelling linearly. The Battery used in the design of this robot has a lifespan of approximately 10 hours.

2.2 | Design 2

Design 2 prioritizes durability and affordability to withstand many tests under rough conditions while remaining inexpensive.

Materials and Components

The mechanical component of this design is composed of VEX Robotics components because they are easily interchangeable and affordable. All the fasteners and connections are stable as the pieces are designed to fit in many configurations. To ensure the electrical components are compatible, all the materials, aside from an LED and an infrared sensor, are VEX Robotics components. All the materials used and their cost are listed in Table 2.2.1.

Table 2.2.1: *Design 2* components and cost
(Blank cells indicate unknown quantity or cost. Reference *Appendix A* for more details)

Part Description	Cost/Part	Quantity	Cost
Structural Components			
C-Channel Frame Rail	$2.00	7	$14.00
Chassis Bumper Angle	$2.00	4	$8.00
Bar	$2.00	4	$8.00
Wheel	$2.50	4	$10.00
Shaft	$2.00	6	$12.00
Gear		8	
Collection of Small Parts (Nuts, Bolts, etc.)	$1.00		
Electrical Components			
Microcontroller		1	
Battery		1	
Limit Switch	$2.00	2	$4.00
Button	$2.00	1	$2.00
Infrared Phototransistor	$0.50	1	$0.50
LED	$0.50	1	$0.50
Wire and Connectors	$0.25	7	$1.75
Resistor		2	
Bread Board		1	
Ultrasonic Sensor	$15.00	1	$15.00
Motor without Encoder	$10.00	2	$20.00
Motor with Encoder	$15.00	1	$15.00
Motor Controller		1	
Total			$110.75

Frame and Movement

Design 2 features a square base with four wheels and a trapdoor apparatus reaching over the front of the robot, seen in Figure 2.2.1. The frame of the robot was inspired by the VEX Clawbot design [5], a previously designed robot by VEX Robotics known for its ability to lift and relocate objects. To move about the search area, two motors (one on each side of the robot) connected to idler gears, seen in Figure 2.2.2, alter the speed and direction of the wheels. Turning all the wheels in the same direction moves the robot forward and backward, whereas reversing the rotation of the left or right side will spin the robot. To maintain a tighter turning radius and to avoid hitting walls while cornering, this design is a few centimetres thinner than the VEX Clawbot. However, the more notable difference in this design is the trapdoor mechanism used to release the ping pong ball. This 13cm tall trapdoor is about 7cm in front of the robot to position the ping pong ball directly above the target. Powering the top motor opens the trapdoor and releases the ping pong ball straight onto the target such that it does not roll off of the target's flat surface.

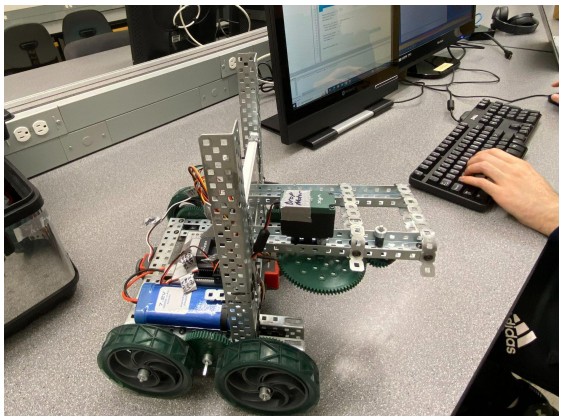

Figure 2.2.1: Image of *Design 2*

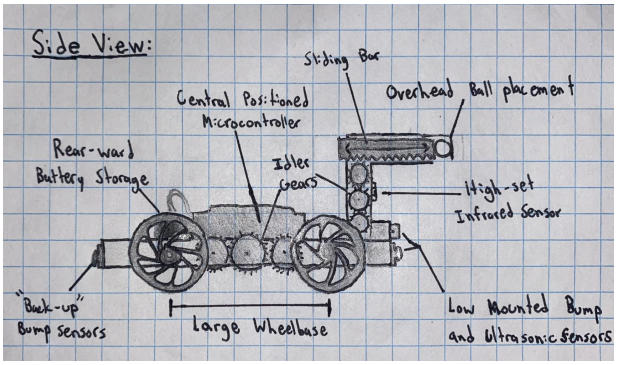

Figure 2.2.2: Concept sketch of *Design 2*

Electrical Systems

Design 2 uses an IR sensor, an ultrasonic sensor, and two limit switches to gather data about the robot's surroundings, which the microcontroller interprets as instructions to complete the task. Once the robot is powered on and the start button is pressed, the robot completes a 360° rotation while searching for the strongest 10Hz IR signal recorded by the IR sensor to distinguish the IR light emitted by the target from natural IR light. Next, it continues to spin until it is aligned with the strongest previously recorded IR signal. The robot then drives forward 75cm or until it encounters an object. If the object is the target, it approaches and places the ball. Otherwise it scans for an IR signal within a 90° range of the front of the robot and realigns with the source. The robot continues to move closer to the source of the IR signal until it places the ping pong ball. Afterward it reverses from the target, turns around, and drives towards the arena walls. The robot continues driving until the limit switches on the front of the robot (Figure 2.2.2) contact the walls. The microcontroller computes the angle relative to the wall using the ultrasonic range finder and positions itself parallel to the wall. A more detailed visual representation of the electrical system can be seen in Figure 2.2.3 or in *Appendix E.* Additional information regarding the set up of the electrical systems can be found in *Appendix C.*

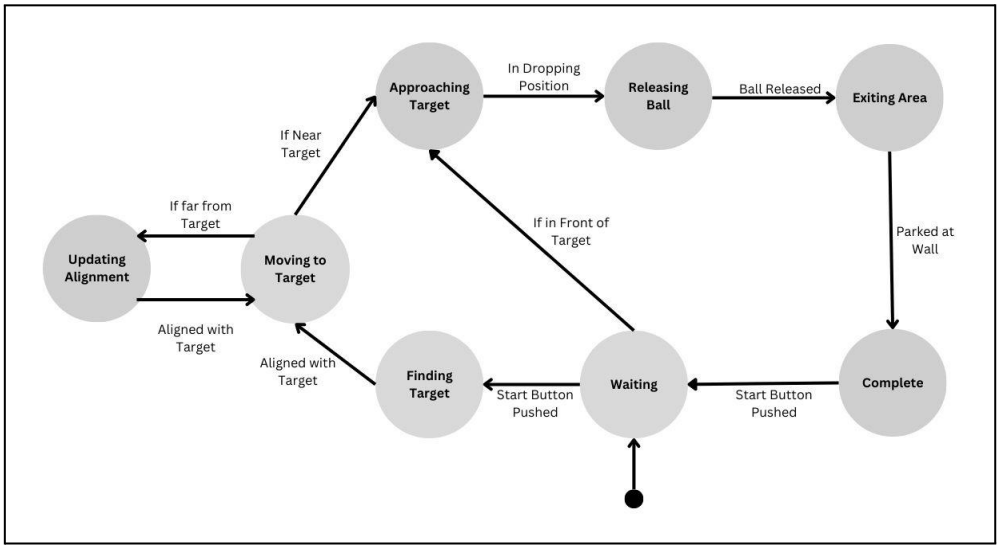

Figure 2.2.3: *Design 2* finite state machine diagram

2.3 | Design 3

Design 3 is a prototype designed with functionality, mobility and accuracy in mind. The robot uses two motors equipped with encoders to navigate and turn accurately, while three distinct sensors allow for measuring distance and IR signal source. The design is based on the VEX Robotics Clawbot [5], with an IR sensor in the front that scans the testing area every 0.05 seconds in search of the strongest IR signal. Once the source is detected, the robot navigates towards it. When the robot reaches the source, it stops and activates its third motor, releasing the ball from the caging area to simulate the depositing of a cleaning object on the target.

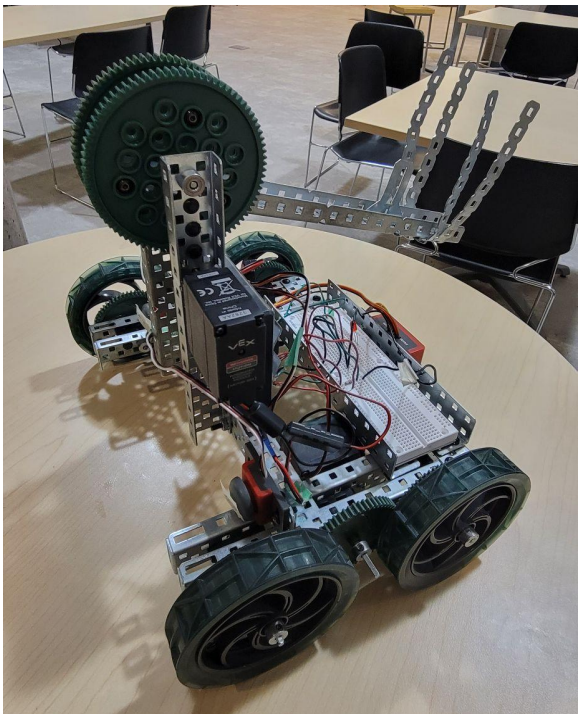

Figure 2.3.1: Most recent model of *Design 3*

Materials and Components

This design's interchangeable components and compatibility with other VEX Robotics designs make it a versatile option for various robotics projects. As part of the VEX Robotics line, all components and designs are made to work together seamlessly, allowing for endless customization and flexibility. The components used in this design can be seen in Table 2.3.1.

Table 2.3.1: *Design 3* components and cost
(Blank cells indicate unknown source or quantity. Reference *Appendix A* for more details)

Part Description	Cost/Part	Quantity	Cost
Structural Components			
C-Channel Frame Rail	$2.00	6	$12.00
Chassis Bumper Angle	$2.00	3	$6.00
Bar	$2.00	4	$8.00
Wheel	$2.50	4	$10.00
Shaft	$2.00	6	$12.00
Gear		8	
Collection of Small Parts (Nuts, Bolts, etc.)			
Electrical Components			
Microcontroller		1	
Battery		2	
Resistor		2	
Bread Board		1	
Button	$2.00	3	$6.00
Infrared Phototransistor	$0.50	1	$0.50
LED	$0.50	2	$1.00
Wire and Connectors	$0.25	8	$2.00
Ultrasonic Sensor	$15.00	1	$15.00
Motor without Encoder	$10.00	2	$20.00
Motor with Encoder	$15.00	1	$15.00
Total			$107.50

Mechanical Framework

The frame of *Design 3* is designed to accommodate two motors, which control forward and backward movements. The frame is equipped with four wheels that can be manoeuvred by altering the speed and direction of the two motors. This design ensures the robot can move around with ease and navigate to its target accurately. The upper part of the robot has a compartment where the ping pong ball is placed, and the lower part is designed to deposit the object in the desired location. The robot has two buttons to start and stop its functions, and a sonar sensor detects the distance from the robot to the target or boundary, ensuring the robot does not go beyond the designated area. During the search, the robot moves around and identifies the source of IR light through the IR sensor located at the front, which detects at 0.05 second intervals.

Electrical System

Design 3's electrical system is composed of an IR sensor, sonar sensor, and three motors, all of which are controlled by a microcontroller. The IR sensor locates the source of the strongest 10Hz infrared signal, while the sonar sensor determines the distance between the robot and its target or boundary. The two motors work together to propel the robot forward or backward,

and one motor is specifically responsible for dropping the cleaning device (in this case, a ball) from the upper part of the robot onto the target location.

The microcontroller receives and processes data from the sensors and motors, ensuring that the robot moves accurately and efficiently. With this design, the robot can navigate to the target location while minimising energy consumption and providing optimal results.

Software System

Design 3 is equipped with a reliable software script that operates continuously while using its front IR Sensor to search for a target. The process begins by capturing infrared signals within each 6-degree radius, while moving around. If the system fails to detect any strong and viable signal after approximately 4 seconds, it will stop moving and reposition itself to a new location for a better signal. The system is designed with wall detection capabilities to avoid collisions with obstacles during operation.

Once a signal is detected, the system will halt and move towards the location of the target. Additionally, the system has realignment capabilities that allow it to adjust its position towards the target while moving towards it, ensuring perfect alignment. As the system gets closer to the target, it will come to a stop and place the ping pong ball onto the target.

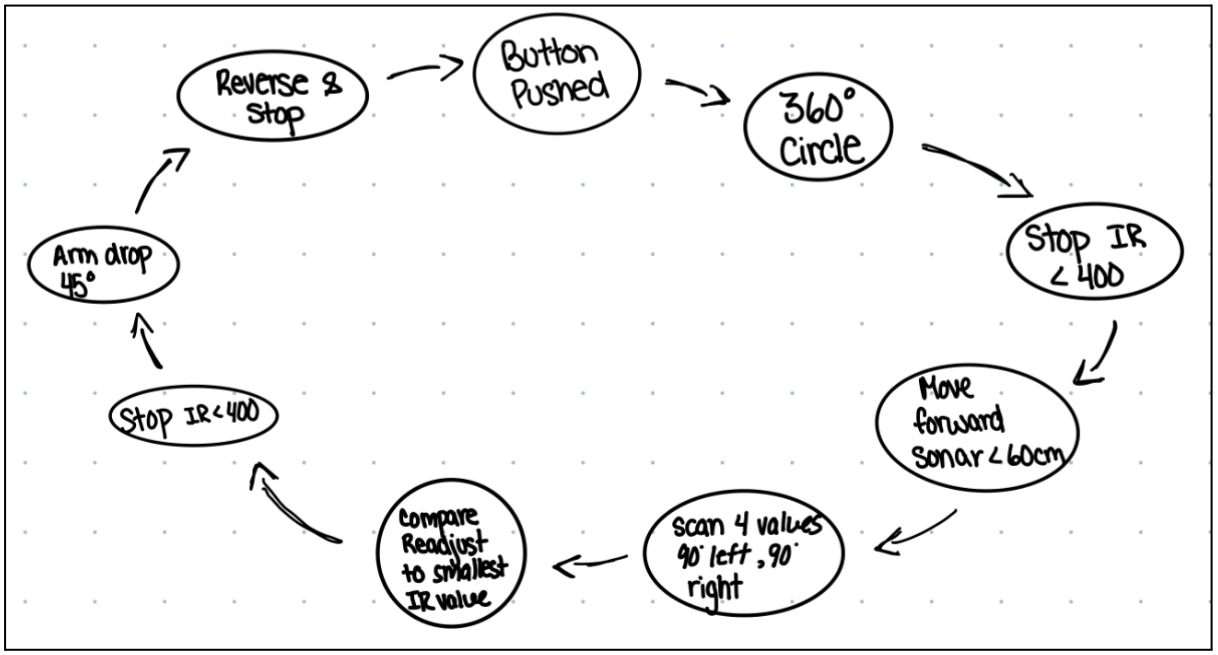

Figure 2.3.2: *Design 3* finite state machine diagram

3 | Results

Testing is a critical step in the prototype phase as it identifies potential issues and problems in designs by systematically evaluating the product under various scenarios and conditions. Different types of tests diagnose different functions or criteria of a design. The results of these tests help to evaluate and improve the designs.

3.1 | Design 1

Based on the dry environment testing, *Design 1* performed exceptionally well. This prototype managed to demonstrate strong skills that were related to the functionality and manoeuvrability aspects of the design. During the testing, *Design 1* successfully navigated itself towards the location of the infrared signal. Once *Design 1* had reached the location of the target, it used its crane mechanic to position an object on top of the target. Once *Design 1* successfully completed the task of detecting and positioning an object on top of the target, it successfully left the proximity. *Design 1* had a couple of weaknesses regarding its linear movement. This design struggled to move in a straight line for more than a metre as it often shifted slightly off course. This caused the robot to often miss the target entirely. More commonly, the robot made a mistake by thinking it had arrived at the target, and used its object positioning mechanism too early. This robot was cost effective and utilised the minimum number of parts necessary for construction, and also is extremely durable. However, the downsides oftenly undermine the success rate of this robot completing tasks. In terms of object positioning and durability, *Design 1* is a great choice as it can be incredibly accurate and consistent in terms of object placement.

3.2 | Design 2

Throughout the construction of the robot, several tests evaluated the primary functions of the robot. The first test evaluated the mechanical systems such as the movement and trapdoor mechanism by driving in a straight line, turning 90° and placing the ping pong ball on the target. The design exceeded the expectations with only concerns about the wheels slipping while cornering due to the thin frame and no differential gear. The second test assessed the electrical components such as the ability to detect the target and surrounding walls. This test revealed some difficulties with sensing the walls, which inspired the addition of limit switches to the front of the robot. The final demonstration tested both the electrical and mechanical systems during task completion, but also evaluated the durability and quality of construction with stress tests. *Design 2* executed the task quickly with 80% accuracy without contacting the walls in any configuration. However, the IR scanning algorithm causes the robot to shake in a manner that may damage the ocean floor. Therefore *Design 2* proves to be a relevant design if durability and reliability are the most important factors.

3.3 | Design 3

Design 3 scored well in terms of movement and sensing distance, indicating that it can navigate accurately and detect the infrared signal from a reasonable distance. However, it scored poorly in terms of time, indicating that it is slow in completing the task. Its build quality also received a mediocre score, indicating that there is room for improvement in terms of the organisation and management of components. Finally, although it was not the most cost-effective design, it still scored well in terms of cost. Overall, Design 3 may be a good choice for applications that prioritise accurate navigation and sensing distance over speed and cost-effectiveness.

3.4 | Comparative Analysis

To determine which prototype best suits ONC's needs, the success of each design was determined by several measures. Since installing a cleaning device and returning to the shore is the prototype's goal, the most important factor is reliability and consistency. Second, the robot should not damage the environment, the sensor, or itself while completing the task. Therefore, the prototype should 1) not come in contact with the target, 2) maneuver in a way that does not damage the ocean floor, and 3) contact any walls or obstacles at safe speeds. Although the design may withstand the dry environment testing, it must withstand harsh ocean conditions, meaning the quality of construction and wiring will influence its effectiveness. Lastly, to appeal to ONC the prototype should be affordable to build and run. This means the cost of the robot and the time taken to complete the task will impact its overall score. All of these criteria and the performance of each design are summarized in Table 3.4.1.

Table 3.4.1: Weighted objectives chart

Criteria	Weight	Design 1 - Manny		Design 2 - Kohen		Design 3 - Arfaz	
		Score (1-5)	Weighted Score	Score (1-5)	Weighted Score	Score (1-5)	Weighted Score
Reliability	0.25	3	0.75	4	1.00	4	1.00
Sensing Distance	0.20	4	0.80	5	1.00	4	0.80
Build Quality	0.05	4	0.20	5	0.25	3	0.15
Durability	0.20	5	1.00	5	1.00	4	0.80
Time	0.10	4	0.40	3	0.30	2	0.20
Cost	0.05	4	0.20	3	0.15	4	0.20
Movement	0.15	2	0.30	1	0.15	5	0.75
Total	**1.00**		**3.65**		**3.85**		**3.90**

4 | Conclusion

Each design offers an innovative approach to the request made by Ocean Networks Canada. *Design 1* provides an intuitive approach to object positioning, but it may not be capable of maintaining linear movement for over extended distances of more than 1 metre. *Design 2* possesses a durable framework along with a reliable electrical system as it manages to detect a target and quickly execute the task with 80% accuracy. However, it may cause ecological damage to the ocean environment as it shakes rapidly while cornering. *Design 3* has incredible navigational skills along with a sharp sensory ability. It can manoeuvre around any given environment and detect infrared signals from long range, however it is not cost effective and prioritises accuracy over speed.

4.1 | Recommendation

Our group has determined that *Design 3* is the most suitable robotic design for Ocean Networks Canada. *Design 3* has proven to be superior to *Design 1* and *Design 2* in terms of sensing a target and accurately navigating towards its location. Its algorithms and mechanisms enable it to manoeuvre through underwater environments with ease. *Design 3* still needs a few adjustments as it can function rather slowly compared to the other designs. This design could also reduce the amount of materials that it uses which in turn would reduce the overall cost for the construction of this robot. Many of the necessary adjustments that are required for *Design 3* could be implemented from the alternative design solutions. *Design 2* has a method of quickly executing given tasks and leaving the vicinity of an area. *Design 1* is incredibly cost effective while being able to execute tasks at the same rate as other design solutions. Overall, *Design 3* is the most optimal choice for ONC but it can be improved by taking mechanisms and ideas from the other design solutions. Although, we are prepared to continue with either of the three designs if ONC feels that *Design 3* does not meet their expectations.

5 | References

[1] "Data". Ocean Networks Canada. Accessed: February 16, 2023. https://www.oceannetworks.ca/data/

[2] D. Riddell. "Request for Proposals". RFP-VN120-‑202301. Ocean Networks Canada. Victoria, BC, Canada.

[3] I. T. Chelvan. Robot Design Project. (2023, Spring). Engineering Design and Communication. Victoria, BC: University of Victoria. [Online]. Available: https://bright.uvic.ca/d2l/le/content/271352/viewContent/2129453/View

[4] A. Brain, "VEX Robotics Design System Review," [Online]. Available: http://www.andybrain.com/extras/vex-robotics-design-system-review.htm. [Accessed: Mar. 25, 2023].

[5] VEX Robotics. Greenville, TX, USA. *Guide for Building the Clawbot.* Accessed: Mar. 22, 2023. [Online]. Available: https://content.vexrobotics.com/docs/instructions/276-2600-CLAWBOT-INST-0512.pdf

[6] I. T. Chelvan. Robot Design Project. (2023, Spring). Engineering Design and Communication. Victoria, BC: University of Victoria. [Online]. Available: https://bright.uvic.ca/d2l/le/content/271352/viewContent/2129465/View

Appendix A | Cost of Components

To determine the cost of the designs, each component of the VEX robotics kits was assigned a virtual price. Unfortunately, we were not provided with a virtual cost of several components (i.e. microcontroller, battery, etc.) or specific amounts of certain pieces such as “small parts”. Therefore, the cost of these components are excluded when calculating the cost of the robot as all three designs use roughly equal quantities of such items.

Table A.1: Parts and their virtual dollars

Part Description	Cost
Differential Gear System	$10.00
Bevel Gears (Set of 2)	$10.00
Structural Parts	$2.00
Small Parts	$1.00
Wheels	$5.00/pair
Limit Switch or Button	$2.00
Infrared or standard LED	$0.50
Infrared Phototransistor	$0.50
Housing with 3 Pins or 3 Sockets	$0.25
Butt Connector	$0.50
Ultrasonic Sensor	$15.00
Bipolar Junction Transistor (2N4401)	$0.25
Motor without Encoder	$10.00
Motor with Encoder	$15.00
Potentiometer	$2.00

Structural parts include the 25-hole Bar, 20-hole Bar, 16-hole Chassis Bumper,20-hole C-Channels,15-hole C-Channels, 30-hole Angle Bars,20-hole Angle Bars, 15-hole Angle Bars, 20" Shafts, and gears. *Small parts* include most small parts included in the small box of your kit such as bearing mounts, bolts, nuts, short shafts, spacers, and rivets. However, the number of parts for $1 is not specified. [6]

Appendix B | Weighted Objectives Parameters

To determine the capability of each robot within a given parameter, each design will receive a score from 1 to 5. *Reliability* measures the likelihood of the robot completing its intended task without failures, with a score of 100% being the most reliable. *Sensing Distance* refers to the range at which the robot detects the infrared signal, where detecting the signal from more than 2 metres scores the highest. *Build Quality* considers the organisation, labelling of components, and overall neatness of the robot. A shakedown will reflect the robot's ability to withstand physical stress as it is stretched, compressed and inverted. This will be recorded as its *durability* from 1 to 5, 5 being the most durable. *Time* evaluates the robot's speed in completing the task where a design completing the task in less than 30 seconds will score the highest. *Cost* is a measure of how cost-effective the robot is in terms of all the components, a design that costs less than $100 will be the most economical, hence will score the highest. *Movement* assesses the robot's accuracy and eco-friendliness in navigation, with a score of not contacting walls or the target being the most precise and efficient. All details regarding the evaluation of the robots are described in Table B.1.

Table B.1: Score descriptions for weighted objectives chart

Criteria	Score				
	5	**4**	**3**	**2**	**1**
Reliability	100%	75%	50%	25%	No Successful Tests
Sensing Distance	> 2 metres	> 1 metres	> 50 cm	< 50 cm	Does not sense IR
Build Quality	All components labelled and properly managed	Few components are not labelled or not properly managed	Less than 5 components not labelled or not properly managed	More than 5 components not labelled or not properly managed	Components not labelled and managed properly
Durability	No issues during shake test	1 or 2 minor issues during shake test	1 or 2 major issues during shake test	Several problems occur during shake test	Fails to complete task after shake test
Time	< 30 seconds	< 50 seconds	< 1 minute	> 1 minute	>> 1 minute
Cost	< $100	< $110	< $130	< $150	> $150
Movement	Does not contact walls or the target	Contacts the walls or target < 2 times	Contacts the walls or target < 4 times	Contacts the walls or target > 5 times	Does not move in an environmentally friendly manner

Appendix C | Electrical Circuit Diagrams

This section includes electrical circuit diagrams for the three designs. These diagrams provide a detailed view of the components, wiring, and connections used to construct the electrical system. The diagrams are essential for understanding the electrical design and troubleshooting any issues that may arise during construction or operation. By following the diagrams carefully, we can ensure that the system is assembled correctly and functions as intended.

Design 1 Schematic

The electrical circuit diagram for *Design 1* can be seen in Figure C.1.1 and Figure C.1.2

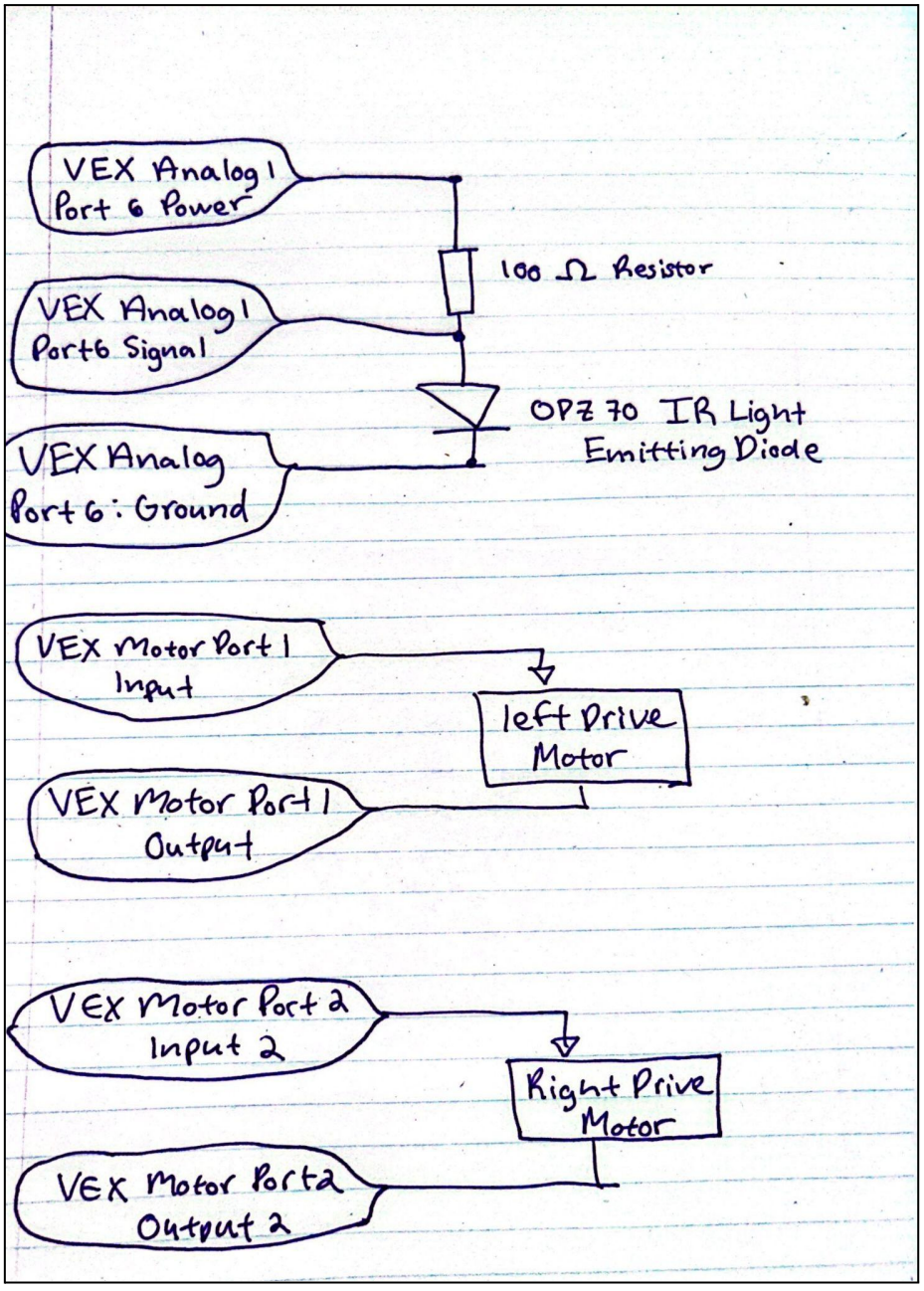

Figure C.1: *Design 1* electrical circuit diagram

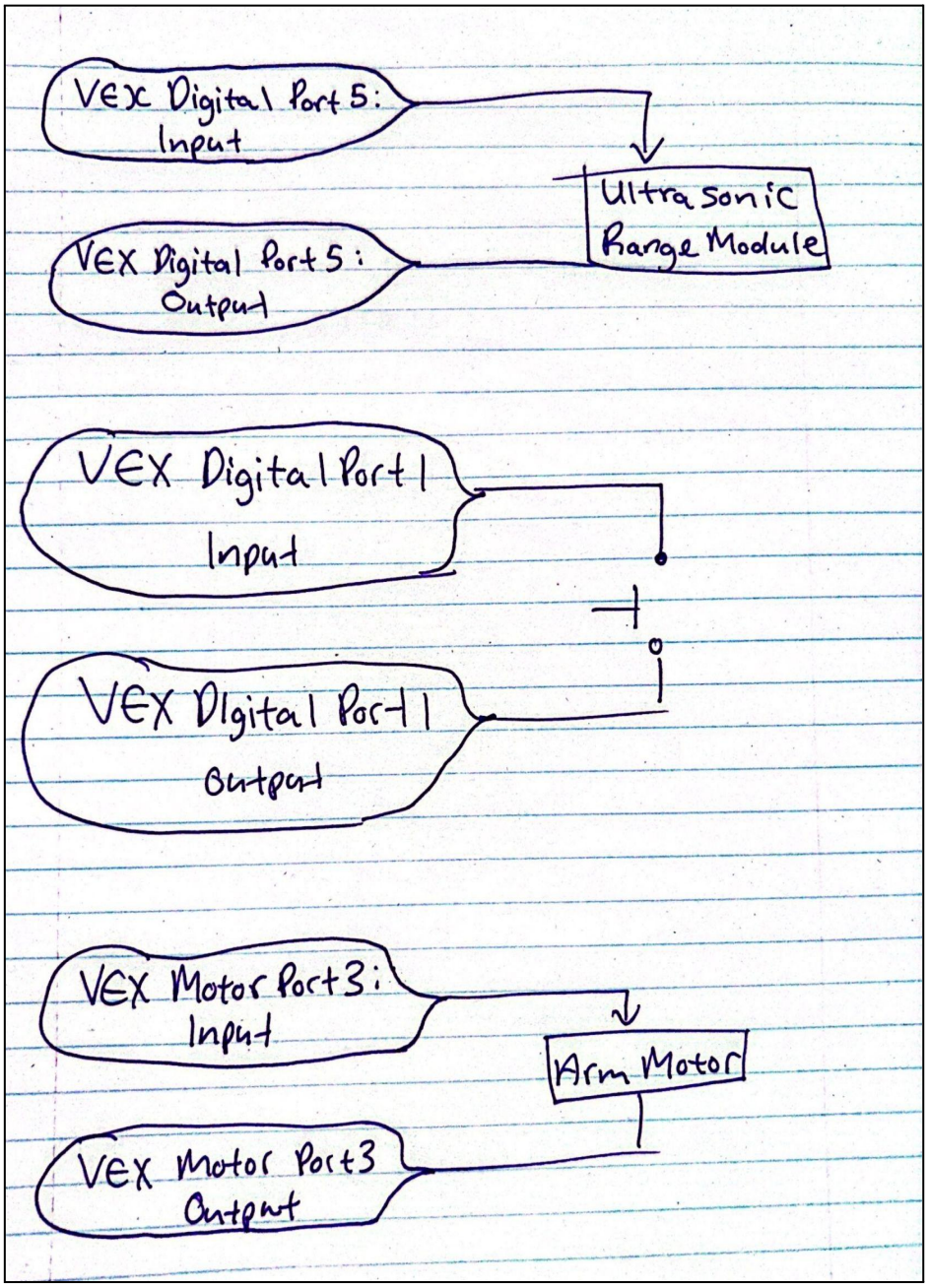

Figure C.2: *Design 1* electrical circuit diagram

Design 2 Schematic

The electrical circuit diagram for *Design 2* can be seen in Figure C.2.

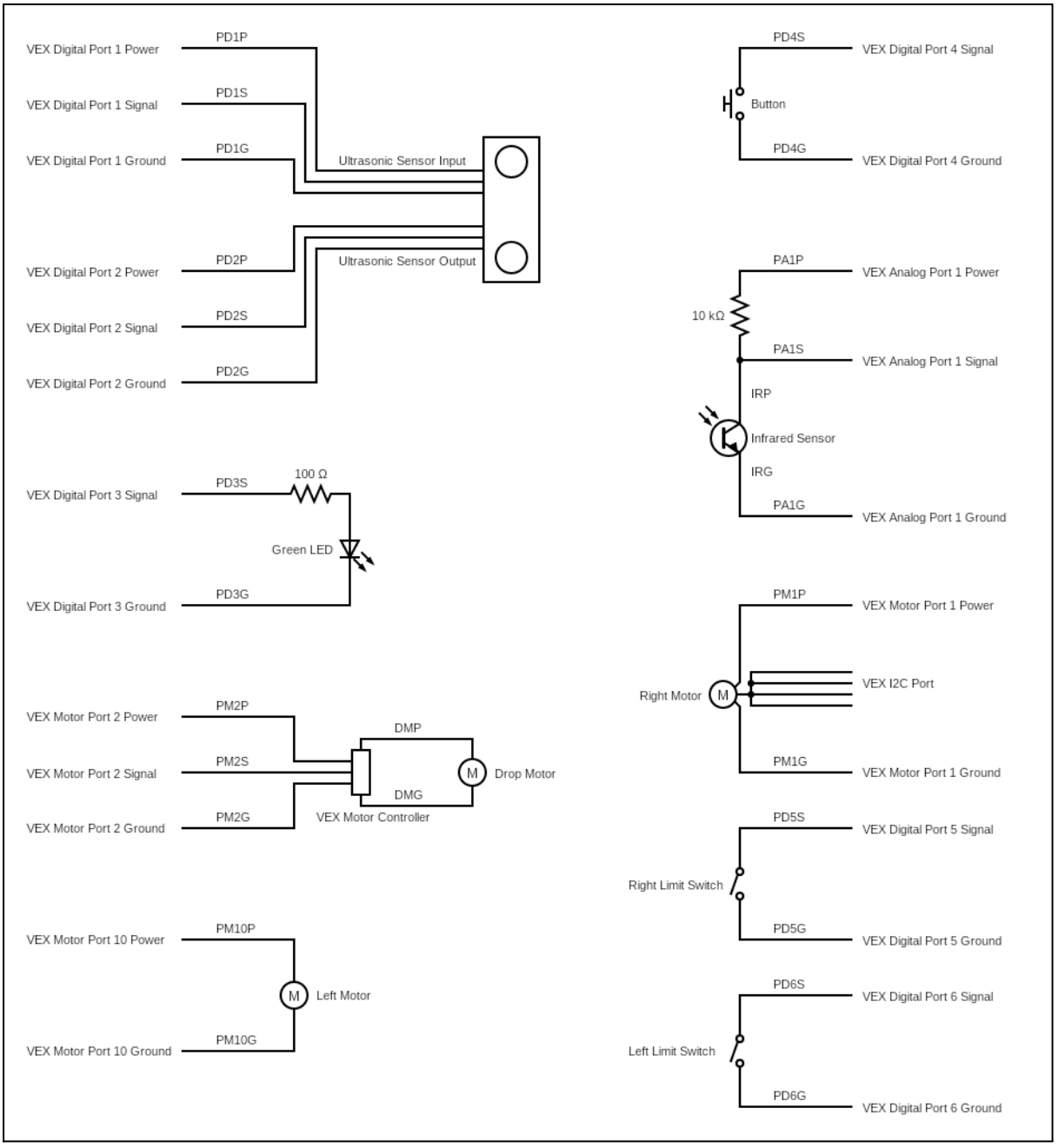

Figure C.3: *Design 2* electrical circuit diagram

Design 3 Schematic

The electrical circuit diagram for *Design 3* can be seen in Figure C.3:

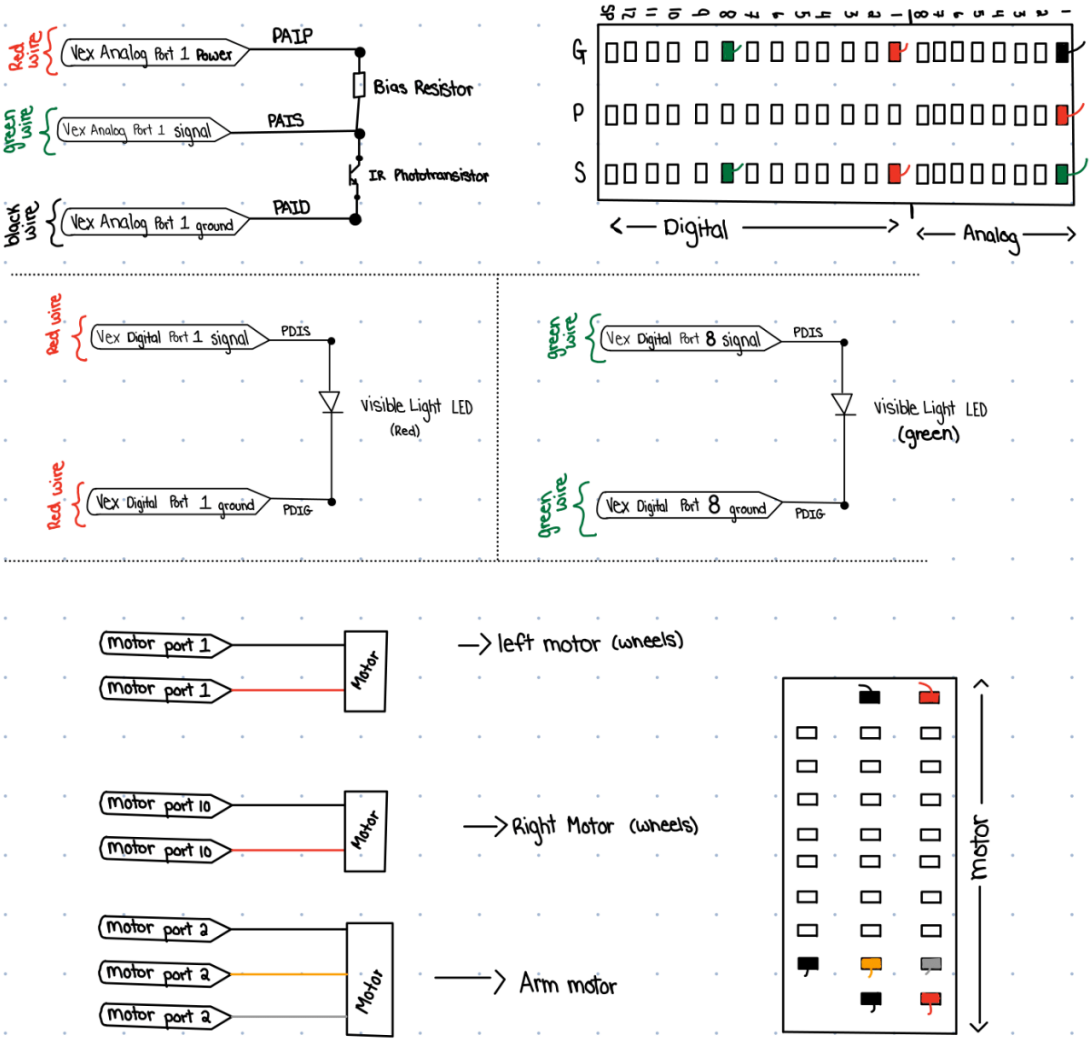

Figure C.4: *Design 3* electrical circuit diagram

Appendix D | Programs

This section includes all the program scripts for the three designs. These programs were designed and implemented to control the robots' movements, sensors, and various components. The information provided in this section serves as a reference for anyone interested in replicating or modifying the designs presented in this report.

Design 1 Program

```
#pragma config(Sensor, in1, box, sensorReflection)
#pragma config(Sensor, dgtl1, Button, sensorTouch)
#pragma config(Sensor, dgtl4, wall, sensorSONAR_inch)
#pragma config(Sensor, dgtl6, red, sensorLEDtoVCC)
#pragma config(Motor, port10, LeftDrive, tmotorVex393_HBridge, openLoop, reversed,
driveLeft)
#pragma config(Motor, port1, RightDrive, tmotorVex393_MC29, openLoop, driveRight)
#pragma config(Motor, port3, ArmDrive, tmotorVex393_MC29, openLoop)
#pragma config(Sensor, dgtl2, RightWall, sensorTouch)
#pragma config(Sensor, dgtl3, LeftWall, sensorTouch)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//

int findbox(int hold, int speedL, int speedR)
{ // function to locate box
   // clears both timers
   clearTimer(T1);
   clearTimer(T2);

   int Bigdif = 0;              // Initialize a variable to keep track of the biggest
difference in sensor reading
   int dif = 0;                 // Initialize a variable to keep track of the difference in
sensor reading
   int current;                 // Initialize a variable to hold the current sensor reading
   int last = SensorValue(box); // Initialize a variable to hold the last sensor reading

   while (Bigdif < hold)
   { // Loop until the difference in sensor readings exceeds the hold value

       if (SensorValue(RightWall) == 1)
       { // if the Right bumpper touches a wall
           // stops motors
           motor[RightDrive] = 0;
           motor[LeftDrive] = 0;
           wait1Msec(500);
           // turns around
           motor[RightDrive] = -50;
```

```
    motor[LeftDrive] = 50;
    wait1Msec(1000);
    // move forward
    motor[RightDrive] = 50;
    motor[LeftDrive] = 50;
    wait1Msec(500);
}
if (SensorValue(LeftWall) == 1)
{ // if the Left bumper touches a wall
    // stops motors
    motor[RightDrive] = 0;
    motor[LeftDrive] = 0;
    wait1Msec(500);
    // turns around
    motor[RightDrive] = 50;
    motor[LeftDrive] = -50;
    wait1Msec(1000);
    // move forward
    motor[RightDrive] = 50;
    motor[LeftDrive] = 50;
    wait1Msec(500);
}
if (SensorValue(wall) < 5)
{ // if the  front of a robot is close to a wall
    // stops motors
    motor[RightDrive] = 0;
    motor[LeftDrive] = 0;
    wait1Msec(500);
    // move back
    motor[RightDrive] = -50;
    motor[LeftDrive] = -50;
    wait1Msec(1000);
    // truns around
    motor[RightDrive] = -50;
    motor[LeftDrive] = 50;
    wait1Msec(500);
}

motor[RightDrive] = speedR; // Set the speed of the right motor
motor[LeftDrive] = -speedL; // Set the speed of the left motor

if (time1[T1] >= 60)
{                                // If a minute has passed
    current = SensorValue(box); // Get the current sensor reading
```

```
            dif = current - last;       // Calculate the difference in sensor reading since
the last loop iteration
            if (dif > Bigdif)
            {                 // If the difference in sensor reading is larger than the
current largest difference
                Bigdif = dif; // Update the largest difference
            }
            last = current; // Update the last sensor reading
            clearTimer(T1); // Reset the timer
        }

        if (time1[T2] >= 2000)
        {             // If two seconds have passed
            return 0; // Exit the function with a value of 0
        }
    }
    // stops motors for 1s
    motor[RightDrive] = 0;
    motor[LeftDrive] = 0;
    wait1Msec(1000);

    return 1; // Exit the function with a value of 1
}

void LowerArm()
{ // function to lower arm on box

    wait1Msec(1000);
    motor[ArmDrive] = 20;  // lowers arm
    wait1Msec(1500);       // waits 1.5 secs
    motor[ArmDrive] = -20; // lifts arm back up
    wait1Msec(2500);       // waits 2.5 secs
    motor[ArmDrive] = 0;   // stops arm
    wait1Msec(2000);       // waits 2 secs
}

task main()
{

    while (true)
    { // loop that checks if start button is clicked
        if (SensorValue(Button) == 1)
        { // if button is clicked
```

```
            int hold = 78; // set new IR dif value call hold

            int test = findbox(hold, 50, 50); // call the "findbox", make test the return
value
            while (test == 0)
            {                                 // if findbox fails
                hold = hold - 10;             // make hold value lower
                test = findbox(hold, 50, 50); // call "findbox" and set test to the return
value
            }
            wait1Msec(500);       // wait 0.5s
            SensorValue(red) = 1; // turn red light on
            wait1Msec(1000);      // wait 1s
            SensorValue(red) = 0; // turn red light off

            if (SensorValue(wall) > 10)
            {                           // if the robot is not near the box
                motor[RightDrive] = 40; // turn motors on
                motor[LeftDrive] = 40;
                wait1Msec(2500); // wait 2 secs
            }

            // stops  motors for 0.5s
            motor[RightDrive] = 0;
            motor[LeftDrive] = 0;
            wait1Msec(500);
            motor[RightDrive] = 40; // Right motor to 40
            motor[LeftDrive] = -40; // Left motor to -40
            wait1Msec(750);         // wait 0.75sec (ensures that the robot won't be facing
the box so the findbox function still works for the 2nd time)
            // stops motors
            motor[RightDrive] = 0;
            motor[LeftDrive] = 0;
            hold = 85;                    // set new IR dif value call hold
            test = findbox(hold, 45, 45); // call "findbox" and set test to the return value
            while (test == 0)
            {                                 // if test fails
                hold = hold - 10;             // lower IR hold value
                test = findbox(hold, 45, 45); // call "findbox" and set test to the return
value
            }
            wait1Msec(500); // wait 0.5s

            // turn light on and off
```

```
    SensorValue(red) = 1;
    wait1Msec(1000);
    SensorValue(red) = 0;

    clearTimer(T3); // clear timer 3

    int m = 1;
    while (m == 1)
    {
        if (SensorValue(wall) < 5)
        {                          // if the robot is next the box
            motor[RightDrive] = 0; // turn off motors
            motor[LeftDrive] = 0;
            wait1Msec(1000); // wait 1s
            LowerArm();      // call "LowerArm"
            wait1Msec(500);  // wait 0.5s
            // turn light on and off
            SensorValue(red) = 1;
            wait1Msec(500);
            SensorValue(red) = 0;
            // reverse for 1 sec
            motor[RightDrive] = -35;
            motor[LeftDrive] = -37;
            wait1Msec(1000);
            // turn 180 degrees
            motor[RightDrive] = 60;
            motor[LeftDrive] = -60;
            wait1Msec(2000);
            // move to middle of area
            motor[RightDrive] = 50;
            motor[LeftDrive] = 50;
            wait1Msec(2000);
            // stop
            motor[RightDrive] = 0;
            motor[LeftDrive] = 0;
            m = 0;
        }
        else
        { // if away from the box drive to it
            motor[RightDrive] = 36;
            motor[LeftDrive] = 36;
        }
    }
}
```

```
    }
}
```

Design 2 Program

```
#pragma config(I2C_Usage, I2C1, i2cSensors)
#pragma config(Sensor, in1,    IR_SENSOR,      sensorAnalog)
#pragma config(Sensor, dgtl1,  ULTRASONIC_IN,  sensorSONAR_mm)
#pragma config(Sensor, dgtl3,  GREEN_LED,      sensorDigitalOut)
#pragma config(Sensor, dgtl4,  START,          sensorTouch)
#pragma config(Sensor, dgtl5,  RIGHT_LIMIT,    sensorTouch)
#pragma config(Sensor, dgtl6,  LEFT_LIMIT,     sensorTouch)
#pragma config(Sensor, I2C_1,  ,               sensorQuadEncoderOnI2CPort,    ,
AutoAssign )
#pragma config(Motor,  port1,           RIGHT_MOTOR,   tmotorVex393_HBridge, openLoop,
encoderPort, I2C_1)
#pragma config(Motor,  port2,           DROP_MOTOR,    tmotorVex393_MC29, openLoop)
#pragma config(Motor,  port10,          LEFT_MOTOR,    tmotorVex393_HBridge, openLoop,
reversed)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//

const float DEGREES_TO_ROTATIONS_CONSTANT = 123;
const float STEPS_PER_REVOLUTION = 627;
const float WHEEL_CIRCUMFERENCE = 2 * 4.9 * PI;
// Conversion Constants

void turn_left(float degrees, int speed){
         // Given an amount to turn in degrees and a speed to turn at,
    // Turn left the desired amount at the desired speed
    float steps = (degrees / DEGREES_TO_ROTATIONS_CONSTANT) * STEPS_PER_REVOLUTION; //
Convert degrees to steps the motor must make
    resetMotorEncoder(RIGHT_MOTOR);  // Reset the motor encoder to 0
    motor[RIGHT_MOTOR] = speed;      // Spin the right wheels forward
    motor[LEFT_MOTOR] = -1 * speed;  // Spin the left wheels backwards
    while(getMotorEncoder(RIGHT_MOTOR) < steps){
        // Continue turning until the desired rotation distance is achieved
    }
    motor[RIGHT_MOTOR] = 0; // Stop the robot
    motor[LEFT_MOTOR] = 0;
}

void turn_right(float degrees, int speed){
         // Given an amount to turn in degrees and a speed to turn at,
    // Turn right the desired amount at the desired speed
    float steps = -1 * (degrees / DEGREES_TO_ROTATIONS_CONSTANT) * STEPS_PER_REVOLUTION;
// Convert degrees to steps the motor must make
    resetMotorEncoder(RIGHT_MOTOR);   // Reset the motor encoder to 0
    motor[RIGHT_MOTOR] = -1 * speed;  // Spin the right wheels backwards
    motor[LEFT_MOTOR] = speed;        // Spin the left wheels forwards
    while(getMotorEncoder(RIGHT_MOTOR) > (steps)){
        // Continue turning until the desired rotation distance is achieved
    }
    motor[RIGHT_MOTOR] = 0; // Stop the robot
    motor[LEFT_MOTOR] = 0;
}

int detect_IR(){
         // Determines the strength of a 10Hz infrared signal at a given time
         // Returns the strength of the signal as a positive integer between 0 and ~3000
```

```
        int check_1 = SensorValue(IR_SENSOR);         // Measure the signal strength at
instant 1
        wait1Msec(50);                                // delay 0.05s (time between high
power and low power of 10Hz square wave
        int check_2 = SensorValue(IR_SENSOR);         // Measure the signal strength at
instant 2
        int signal_strength = abs(check_2 - check_1); // determine the difference in IR
strength of the LED and the environment
        return signal_strength;
}

void find_IR(){
        // Aligns the robot with the source of the strongest infrared signal
    float percent_error = 0.10; // The percent the IR signal must be within of the
strongest
        int degrees_per_check = 2;            // Determines the amount of rotation
between measurements
        int strongest_ir = detect_IR();
        int current_ir = strongest_ir;
        int i = 0;
    while(i < 220){                       // Complete a full circle searching for the
highest intensity IR signal
        turn_left(degrees_per_check, 90);
        current_ir = detect_IR();
        if(current_ir > strongest_ir){
                strongest_ir = current_ir;
        }
        i += degrees_per_check;
    }
                                          // Begin spinning in a circle again and stop
when the strongest IR signal is found again
    while(detect_IR() < strongest_ir - (strongest_ir * percent_error)){
        turn_left(degrees_per_check, 90);
    }
}

void update_alignment(){
        // Ensures that the robot is aligned with the target
        float percent_error = 0.20; // Percent error of strongest IR signal
        int degrees_per_check = 2;  // The amount of degrees between each measurement
        turn_left(60, 60);          // scan 45 degrees to the left of the target and 45
degrees
        int strongest_ir = detect_IR(); // to the right of the target and record the
strongest
        int current_ir = strongest_ir;  // IR signal
        int i = 0;
        while(i < 60){
                turn_right(degrees_per_check, 90);
                current_ir = detect_IR();
                if(current_ir > strongest_ir){
                strongest_ir = current_ir;
        }
        i += degrees_per_check;
    }
    // Turn left until the strongest IR signal is found again
    while(detect_IR() < strongest_ir - (strongest_ir * percent_error)){
```

```
        turn_left(degrees_per_check, 90);
    }
}

void move_forward(float distance){
        // Drive forward a set distance (in cm) until the given distance is reached
        // If an object is encountered before reaching the given distance, slow down
and stop
    float steps = (distance / WHEEL_CIRCUMFERENCE) * STEPS_PER_REVOLUTION; // Convert
distance to motor steps
    resetMotorEncoder(RIGHT_MOTOR);  // Set the encoder to 0
    int wall_distance = SensorValue(ULTRASONIC_IN); // Check the distance from the
nearest object
    while(getMotorEncoder(RIGHT_MOTOR) < steps && wall_distance > 300){
        // Continue driving until the step count is reached or the robot is within 30cm
of an object
       // If within 40cm of an object drive slow
        if(wall_distance < 400){
           motor[RIGHT_MOTOR] = 30;
           motor[LEFT_MOTOR] = 30;
       }
       // If no objects are within 40cm, drive at full speed
       else{
                motor[RIGHT_MOTOR] = 100;
           motor[LEFT_MOTOR] = 107;
       }
       wall_distance = SensorValue(ULTRASONIC_IN);  // Update the distance from the
nearest object
    }
    motor[RIGHT_MOTOR] = 0;  // Stop the Robot
    motor[LEFT_MOTOR] = 0;
}

void reverse(float distance){
        // Drive backwards a set distance (in cm) until the given distance is reached
    float steps = -(distance / WHEEL_CIRCUMFERENCE) * STEPS_PER_REVOLUTION; // Compute
the number of motor steps
    resetMotorEncoder(RIGHT_MOTOR);  // Set the encoder to 0
    motor[RIGHT_MOTOR] = -50; // Turn the motors in reverse
    motor[LEFT_MOTOR] = -50;
    while(getMotorEncoder(RIGHT_MOTOR) > steps){
        // Reverse until the desired distance is reached
    }
    motor[RIGHT_MOTOR] = 0;  // Stop the Robot
    motor[LEFT_MOTOR] = 0;
}

void optimal_distance(){
        // Positions the robot the optimal distance from the target
    // If the robot is too close back up to the correct position
    // If the robot is too far, move forward until the optimal distance is reached
        int droppingDistance = 67; // The ideal distance from the target to the
ultrasonic
                                // sensor for dropping the ball is 67mm
        int dropping_Uncertainty = 5;  // The robot maybe up to 5mm off of the exact
distance for dropping the ball
```

```
                                        // This helps with inconsistencies with the
ultrasonic sensor
          // If the robot is too far from the target, drive closer
          while(SensorValue(ULTRASONIC_IN) > droppingDistance + dropping_Uncertainty){
                   motor[RIGHT_MOTOR] = 40;
                        motor[LEFT_MOTOR] = 40;
        }
          motor[RIGHT_MOTOR] = 0; // Stop the motors
               motor[LEFT_MOTOR] = 0;
               wait1Msec(100); // Delay 0.10s to let the robot settle

               // If the robot is too close to the target, back up slowly
               while(SensorValue(ULTRASONIC_IN) < droppingDistance -
dropping_Uncertainty){
                        motor[RIGHT_MOTOR] = -40;
                        motor[LEFT_MOTOR] = -40;
               }
               motor[RIGHT_MOTOR] = 0; // Stop the motors
               motor[LEFT_MOTOR] = 0;
               wait1Msec(100); // Delay 0.10s to let the robot settle
}

void drop_ball(){
          // Opens the trapdoor that holds the ping pong ball
          motor[DROP_MOTOR] = 100; // Start the Motor
          wait1Msec(200); // Delay the amount of time it take for the motor to rotate the
trapdoor once
        motor[DROP_MOTOR] = 0; // Stop the motor
}

void exit_area(){
          const float robot_width = 148.5;
          reverse(20);                // Back up 20cm
          turn_left(90, 60);          // Turn 90 degrees to the left
          int wall_interference = 0;  // Assume the walls are not interfering with the
robot
          motor[RIGHT_MOTOR] = 45;    // drive forward
          motor[LEFT_MOTOR] = 45;
          while(wall_interference == 0){  // Continue forward until some wall
interference occurs
                if(SensorValue(ULTRASONIC_IN) < 130){
                    wall_interference = 1;  // Wall interference type 1 if the
ultrasonic sensor senses the wall
              }
              if(SensorValue(RIGHT_LIMIT)){
                 wall_interference = 2;  // Wall interference type 2 if the right limit
switch is activated
              }
              if(SensorValue(LEFT_LIMIT)){
                 wall_interference = 3;  // Wall interference type 3 if the left limit
switch is activated
              }
          }
          motor[RIGHT_MOTOR] = 0; // Stop the robot
          motor[LEFT_MOTOR] = 0;
          if(wall_interference == 1){ // If type 1, stop and end task
```

```
        }
        else if(wall_interference == 2){  // If type 2
                float degrees = 0.9 * (90 - (asin(robot_width /
SensorValue(ULTRASONIC_IN)) * 180 / PI)); // Calculate the angle of robot relative to
wall
                reverse(5);              // Backup 5cm
                turn_right(degrees, 60); // Turn the calculated degrees
        }
        else{ // If type 3
                float degrees = 0.5 * (90 - (asin(robot_width /
SensorValue(ULTRASONIC_IN)) * 180 / PI)); // Calculate the angle of robot relative to
wall
                reverse(5);              // Backup 5cm
                turn_left(degrees, 60);  // Turn the calculated degrees
        }
}

enum T_system_state{
        WAITING = 0,
        FINDING_TARGET,
        MOVING_TO_TARGET,
        UPDATING_ALIGNMENT,
        APPROACHING_TARGET,
        RELEASING_BALL,
        EXITING_AREA,
        COMPLETE};
        // Names for each of the states

task main(){
        T_system_state system_state;
        while(true){
                switch(system_state){
                        case(WAITING):
                            SensorValue(GREEN_LED) = 0;
                            if(SensorValue(START) != 0){
                                if(detect_IR() > 1500){
                                  system_state = APPROACHING_TARGET;
                           }
                           else{
                               system_state = FINDING_TARGET;  // Set system state to
waiting
                           }
                            }
                            break;
                        case(FINDING_TARGET):
                            find_IR();                          // Align with the target
                            system_state = MOVING_TO_TARGET;
                            break;
                        case(MOVING_TO_TARGET):
                            move_forward(75);                   // Move forward 75cm or
until an object is reached
                            if(detect_IR() > 1000){             // If a significant IR
signal is received
                                system_state = APPROACHING_TARGET; // Approach the target
                            }
                            else{
```

```
                            system_state = UPDATING_ALIGNMENT; // Otherwise, realign
with the target
                        }
                        break;
                    case(UPDATING_ALIGNMENT):
                        update_alignment();               // Ensure the robot is
facing the target
                        system_state  = MOVING_TO_TARGET;
                        break;
                    case(APPROACHING_TARGET):
                        optimal_distance();               // Get the robot in
position to place the ball
                        system_state = RELEASING_BALL;
                        break;
                    case(RELEASING_BALL):
                        drop_ball();                      // Release the ball
                        system_state = EXITING_AREA;
                        break;
                    case(EXITING_AREA):
                        exit_area();                      // Exit the search area
                        system_state = COMPLETE;
                        break;
                    case(COMPLETE):
                        SensorValue(GREEN_LED) = 1;       // Turn on the LED
                        if(SensorValue(START) != 0){      // If the button is
pressed again, return to the waiting state
                            delay(1000);
                            system_state = WAITING;
                        }
                        break;
                    default:
                }
            }
}
```

Design 3 Program

```
#pragma config(Sensor, in1, infraC, sensorReflection)
#pragma config(Sensor, dgtl1, RedLED, sensorDigitalOut)
#pragma config(Sensor, dgtl4, StopButton, sensorDigitalIn)
#pragma config(Sensor, dgtl5, SonarIn, sensorSONAR_cm)
#pragma config(Sensor, dgtl7, GreenLED, sensorDigitalOut)
#pragma config(Sensor, dgtl9, LimitSwitchR, sensorDigitalIn)
#pragma config(Sensor, dgtl10, LimitSwitchL, sensorDigitalIn)
#pragma config(Sensor, dgtl11, StartButton, sensorDigitalIn)
#pragma config(Motor, port2, upperMotor, tmotorVex393_MC29, openLoop)
#pragma config(Motor, port1, rightMotor, tmotorVex393_HBridge, openLoop, reversed)
#pragma config(Motor, port10, leftMotor, tmotorVex393_HBridge, openLoop)

const int motorSpeed = 50;
int rightMotorSpeed = motorSpeed;
int leftMotorSpeed = motorSpeed;
const int thresholdSensorValue = 1000;
int ON = 1;

void Stop()
{
   motor[leftMotor] = 0;
   motor[rightMotor] = 0;
}
void moveAround()
{
   motor[leftMotor] = rightMotorSpeed;
   motor[rightMotor] = -leftMotorSpeed;
}
void moveTo()
{
   motor[rightMotor] = rightMotorSpeed;
   motor[leftMotor] = leftMotorSpeed;
}
void moveToSlow()
{
   motor[rightMotor] = rightMotorSpeed - 20;
   motor[leftMotor] = leftMotorSpeed - 20;
}
void moveAroundSlowRight()
{
   motor[rightMotor] = -rightMotorSpeed + 20;
   motor[leftMotor] = leftMotorSpeed - 20;
```

```
   wait1Msec(200);
   Stop();
}
void moveAroundSlowLeft()
{
   motor[rightMotor] = +rightMotorSpeed - 20;
   motor[leftMotor] = -leftMotorSpeed + 20;
   wait1Msec(200);
   Stop();

}
void moveBack60()
{
   motor[rightMotor] = -rightMotorSpeed;
   motor[leftMotor] = -leftMotorSpeed;
   wait1Msec(1000);
   Stop();
}
void objectPlacement()
{
   /*MAIN*/
   motor[upperMotor] = -15;
   wait1Msec(4000);
   motor[upperMotor] = 0;
   wait1Msec(5000);
}
void objectPlacementReverse()
{
   /*MAIN*/
   motor[upperMotor] = 15;
   wait1Msec(2000);
   motor[upperMotor] = 0;
}
int min(int a, int b)
{
   if (a < b) {return a;}
   else {return b;}
}

int smallestSignal10(int SignalOne, int SignalTwo, int SignalThree, int SignalFour, int
SignalFive, int SignalSix, int SignalSeven, int SignalEight, int SignalNine, int SignalTen)
{
   int smallest = SignalOne;
   if (SignalTwo < smallest) {smallest = SignalTwo;}
```

```
   if (SignalThree < smallest) {smallest = SignalThree;}
   if (SignalFour < smallest) {smallest = SignalFour;}
   if (SignalFive < smallest) {smallest = SignalFive;}
   if (SignalSix < smallest) {smallest = SignalSix;}
   if (SignalSeven < smallest) {smallest = SignalSeven;}
   if (SignalEight < smallest) {smallest = SignalEight;}
   if (SignalNine < smallest) {smallest = SignalNine;}
   if (SignalTen < smallest) {smallest = SignalNine;}
   return smallest;
}

int smallestSignal5(int SignalOne, int SignalTwo, int SignalThree, int SignalFour, int
SignalFive)
{
   int smallest = SignalOne;
   if (SignalTwo < smallest) {smallest = SignalTwo;}
   if (SignalThree < smallest) {smallest = SignalThree;}
   if (SignalFour < smallest) {smallest = SignalFour;}
   if (SignalFive < smallest) {smallest = SignalFive;}

   return smallest;
}

void signalCheck()
{
   while (1 == 1) // re-align
   {
       // take signal 0
       int sensorValue1 = SensorValue[infraC];
       wait1Msec(51);
       int sensorValue2 = SensorValue[infraC];
       int signalMain = min(sensorValue1, sensorValue2);

       // take signal -1
       moveAroundSlowRight();
       sensorValue1 = SensorValue[infraC];
       wait1Msec(51);
       sensorValue2 = SensorValue[infraC];
       int signalRightOne = min(sensorValue1, sensorValue2);

       // take signal -2
       moveAroundSlowRight();
       sensorValue1 = SensorValue[infraC];
       wait1Msec(51);
```

```
sensorValue2 = SensorValue[infraC];
int signalRightTwo = min(sensorValue1, sensorValue2);

// take signal -3
moveAroundSlowRight();
sensorValue1 = SensorValue[infraC];
wait1Msec(51);
sensorValue2 = SensorValue[infraC];
int signalRightThree = min(sensorValue1, sensorValue2);

// take signal -4
moveAroundSlowRight();
sensorValue1 = SensorValue[infraC];
wait1Msec(51);
sensorValue2 = SensorValue[infraC];
int signalRightFour = min(sensorValue1, sensorValue2);


// take signal -5
moveAroundSlowRight();
sensorValue1 = SensorValue[infraC];
wait1Msec(51);
sensorValue2 = SensorValue[infraC];
int signalRightFive = min(sensorValue1, sensorValue2);


// take signal -6
moveAroundSlowRight();
sensorValue1 = SensorValue[infraC];
wait1Msec(51);
sensorValue2 = SensorValue[infraC];
int signalRightSix = min(sensorValue1, sensorValue2);

// take signal -7
moveAroundSlowRight();
sensorValue1 = SensorValue[infraC];
wait1Msec(51);
sensorValue2 = SensorValue[infraC];
int signalRightSeven = min(sensorValue1, sensorValue2);

// take signal -8
moveAroundSlowRight();
sensorValue1 = SensorValue[infraC];
wait1Msec(51);
```

```
sensorValue2 = SensorValue[infraC];
int signalRightEight = min(sensorValue1, sensorValue2);

// take signal -9
moveAroundSlowRight();
sensorValue1 = SensorValue[infraC];
wait1Msec(51);
sensorValue2 = SensorValue[infraC];
int signalRightNine = min(sensorValue1, sensorValue2);

// take signal -10
moveAroundSlowRight();
sensorValue1 = SensorValue[infraC];
wait1Msec(51);
sensorValue2 = SensorValue[infraC];
int signalRightTen = min(sensorValue1, sensorValue2);

// take signal -11
moveAroundSlowRight();
sensorValue1 = SensorValue[infraC];
wait1Msec(51);
sensorValue2 = SensorValue[infraC];
int signalRightEleven = min(sensorValue1, sensorValue2);

// take signal -12
moveAroundSlowRight();
sensorValue1 = SensorValue[infraC];
wait1Msec(51);
sensorValue2 = SensorValue[infraC];
int signalRightTwelve = min(sensorValue1, sensorValue2);

// take signal -13
moveAroundSlowRight();
sensorValue1 = SensorValue[infraC];
wait1Msec(51);
sensorValue2 = SensorValue[infraC];
int signalRightThirteen = min(sensorValue1, sensorValue2);

// take signal -14
moveAroundSlowRight();
sensorValue1 = SensorValue[infraC];
wait1Msec(51);
sensorValue2 = SensorValue[infraC];
int signalRightFourteen = min(sensorValue1, sensorValue2);
```

```
// take signal -15
moveAroundSlowRight();
sensorValue1 = SensorValue[infraC];
wait1Msec(51);
sensorValue2 = SensorValue[infraC];
int signalRightFifteen = min(sensorValue1, sensorValue2);

// take signal -16
moveAroundSlowRight();
sensorValue1 = SensorValue[infraC];
wait1Msec(51);
sensorValue2 = SensorValue[infraC];
int signalRightSixteen = min(sensorValue1, sensorValue2);

// take signal -17
moveAroundSlowRight();
sensorValue1 = SensorValue[infraC];
wait1Msec(51);
sensorValue2 = SensorValue[infraC];
int signalRightSeventeen = min(sensorValue1, sensorValue2);

// take signal -18
moveAroundSlowRight();
sensorValue1 = SensorValue[infraC];
wait1Msec(51);
sensorValue2 = SensorValue[infraC];
int signalRightEighteen = min(sensorValue1, sensorValue2);

// take signal -19
moveAroundSlowRight();
sensorValue1 = SensorValue[infraC];
wait1Msec(51);
sensorValue2 = SensorValue[infraC];
int signalRightNineteen = min(sensorValue1, sensorValue2);

// take signal -20
moveAroundSlowRight();
sensorValue1 = SensorValue[infraC];
wait1Msec(51);
sensorValue2 = SensorValue[infraC];
int signalRightTwenty = min(sensorValue1, sensorValue2);

// take signal 1
```

```
int countBack = -20;
while (countBack!=1)
{
    moveAroundSlowLeft();
    countBack++;
}
sensorValue1 = SensorValue[infraC];
wait1Msec(51);
sensorValue2 = SensorValue[infraC];
int signalLeftOne = min(sensorValue1, sensorValue2);

// take signal 2
moveAroundSlowLeft();
sensorValue1 = SensorValue[infraC];
wait1Msec(51);
sensorValue2 = SensorValue[infraC];
int signalLeftTwo = min(sensorValue1, sensorValue2);

// take signal 3
moveAroundSlowLeft();
sensorValue1 = SensorValue[infraC];
wait1Msec(51);
sensorValue2 = SensorValue[infraC];
int signalLeftThree = min(sensorValue1, sensorValue2);

// take signal 4
moveAroundSlowLeft();
sensorValue1 = SensorValue[infraC];
wait1Msec(51);
sensorValue2 = SensorValue[infraC];
int signalLeftFour = min(sensorValue1, sensorValue2);

// take signal 5
moveAroundSlowLeft();
sensorValue1 = SensorValue[infraC];
wait1Msec(51);
sensorValue2 = SensorValue[infraC];
int signalLeftFive = min(sensorValue1, sensorValue2);

// take signal 6
moveAroundSlowLeft();
sensorValue1 = SensorValue[infraC];
wait1Msec(51);
sensorValue2 = SensorValue[infraC];
```

```
int signalLeftSix = min(sensorValue1, sensorValue2);

// take signal 7
moveAroundSlowLeft();
sensorValue1 = SensorValue[infraC];
wait1Msec(51);
sensorValue2 = SensorValue[infraC];
int signalLeftSeven = min(sensorValue1, sensorValue2);

// take signal 8
moveAroundSlowLeft();
sensorValue1 = SensorValue[infraC];
wait1Msec(51);
sensorValue2 = SensorValue[infraC];
int signalLeftEight = min(sensorValue1, sensorValue2);

// take signal 9
moveAroundSlowLeft();
sensorValue1 = SensorValue[infraC];
wait1Msec(51);
sensorValue2 = SensorValue[infraC];
int signalLeftNine = min(sensorValue1, sensorValue2);

// take signal 10
moveAroundSlowLeft();
sensorValue1 = SensorValue[infraC];
wait1Msec(51);
sensorValue2 = SensorValue[infraC];
int signalLeftTen = min(sensorValue1, sensorValue2);

// take signal 11
moveAroundSlowLeft();
sensorValue1 = SensorValue[infraC];
wait1Msec(51);
sensorValue2 = SensorValue[infraC];
int signalLeftEleven = min(sensorValue1, sensorValue2);

// take signal 12
moveAroundSlowLeft();
sensorValue1 = SensorValue[infraC];
wait1Msec(51);
sensorValue2 = SensorValue[infraC];
int signalLeftTwelve = min(sensorValue1, sensorValue2);
```

```
// take signal 13
moveAroundSlowLeft();
sensorValue1 = SensorValue[infraC];
wait1Msec(51);
sensorValue2 = SensorValue[infraC];
int signalLeftThirteen = min(sensorValue1, sensorValue2);

// take signal 14
moveAroundSlowLeft();
sensorValue1 = SensorValue[infraC];
wait1Msec(51);
sensorValue2 = SensorValue[infraC];
int signalLeftFourteen = min(sensorValue1, sensorValue2);

// take signal 15
moveAroundSlowLeft();
sensorValue1 = SensorValue[infraC];
wait1Msec(51);
sensorValue2 = SensorValue[infraC];
int signalLeftFifteen = min(sensorValue1, sensorValue2);

// take signal 16
moveAroundSlowLeft();
sensorValue1 = SensorValue[infraC];
wait1Msec(51);
sensorValue2 = SensorValue[infraC];
int signalLeftSixteen = min(sensorValue1, sensorValue2);

// take signal 17
moveAroundSlowLeft();
sensorValue1 = SensorValue[infraC];
wait1Msec(51);
sensorValue2 = SensorValue[infraC];
int signalLeftSeventeen = min(sensorValue1, sensorValue2);

// take signal 18
moveAroundSlowLeft();
sensorValue1 = SensorValue[infraC];
wait1Msec(51);
sensorValue2 = SensorValue[infraC];
int signalLeftEighteen = min(sensorValue1, sensorValue2);

// take signal 19
moveAroundSlowLeft();
```

```
        sensorValue1 = SensorValue[infraC];
        wait1Msec(51);
        sensorValue2 = SensorValue[infraC];
        int signalLeftNineteen = min(sensorValue1, sensorValue2);

        // take signal 20
        moveAroundSlowLeft();
        sensorValue1 = SensorValue[infraC];
        wait1Msec(51);
        sensorValue2 = SensorValue[infraC];
        int signalLeftTwenty = min(sensorValue1, sensorValue2);

        // now move to 0
        countBack = 20;
        while (countBack != 0)
        {
            moveAroundSlowRight();
            countBack--;
        }

        // calculate the smallest signal
        int signal1 = smallestSignal10 (signalRightOne, signalRightTwo, signalRightThree,
signalRightFour, signalRightFive, signalRightSix, signalRightSeven, signalRightEight,
signalRightNine, signalRightTen);
        int signal2 = smallestSignal10 (signalLeftOne, signalLeftTwo, signalLeftThree,
signalLeftFour, signalLeftFive, signalLeftSix, signalLeftSeven, signalLeftEight,
signalLeftNine, signalLeftTen);
        int signal3 = smallestSignal10 (signalRightEleven, signalRightTwelve,
signalRightThirteen, signalRightFourteen, signalRightFifteen, signalRightSixteen,
signalRightSeventeen, signalRightEighteen, signalRightNineteen, signalRightTwenty);
        int signal4 = smallestSignal10 (signalLeftEleven, signalLeftTwelve,
signalLeftThirteen, signalLeftFourteen, signalLeftFifteen, signalLeftSixteen,
signalLeftSeventeen, signalLeftEighteen, signalLeftNineteen, signalLeftTwenty);
        int signal5 = signalMain;
        int signal = smallestSignal5(signal1, signal2,signal3, signal4, signal5);

        if (signal == signalRightOne)
        {
            moveAroundSlowRight();
            break;
        }
        if (signal == signalRightTwo)
        {
            moveAroundSlowRight();
```

```
    moveAroundSlowRight();
    break;
}
if (signal == signalRightThree)
{
    moveAroundSlowRight();
    moveAroundSlowRight();
    moveAroundSlowRight();
    break;
}
if (signal == signalRightFour)
{
    moveAroundSlowRight();
    moveAroundSlowRight();
    moveAroundSlowRight();
    moveAroundSlowRight();
    break;
}
if (signal == signalRightFive)
{
    countBack=0;
    while (countBack!=5)
    {
        moveAroundSlowRight();
        countBack++;
    }
    break;
}
if (signal == signalRightSix)
{
    countBack = 0;
    while (countBack != 6)
    {
        moveAroundSlowRight();
        countBack++;
    }
    break;
}
if (signal == signalRightSeven)
{
    countBack = 0;
    while (countBack != 7)
    {
        moveAroundSlowRight();
```

```
        countBack++;
    }
    break;
}
if (signal == signalRightEight)
{
    countBack = 0;
    while (countBack != 8)
    {
        moveAroundSlowRight();
        countBack++;
    }
    break;
}
if (signal == signalRightNine)
{
    countBack = 0;
    while (countBack != 9)
    {
        moveAroundSlowRight();
        countBack++;
    }
    break;
}
if (signal == signalRightTen)
{
    countBack = 0;
    while (countBack != 10)
    {
        moveAroundSlowRight();
        countBack++;
    }
    break;
}
if (signal == signalRightEleven)
{
    countBack = 0;
    while (countBack != 11)
    {
        moveAroundSlowRight();
        countBack++;
    }
    break;
}
```

```
if (signal == signalRightTwelve)
{
    countBack = 0;
    while (countBack != 12)
    {
        moveAroundSlowRight();
        countBack++;
    }
    break;
}
if (signal == signalRightThirteen)
{
    countBack = 0;
    while (countBack != 13)
    {
        moveAroundSlowRight();
        countBack++;
    }
    break;
}
if (signal == signalRightFourteen)
{
    countBack = 0;
    while (countBack != 14)
    {
        moveAroundSlowRight();
        countBack++;
    }
    break;
}
if (signal == signalRightFifteen)
{
    countBack = 0;
    while (countBack != 15)
    {
        moveAroundSlowRight();
        countBack++;
    }
    break;
}
if (signal == signalRightSixteen)
{
    countBack = 0;
    while (countBack != 16)
```

```
    {
        moveAroundSlowRight();
        countBack++;
    }
    break;
}
if (signal == signalRightSeventeen)
{
    countBack = 0;
    while (countBack != 17)
    {
        moveAroundSlowRight();
        countBack++;
    }
    break;
}
if (signal == signalRightEighteen)
{
    countBack = 0;
    while (countBack != 18)
    {
        moveAroundSlowRight();
        countBack++;
    }
    break;
}
if (signal == signalRightNineteen)
{
    countBack = 0;
    while (countBack != 19)
    {
        moveAroundSlowRight();
        countBack++;
    }
    break;
}
if (signal == signalRightTwenty)
{
    countBack = 0;
    while (countBack != 20)
    {
        moveAroundSlowRight();
        countBack++;
    }
```

```
    break;
}
if (signal == signalLeftOne)
{
    moveAroundSlowLeft();
    break;
}
if (signal == signalLeftTwo)
{
    moveAroundSlowLeft();
    moveAroundSlowLeft();
    break;
}
if (signal == signalLeftThree)
{
    moveAroundSlowLeft();
    moveAroundSlowLeft();
    moveAroundSlowLeft();
    break;
}
if (signal == signalLeftFour)
{
    moveAroundSlowLeft();
    moveAroundSlowLeft();
    moveAroundSlowLeft();
    moveAroundSlowLeft();
    break;
}
if (signal == signalLeftFive)
{
    countBack = 0;
    while (countBack != 5)
    {
        moveAroundSlowLeft();
        countBack++;
    }
    break;
}
if (signal == signalLeftSix)
{
    countBack = 0;
    while (countBack != 6)
    {
        moveAroundSlowLeft();
```

```
        countBack++;
    }
    break;
}
if (signal == signalLeftSeven)
{
    countBack = 0;
    while (countBack != 7)
    {
        moveAroundSlowLeft();
        countBack++;
    }
    break;
}
if (signal == signalLeftEight)
{
    countBack = 0;
    while (countBack != 8)
    {
        moveAroundSlowLeft();
        countBack++;
    }
    break;
}
if (signal == signalLeftNine)
{
    countBack = 0;
    while (countBack != 9)
    {
        moveAroundSlowLeft();
        countBack++;
    }
    break;
}
if (signal == signalLeftTen)
{
    countBack = 0;
    while (countBack != 10)
    {
        moveAroundSlowLeft();
        countBack++;
    }
    break;
}
```

```
if (signal == signalLeftEleven)
{
    countBack = 0;
    while (countBack != 11)
    {
        moveAroundSlowLeft();
        countBack++;
    }
    break;
}
if (signal == signalLeftTwelve)
{
    countBack = 0;
    while (countBack != 12)
    {
        moveAroundSlowLeft();
        countBack++;
    }
    break;
}
if (signal == signalLeftThirteen)
{
    countBack = 0;
    while (countBack != 13)
    {
        moveAroundSlowLeft();
        countBack++;
    }
    break;
}
if (signal == signalLeftFourteen)
{
    countBack = 0;
    while (countBack != 14)
    {
        moveAroundSlowLeft();
        countBack++;
    }
    break;
}
if (signal == signalLeftFifteen)
{
    countBack = 0;
    while (countBack != 15)
```

```
    {
        moveAroundSlowLeft();
        countBack++;
    }
    break;
}
if (signal == signalLeftSixteen)
{
    countBack = 0;
    while (countBack != 16)
    {
        moveAroundSlowLeft();
        countBack++;
    }
    break;
}
if (signal == signalLeftSeventeen)
{
    countBack = 0;
    while (countBack != 17)
    {
        moveAroundSlowLeft();
        countBack++;
    }
    break;
}
if (signal == signalLeftEighteen)
{
    countBack = 0;
    while (countBack != 18)
    {
        moveAroundSlowLeft();
        countBack++;
    }
    break;
}
if (signal == signalLeftNineteen)
{
    countBack = 0;
    while (countBack != 19)
    {
        moveAroundSlowLeft();
        countBack++;
    }
```

```
            break;
        }
        if (signal == signalLeftTwenty)
        {
            countBack = 0;
            while (countBack != 20)
            {
                moveAroundSlowLeft();
                countBack++;
            }
            break;
        }
        if (signal == signalMain)
        {
            break;
        }
    }
    return;
}

task main()
{
    int sensorValue1 = 0;
    int sensorValue2 = 0;
    while (1 == 1)
    {
        if (SensorValue[StartButton] == 0)
        {
            while (1 == 1) // spin around find target
            {
                moveAround();
                int sensorValue1 = 0;
                int sensorValue2 = 0;
                sensorValue1 = SensorValue[infraC];
                wait1Msec(51);
                sensorValue2 = SensorValue[infraC];
                int signal = min(sensorValue1, sensorValue2);
                if (signal < thresholdSensorValue)
                {
                    SensorValue(RedLED) = ON;
                    Stop();
                    wait1Msec(1000);
                    break;
                }
```

```
            }
            // move to target till off path
            while ((((SensorValue[SonarIn] > 80) && (SensorValue[SonarIn] != -1))))
            {
                moveTo();
            }
            Stop();
            wait1Msec(1000);
            signalCheck();
            // move to target till off path
            while ((SensorValue[SonarIn] > 30) && (SensorValue[SonarIn] != -1))
            {
                moveTo();
            }
            Stop();
            wait1Msec(1000);
            signalCheck();
            while (1 == 1) // go to target
            {
                moveToSlow();
                int sensorValue1 = SensorValue[SonarIn];
                if ((sensorValue1 <= 3) && (sensorValue1 != -1))
                {
                    Stop();
                    break;
                }
            }
            objectPlacement();
            objectPlacementReverse();
            Stop();
            wait1Msec(500);
            moveBack60();
            Stop();
            SensorValue(RedLED) = ON;
        }
    }
}
```