Push Bot

Author: Neima Yeganeh

Advisor: Dr. John Seng

Senior Project, Spring 2020

Table of Contents

Problem Statement	**2**
Software	**3**
Hardware	**7**
Mechanical	**11**
Budget and Bill of Materials	**13**
Lessons Learned	**15**
Conclusion	**15**
#Code	**16**

Introduction

Every Spring, Cal Poly holds their annual robotics competition, Roborodentia. This is a head-to-head autonomous competition, where students build their robots and face off in a predetermined 1v1 challenge. Each year the challenge changes, and this year's challenge consisted of collecting and stacking Purina cat food cans.

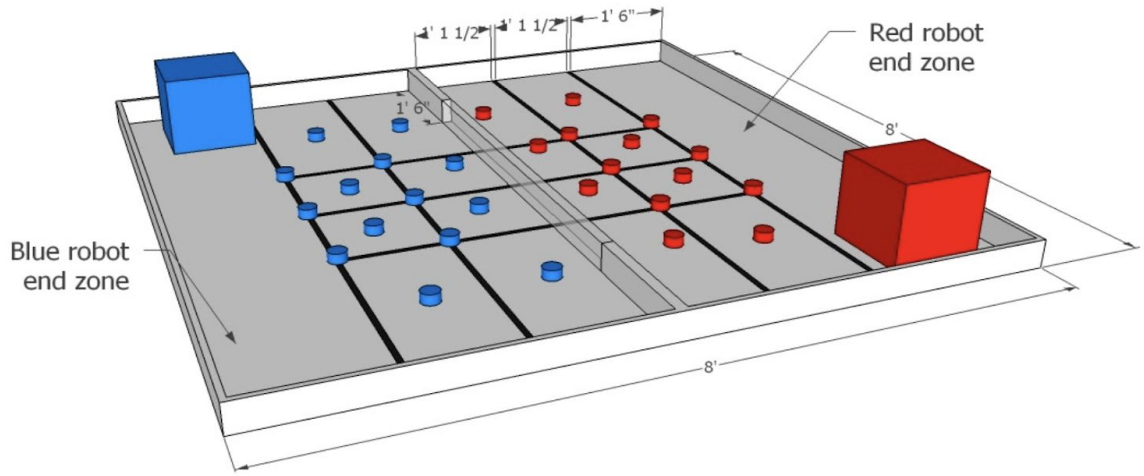
This project was originally intended to compete in the 2020 Roborodentia competition, but the event was canceled due to COVID-19. Due to pandemic-related constraints, a lot of our original robot plans were scrapped. The robot was originally designed to *stack* cans, but since our access to resources became very limited very quickly, we only produced the drive train to *push* the cans into the endzone.

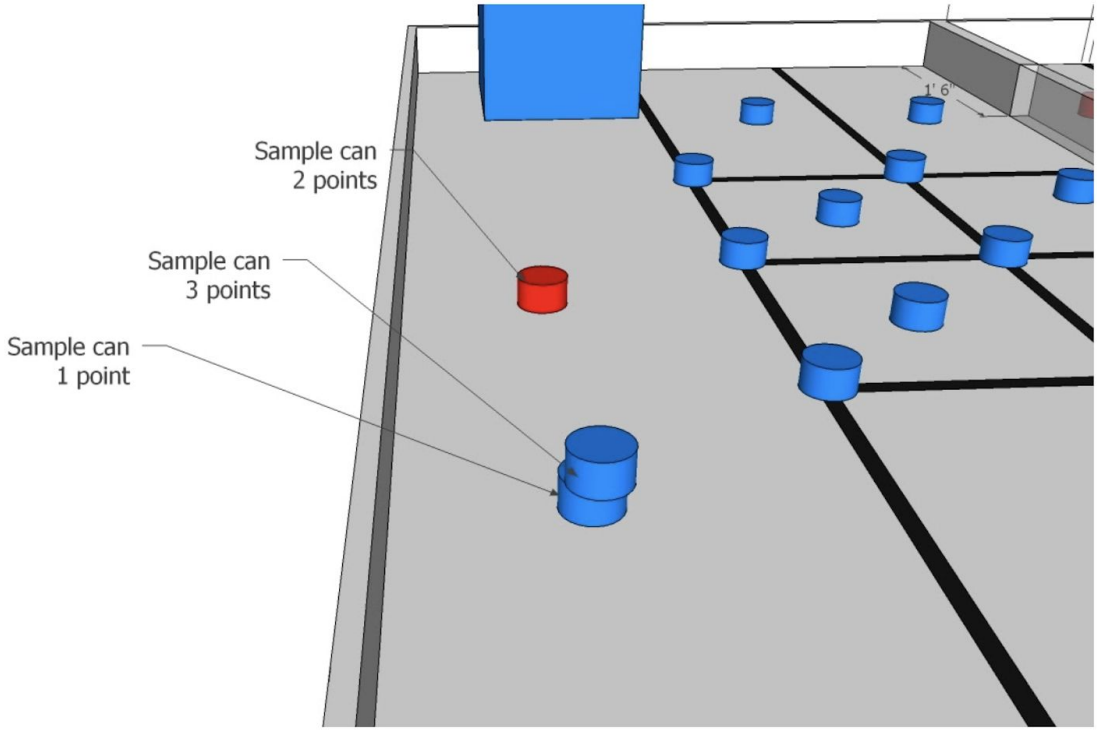
Problem Statement

This year's challenge was to collect and stack Purina cat food cans to gain points. Points were scored based on how many cans were in your endzone, and how high they were stacked by the end of the match. Each can in the endzone was worth 1 point if unstacked. Stacked cans were awarded more points based on how high they were stacked. Each stacked can was worth 2 more points than the one underneath it, so if you have a stack of three you would get 9 points: 1 point for the bottom can, 3 for the middle can, and 5 for the top can. This increase in points gives the competitors high incentive to not only push cans into their endzone but to stack them as well. Each match lasts 3 minutes.

The playing field is an 8’ x 8’ grid with a divider down the middle. Each robot (team) is responsible for the cans on their own side for the first chunk of time (exact number not listed on the competition specification), at which point the divider is lifted so the robots can collect cans from the other team’s side for the remainder of the match.

There were rules for the robot itself, too. The robot must be fully autonomous and self contained. It also must be no bigger than 12” x 12” at the beginning of the match, and may expand autonomously to no larger than 14” x 14” during the match, with no height limit. There are also rules regarding what kind of equipment the robots cannot use, such as RF wireless receivers/transmitters.





Software

I developed the software for this robot using MicroPython, a language commonly used for robotics. While the MicroPython has the syntax of Python for ease of use, it

has great low-level control since the language itself was written in C. This worked out well for me because the ease of use aspect minimized the number of troubling bugs that were produced.

My code is broken down into three main classes along with a main file. A motor class, to interface with the motors; an IR class, to interface with the infrared sensors; and a line class to combine the two other classes and produce movement depending on the environment and location of the bot.

Motor Class

The Motor class has three main functions and stores information on which pins are being used, along with the timer channel info. The three functions are enable, to enable to motor; disable, to disable the motor; and set_duty to set the PWM output on the motor. For this robot we have two motors so we have two different motor class instances.

IR Class

The IR class has three functions and stores the 4 different output pins. The three functions are update, which reads the sensor values and returns a tuple of boolean values; up, which takes in the tuple from update and returns a number corresponding with a direction to move on the endzone line; and upMiddle, the same as up but for when the robot is on the middle line.

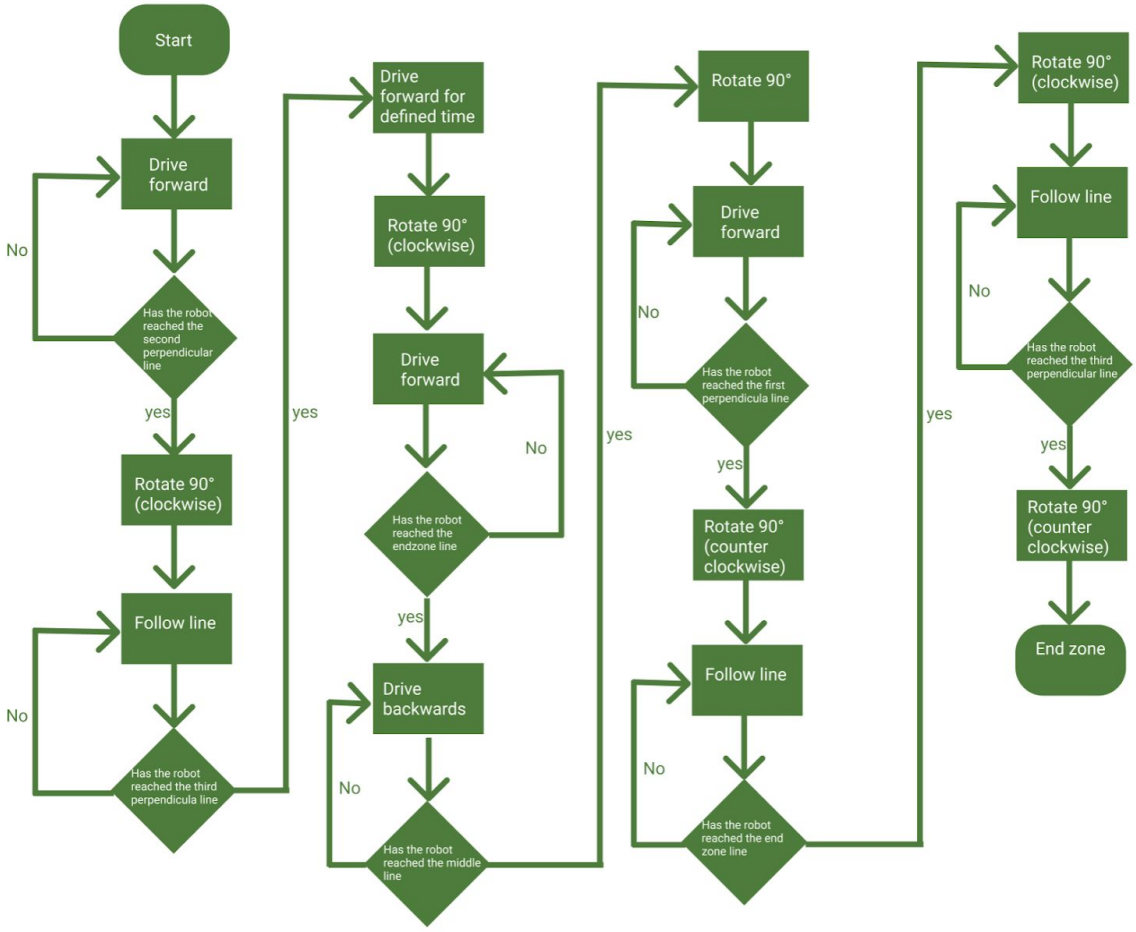
Line Class

The Line class has five functions and stores an instance of the infrared object and the two motor objects. The five functions are findLine, rotateStart, runLine and driveForward. FindLine takes in the number of lines to drive passed and the direction and moves the robot straight in the direction until it passes the number of lines specified. RotateStart spins the motors in the given direction for 90°. RunLine takes in the number of lines to pass and the type of line. It then uses the infrared sensors to detect the line and drive until passing the specified number of lines. DriveForward rotates the motors forward for a predetermined amount of time.

Main

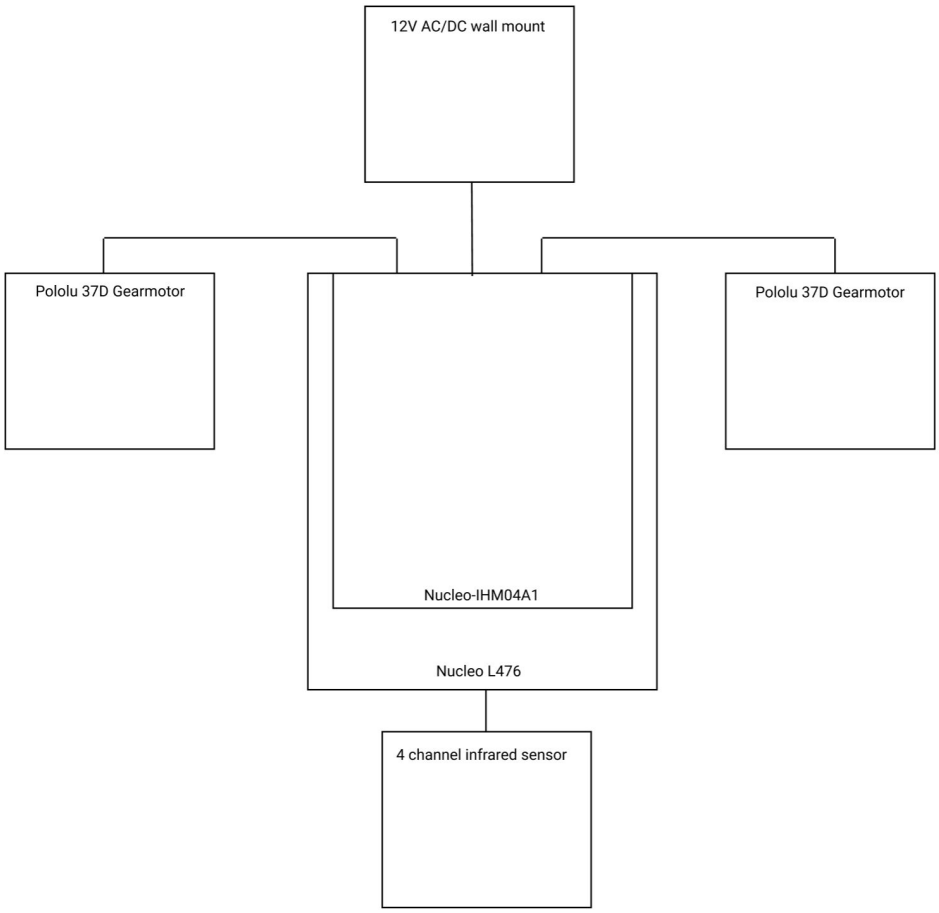
The main file is where the main function is run. It first sets up the motors and infrared sensor and line classes, then runs each function, one at a time in the path order. It does not loop.

```
def main():
    #setup
    pin_EN = pyb.Pin (pyb.Pin.cpu.A10, pyb.Pin.OUT_PP)
    pin_IN1 = pyb.Pin (pyb.Pin.cpu.B4, pyb.Pin.OUT_PP)
    pin_IN2 = pyb.Pin (pyb.Pin.cpu.B5, pyb.Pin.OUT_PP)
    tim = pyb.Timer(3, freq = 20000)
    pin_EN2 = pyb.Pin (pyb.Pin.cpu. C1, pyb.Pin.OUT_PP)
    pin_IN12 = pyb.Pin (pyb.Pin.cpu.A0, pyb.Pin.OUT_PP)
    pin_IN22 = pyb.Pin (pyb.Pin.cpu.A1, pyb.Pin.OUT_PP)
    tim2 = pyb.Timer(5, freq = 20000)
    moe1 = MotorDriver(pin_EN, pin_IN1, pin_IN2, tim)
    moe2 = MotorDriver(pin_EN2, pin_IN12, pin_IN22, tim2)
    in_1CA = pyb.Pin (pyb.Pin.cpu.C3)
    pin_1CB = pyb.Pin (pyb.Pin.cpu.A4)
    pin_2CA = pyb.Pin (pyb.Pin.cpu.C2)
    pin_2CB = pyb.Pin (pyb.Pin.cpu.C0)
    i = IR(pin_1CA, pin_1CB, pin_2CA, pin_2CB)
    l = Line(i, moe1, moe2)
    #run
    l.findLine(2, 1)
    l.rotateStart(-1)
    l.runLine(3, 0)
    l.driveForward()
    l.rotateStart(-1)
    l.findLine(1, 1)
    l.findLine(1, -1)
    l.rotateStart(-1)
    l.runLine(1, 0)
    l.rotateStart(1)
    l.runLine(1, 0)
    l.rotateStart(-1)
    l.runLine(2, 1)
    l.rotateStart(-1)
if __name__ == "__main__":
    main()
```



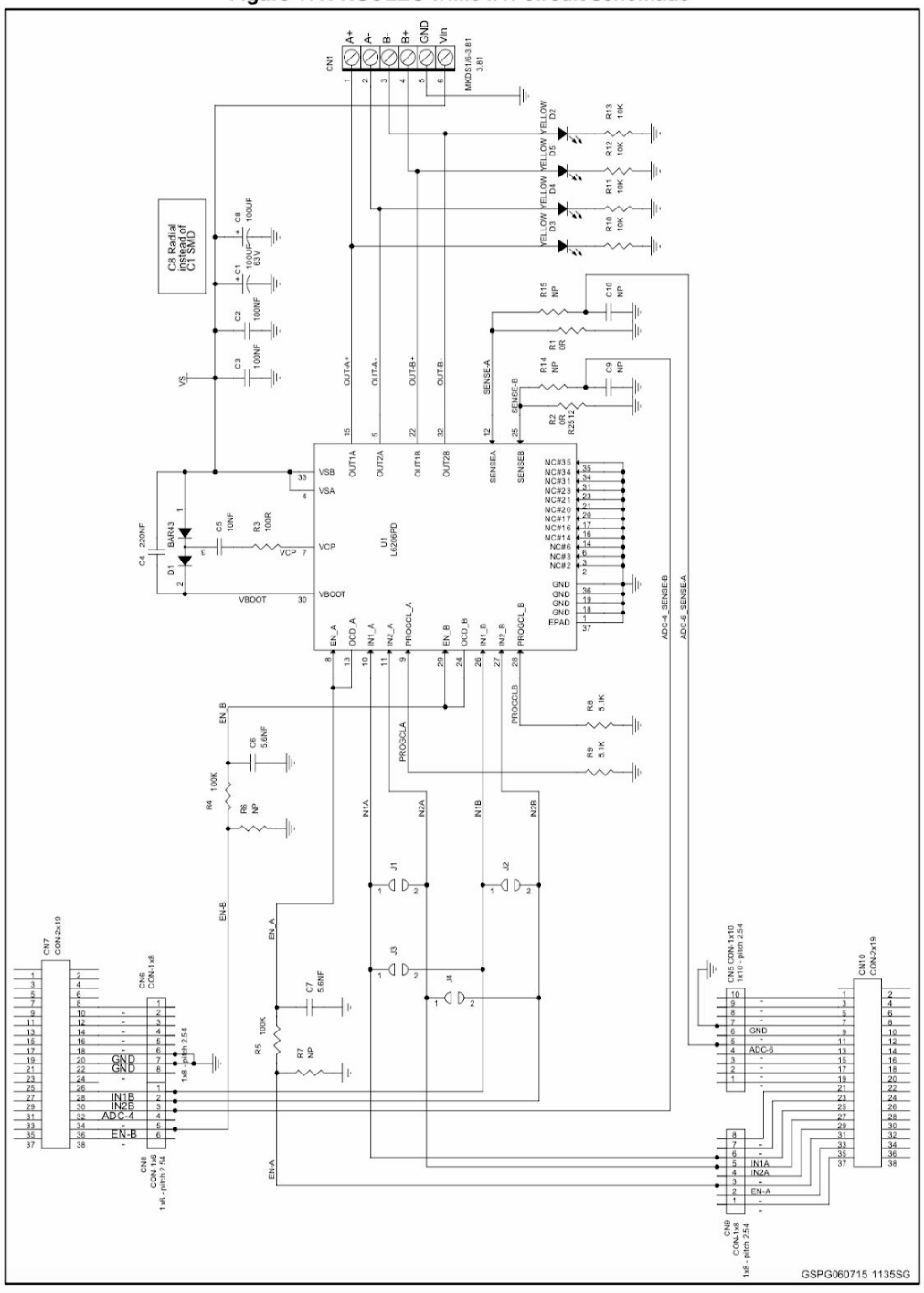
In order to locate cans to score, our robot is programmed to drive forward until the IR sensors sense the second perpendicular line. Once it arrives at the second line, it rotates 90 degrees towards the center of the field. It then follows that line, counting each intersection of perpendicular lines to know where it is currently at. Once it's at the third and final intersection with cans in the middle, the robot moves forward based on a timer so that it can line up with the last line of cans. From here it rotates 90 towards the end zone. Once done rotating the robot drives forward until it senses the end zone line. From here it backs up to leave the first set of cans in the end zone. It backs up until it senses the far line. It then rotates 90 towards the center of the field. It then follows the line until it hits the first intersection where it then rotates again 90 degrees towards the end zone. From here it follows the line until it hits the endzone and then again it turns 90 degrees towards the starting area. From here it collects more cans by following the line and driving into them. Once it senses the third and final intersection it then rotates again towards the end zone, dumping the rest of the cans into the scoring area.

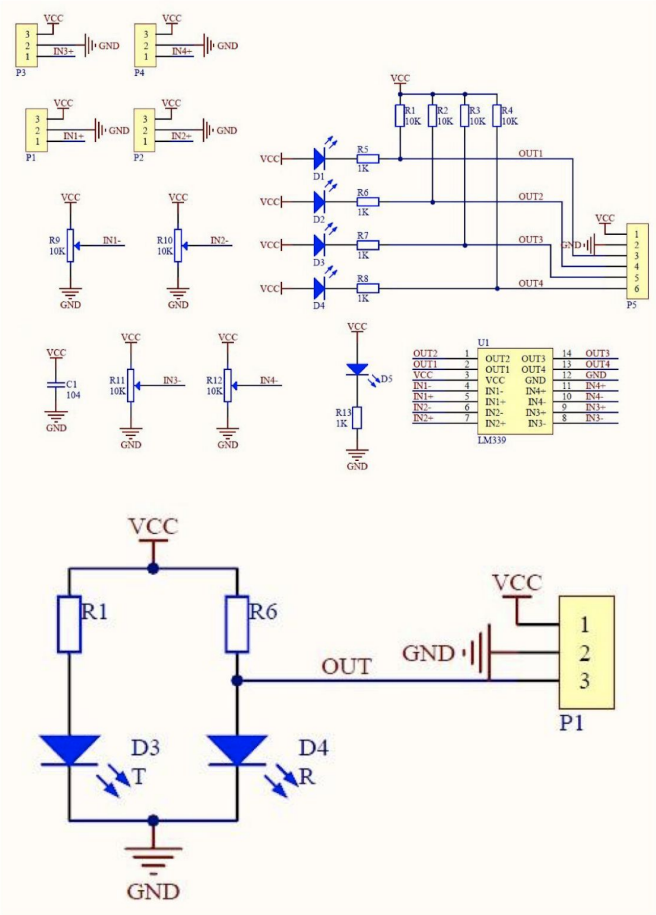
Hardware



We used a Nucleo L476 board flashed with MicroPython. Along with that we use a Nucleo brush DC motor driver expansion board to interface directly with the motors. This board is connected directly on top of the Nucleo L476. The expansion board consisted of two motor drivers. These were used for the drive motors. The expansion board was directly connected to a 12V AC to DC power cord. If this robot was to actually compete I would have used a battery and found a way to direct 12 volts to the expansion board.

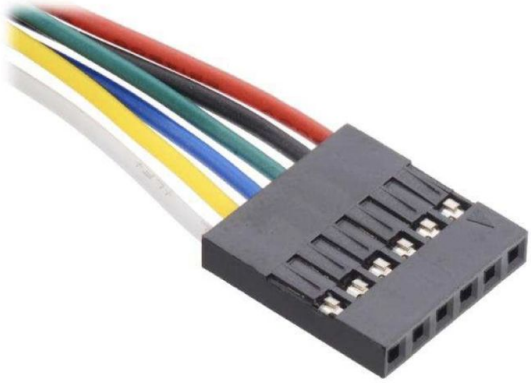
Figure 1: X-NUCLEO-IHM04A1 circuit schematic





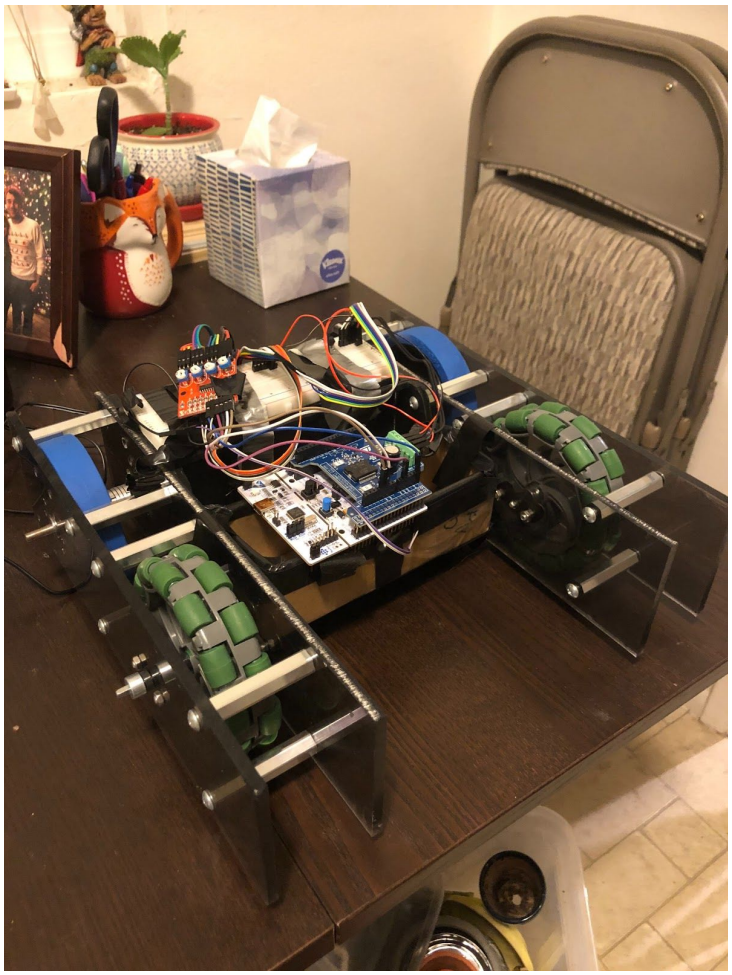
We also used a four channel infrared sensor to detect lines on the field. This sensor had a Vcc, ground and 4 output pins with high/low values. The Vcc accepts a 5V power directly from the Nucleo board. The data sent to the 4 output pins are used to determine what direction to move.

Lead Color	Function
Red	Motor power
Black	Motor power
Green	Encoder ground
Blue	Encoder Vcc (3.5 V to 20 V)
Yellow	Encoder A output
White	Encoder B output

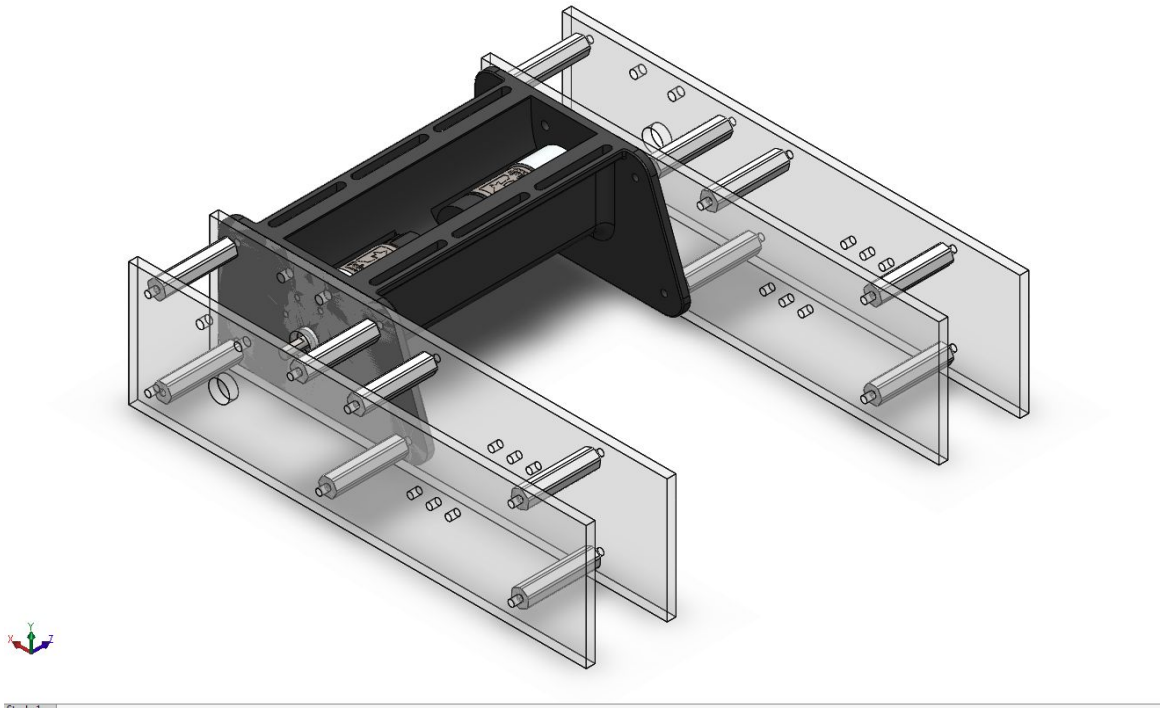


For our motors we use 2 Pololu 37D metal gearmotors with encoders. These motors have 6 different connections to them. Two connections to run the motor, one for the encoder ground, one for the encoder Vcc, and two encoder outputs. For this project we ended up scrapping our code interfacing with the encoders and only used the motor power connections.

Mechanical



The design for the robot was based off of previously seen robots. We designed the drivetrain to be a tank drive so that it could maximize pushing power. This was chosen because after a certain amount of time the middle of the stage would be taken away and the robots could push on each other's side. We wanted to be able to push other robots so we chose to use two BaneBots wheels directly attached to the motor so we could maximize our pushing power. Our front two wheels were OMNI wheels so that we could turn without causing friction on the side of the wheels.



We used polycarbonate as a base structure that was waterjeted to ensure precision. Our sketches were done in SolidWorks. This was done on campus for free at our machine shop. The polycarbonate was .25" thick so that it could survive the beating it was going to take. The polycarbonate was attached to each other to create 2 channels using metal hex spacers. These two channels were attached together by a custom made connector that also housed the motos. This made it easier for us to mount the motors and made everything cleaner. The connector was designed on SolidWorks and 3D printed on my partners 3D printer.

We mounted the Nucleo board on a piece of cardboard that was attached to the middle of the robot. The infrared sensors were attached underneath the connector to make them aligned with our pivot point. We didn't have many tools at our disposal due to the COVID lockdown, so we attached the infrared sensors with electrical tape.

We pushed the cans with the front of our robot. This was not our original plan but because COVID struck we limited our design complexity. This was because of the lack of building tools available to us during this time. We had originally designed a lift system with a claw mechanism to stack the cans. This system was going to use drawer slides as rails and more 3D printed parts to connect everything.

Budget and Bill of Materials

There was no set budget for this project, but our goal was to minimize the total cost. The most expensive part was the VEX OMNI wheels. We decided on these wheels because they would allow the robot to turn with ease. Most of our parts we ordered

straight from retailers (VEX, goBILDA, Amazon), while some others were donated from the ME405 class I took at Cal Poly.

		Info	Quantity	Price
Board/Electronics	Nucleo L476	ME 405	1	$14.60
	Brush DC motor driver	ME 405	1	$13
	Jumper Wire	https://www.amazon.com/gp/product/B07FQJVTNK/ref=ppx_yo_dt_b_asin_title_o09_s00?ie=UTF8&psc=1	1	$6.59
	Power cord	ME 405	1	$14.00
	IR Sensor	https://www.amazon.com/gp/product/B07662JJF7/ref=ppx_yo_dt_b_asin_title_o02_s01?ie=UTF8&psc=1	1	$7.99
Wheels				
	banebot 3 7/8"	http://www.banebots.com/product/T81P-395BA.html	2	2.97
	T81 Hub, 6mm Shaft	http://www.banebots.com/product/T81H-RM61.html	2	3.15
	Ball Bearing	gobilda	2	2.54
	Clamping Shaft Couple	gobilda	2	4.24
	4" Omni-Directional Wheel (2-pack)	https://www.vexrobotics.com/edr-wheels.html	1	27.49
	0.375" OD Nylon Spacer Variety Pack	https://www.vexrobotics.com/edr-wheels.html	1	5.49
	2" & 3" Drive Shaft Pack	https://www.vexrobotics.com/edr-wheels.html	1	5.99
	Star Drive Shaft Collar (16-pack)	https://www.vexrobotics.com/edr-wheels.html	1	8.99
	Flat Bearing (10-pack)	https://www.vexrobotics.com/edr-wheels.html	1	5.49
Motors		Robotics Club	2	15
3D print materiel	1.75 overture petg	amazon	0.5kg	$13
polycarbonate	15"x15"x0.25"	https://www.tapplastics.com/product/plastics/cut_to_size_plastic/polycarbonate_sheets/516	1	$21

			Total	$171.53

Lessons Learned

This project taught me a lot in the last two quarters. From design to planning there were a lot of challenges that came up.

Mechanical

Double check every measurement. This was something we learned the hard way this quarter. From almost miscalculating the size of the channel, to cutting out the wrong pattern for our motor to connect to the polycarbonate, we learned that double and triple checking everything is important to having things fit and connect easily.

Electric

Micro controllers and sensors are very delicate. Learned this one the hard way during finals week. It's important to treat them with the utmost care. It's also important to have spare electronic parts when you are planning to test and use the electronics frequently.

Software

Choosing a language that is not that common makes it harder for troubleshooting online. With only so many people using MicroPython it's hard to find good coding examples and helpful troubleshooting info.

If we were to do this again we would have planned out our build better and allotted more time for coding. We would also order extra sensor parts just in case ours broke.

Conclusion

This project encompassed a lot of different engineering disciplines, from mechanical engineering to electrical engineering and computer science. Our goal was to create a functioning robot to compete in the 2020 Roborodentia. This sadly did not come to fruition since we faced many constraints due to the pandemic, including that the competition was canceled. Even with the cancellation we were still able to build most of the robot that we wanted to build for the competition.

#Code

```
import pyb
from mo import MotorDriver
from line import Line
def main():
    #setup
    pin_EN = pyb.Pin (pyb.Pin.cpu.A10, pyb.Pin.OUT_PP)
    pin_IN1 = pyb.Pin (pyb.Pin.cpu.B4, pyb.Pin.OUT_PP)
    pin_IN2 = pyb.Pin (pyb.Pin.cpu.B5, pyb.Pin.OUT_PP)
    tim = pyb.Timer(3, freq = 20000)
    pin_EN2 = pyb.Pin (pyb.Pin.cpu. C1, pyb.Pin.OUT_PP)
    pin_IN12 = pyb.Pin (pyb.Pin.cpu.A0, pyb.Pin.OUT_PP)
    pin_IN22 = pyb.Pin (pyb.Pin.cpu.A1, pyb.Pin.OUT_PP)
    tim2 = pyb.Timer(5, freq = 20000)
    moe1 = MotorDriver(pin_EN, pin_IN1, pin_IN2, tim)
    moe2 = MotorDriver(pin_EN2, pin_IN12, pin_IN22, tim2)
    in_1CA = pyb.Pin (pyb.Pin.cpu.C3)
    pin_1CB = pyb.Pin (pyb.Pin.cpu.A4)
    pin_2CA = pyb.Pin (pyb.Pin.cpu.C2)
    pin_2CB = pyb.Pin (pyb.Pin.cpu.C0)
    i = IR(pin_1CA, pin_1CB, pin_2CA, pin_2CB)
    l = Line(i, moe1, moe2)
    #run
    l.findLine(2, 1)
    l.rotateStart(-1)
    l.runLine(3, 0)
    l.driveForward()
    l.rotateStart(-1)
    l.findLine(1, 1)
    l.findLine(1, -1)
    l.rotateStart(-1)
    l.runLine(1, 0)
    l.rotateStart(1)
    l.runLine(1, 0)
    l.rotateStart(-1)
    l.runLine(2, 1)
    l.rotateStart(-1)
```

```
if __name__ == "__main__":
    main()

import pyb
class IR:
    def __init__ (self,p1 , p2, p3, p4):
        self.p1 = pyb.ADC(p1)
        self.p2 = pyb.ADC(p2)
        self.p3 = pyb.ADC(p3)
        self.p4 = pyb.ADC(p4)
    def update (self):
        a = self.p1.read()
        b = self.p2.read()
        c = self.p3.read()
        d = self.p4.read()
        a = a < 4000
        b = b < 4000
        c = c < 4000
        d = d < 4000
        return (a, b, c, d)
    def upMiddle(self, a, b):
        print(a)
        if (a == (False, False, False, False) or a == (True, False, False, False) or a ==
(True, True, False, False) or a == (False, False, True, True) or a == (False, False,
False, True)): #on line end
            return 0
        elif (a == (True, True, True, True)):
            return b
        elif (a == (True, False, True, True) or a == (False, True, True, True) or a == (False,
False, True, True) ):
            return 2
        elif (a == (True, True,False, True) or a == (True, True, True, False) or a == (True,
True, False, False)):
            return 1
        elif (a == (True, False, False, True)):
            return 3
    def up(self, a, b):
        print(a)
```

```
        if (a == (True, False, False, False) or a == (True, True, False, False) or a ==
(False, False, True, True) or a == (False, False, False, True)): #on line end
            return 0 #all on
        elif (a == (True, True, True, True)):
            return b #off line
        elif (a == (True, False, True, True)):
            return 2 #right
        elif (a == (True, True, False, True)):
            return 1 #left
        elif (a == (True, False, False, True)):
            return 3 #straight
import pyb
import utime
from mo import MotorDriver
class Line:
    def __init__ (self, ir, moe1, moe2):
        self.ir = ir
        self.moe1 = moe1
        self.moe2 = moe2
    def driveForward(self):
        self.moe1.enable()
        self.moe2.enable()
        self.moe1.setduty(20)
        self.moe2.setduty(20)
        utime.sleep_ms(1400)
        self.moe1.set_duty(0)
        self.moe2.set_duty(0)
        self.moe1.disable()
        self.moe2.disable()
    def rotateStart(self, direction):
        self.moe1.enable()
        self.moe2.enable()
        self.moe1.setduty(direction*20)
        self.moe2.setduty(direction*-20)
        utime.sleep_ms(1400)
        self.moe1.set_duty(0)
        self.moe2.set_duty(0)
        self.moe1.disable()
        self.moe2.disable()
```

```
def findLine(self, numLines, direction):
    self.moe1.enable()
    self.moe2.enable()
    self.moe1.setduty(direction* 20)
    self.moe2.setduty(direction* 20)
    lineCount = 0
    while(1):
        b = self.ir.update()
        if (b[1] == False):
            lineCount += 1
            if(lineCount == numLines):
                self.moe1.set_duty(0)
                self.moe2.set_duty(0)
                print("stop")
                self.moe1.disable()
                self.moe2.disable()
                break
            utime.sleep_ms(400)
def runLine(self, numlines, typeLine):
    self.moe1.enable()
    self.moe2.enable()
    a = 3
    lineCount = 0
    while(1):
        utime.sleep_ms(10)
        b = self.ir.update()
        if (typeLine):
            a = self.ir.up(b, a)
        else:
            a = self.ir.upMiddle(b, a)
        if (a == 0 and b == numline):
            self.moe1.set_duty(0)
            self.moe2.set_duty(0)
            print("stop")
            self.moe1.disable()
            self.moe2.disable()
            break
        elif (a == 1):
            print("left")
```

```
    self.moe1.set_duty(20)
    self.moe2.set_duty(15)
elif (a == 2):
    print("right")
    self.moe1.set_duty(15)
    self.moe2.set_duty(20)
else:
    if (a == 0):
        lineCount += 1
    print("straight")
    self.moe1.set_duty(20)
    self.moe2.set_duty(20)
```