Skip-the-Servers

Restaurant Food Delivery Robot

Sebastian Felipe Rivera Alfonso

Submitted in fulfilment of the requirements of the PEO Engineering Report

Date: December 15th, 2021

Statement of Originality: I hereby certify that all the work described within this report is the original work of the author. Any published (or unpublished) ideas/or techniques from the work others are fully acknowledged in accordance with the standard referencing practises, and I have obtained the necessary permission from the company to submit this report.

Acknowledgments

I would like to extend my sincere thanks to Jack gillies for his undivided attention and guidance during the electrical powertrain aspects of this project.

I also wish to thank Dr. Timber Yuen for his critical questions that shaped my critical thinking skills to seek solutions during the complications of this project.

Abstract

In the restaurant industry since the pandemic there has been a demand for a more safe and sanitary experience. Efforts even before the pandemic introduced autonomous robots in many industries where the tasks range from simple to very complex. These new methods of delivering food from a kitchen to table are in practise today in many restaurants around the world. Bear Robotics was featured in a Forbes article in January 2020 titled "*Mass Production of Self driving robots Restaurant Robot is coming*", where it outlined an investment valued at $32 million to initiate the mass production of self driving robots. Fortunately, there is a supply for the demand for servers and bartenders in Canada's food industry is evident, according *to Restaurants Canada's* most recent survey conducted in July 2021, it was concluded that "80% of respondents said that they were finding it difficult to hire back-of-house staff and 67% were having trouble filling front-of-house positions". Not only in Canada but around the world restaurant labour shortages are concerning and finding staff for a restaurant is becoming a new challenge, leaving restaurants to decide if it is worth investing in autonomous robots. The capabilities and characteristics of robots already on the market like the *Keenon Robot T6* and *Servi* both serve as a benchmark to meet and influenced the goals of the capstone team. The goal of SkiptheServers robot is to provide a semi autonomous food serving robot with the capability to maintain the meal in an enclosed sanitary space and present guests with a warm meal when it arrives at a table via a vertical revolving tray mechanism. These objectives were separated into three parts, one was the ability for the robot to maneuver though crowded and tight spaces using sensors and appropriate powertrain components, second the robots semi autonomous abilities had to accompanied with a navigation system to reach different locations within the restaurant; lastly the tray mechanism functionality should present dining guest with their plates. In this report the semi autonomous capabilities and powertrain will be explored in detail. The main goals of this specific section of the project are the following.

- Reach a power efficient and safe velocity
- Obstacle avoidance, maneuver through a busy restaurant setting
- Low-cost components, household materials as well as low-cost electronics
- Implement PID control for motors responsiveness

These objectives are all to be met using mechanical engineering design and mechatronics engineering principles. The programming of the robot was aided by online resources, the design was all custom. Below is a plan diagram for this part of the project.

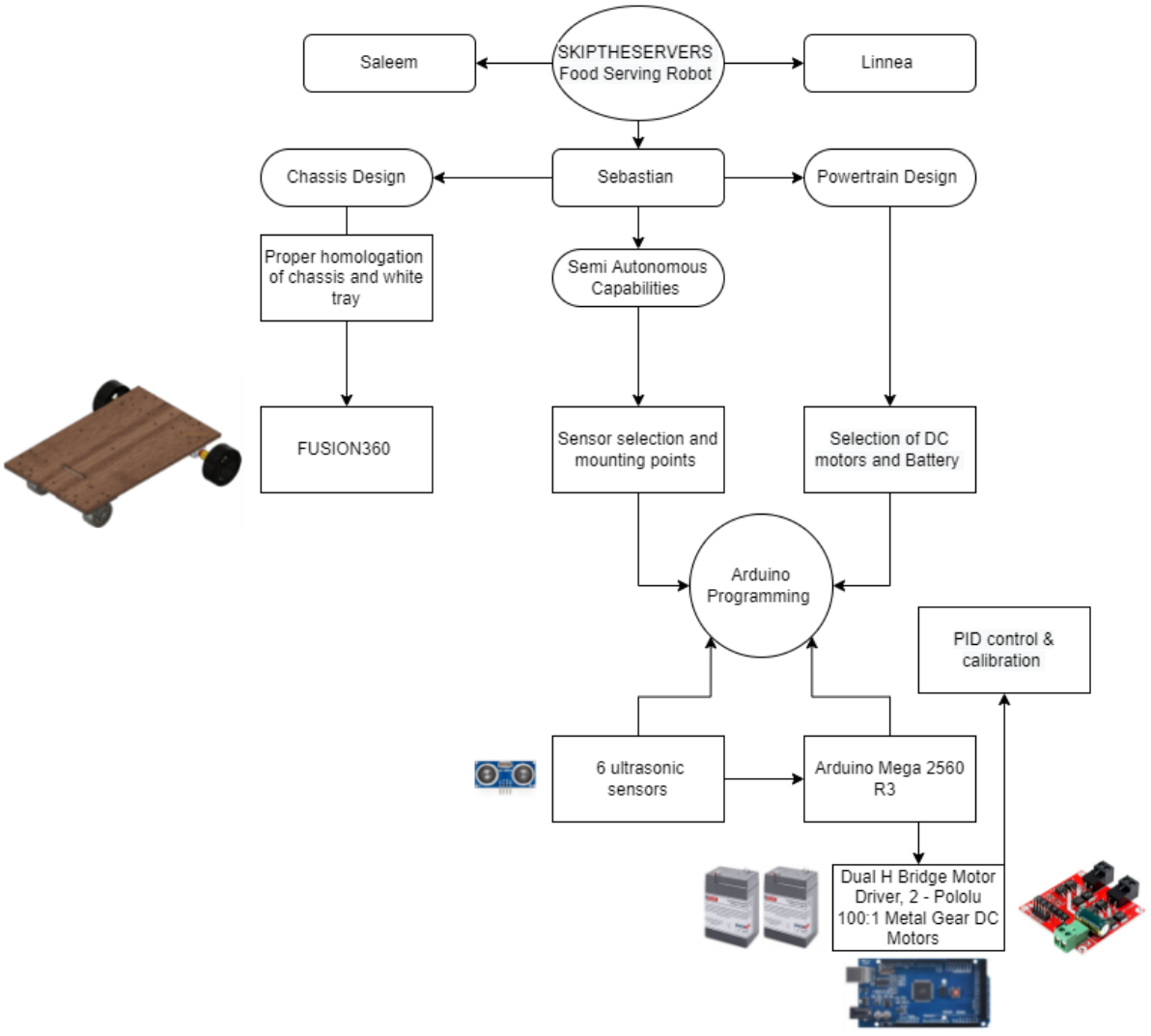
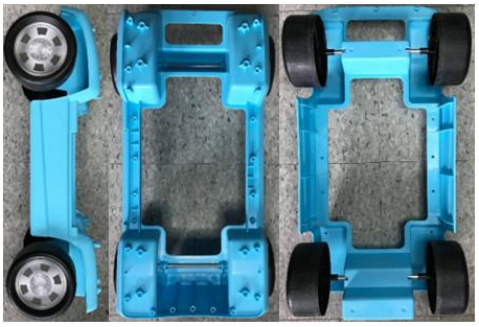


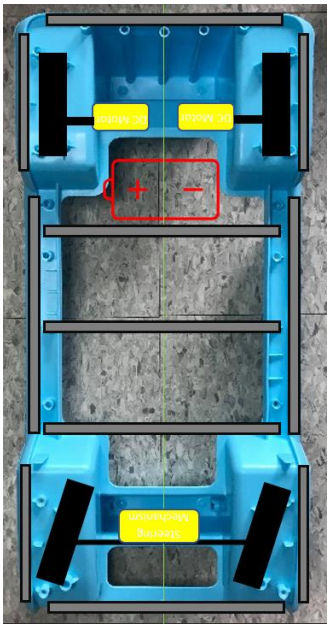
Table of Contents

1.1 Introduction

SkiptheServers food serving robot is to provide a safer and sanitary experience for a dinning at a restaurant. Using low-cost components and materials, a powertrain and chassis will be engineered, designed, manufactured, programmed, and assembled, all using a multitude of engineering skills learned from the Bachelor of Technology Automotive and Vehicle Engineering Technology Program. The objective is to construct a semi autonomous robot that avoids obstacles and provides a powertrain solution for the tray and navigation mechanism that is to be homologated.

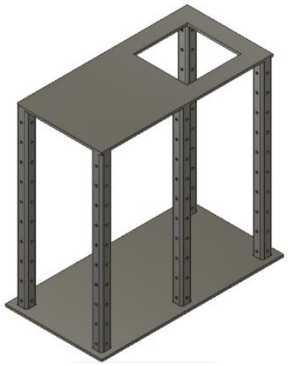
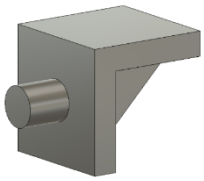
1.2 General approach to Problem





Purchasing the main frame and chassis of the robot was the first step to dictate the direction of all the three sections of the project. Designing the main frame and chassis from the ground up would have been more costly than sourcing an children's toy to repurpose. Efforts to steer and give direction to the project started very early in 4TR1, below are images of the first prototype of the chassis and placement of all the electronic components necessary for a powertrain. The placement of electronic components can be seen in the image above where the battery (in red) and motors (in yellow) can be identified. The grey bars represented metal rods that would act as a reinforcement to provide more rigidity to avoid chassis flex. This was supposed to be the chassis while the frame of the robot could have been homologated via the 24 screw holes, these holes provided already cut mounting holes for the main frame.

Utilizing SOLIDWORKS the frame below represents the first vision of the frame that was planned to be assembled from plywood and four c-channel steel structs, the bottom middle image on the left provides flexibility to the placement of trays for the revolving tray mechanism and other components. This small piece would have been 3D printed.


As a team the decision was made to purchase a low-cost Ikea bathroom shelving storage unit. The VESKEN shelf units provided a flexible staring point to for the material and size of the chassis.

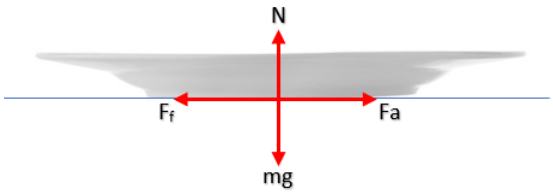
1.4 Approach and Methods

1.4.1 Powertrain Selection

The original motor that was supposed to be sourced was the 30:1 Metal Gearmotor, subsequently this motor is from a specific supplier and ordering on from Pololu's online store would have increased the cost and extended the estimated time of arrival. Therefore, the only other option was to purchase a 100:1 Metal Gearmotor locally form Robotix, this was the decision taken to minimize wait time for prototyping. This motor provides more torque at 34 kg-cm over the 14 kg-cm that the 30:1 motor would have produced, this unintentional decision will benefit the robot in the long term since it can carry heavier meals. The higher the stall torque the higher the limit of the motor's ability to pull or push heavy loads.

Rated Voltage	Stall Current	No Load Current	Gear Ratio	No-Load Speed (RPM)	Extrapolated Stall Torque (Kg-cm)	Max Power (W)	Model
12 V	5.5 A	0.2 A	30:1	330	14	12	4752
			100:1	100	34	8*	4755


Assumptions



- Mass Robot: M = 25 kg
- Gravity: g = 9.8m/s^2
- Mass of Plate: M_{Plate} = *does not matter
- Coefficient of Static Friction: μ_s = 0.2 (Ceramic and Plastic), this means that the acceleration should not be higher than 0.2g(1.962m/s^2)
- Acceleration: a = 0.15 m/s
- Wheel diameter: D_w = 7cm, r_w = 3.5cm

$$F = ma \rightarrow F_{wheel} = (25kg)\left(0.15\frac{m}{s^2}\right) = 3.75\ N$$

Therefore, each wheel must exert 3.75 N of tractive force.

$$T_{wheel} = (F_{wheel})(r_{wheel}) = (3.75\ N)(0.035m) = 0.13383\ Nm\ per\ wheel \rightarrow two\ wheel\ drive$$

Using the conversion of 1 N-m to kg-cm = 10.19716 kg-cm,

Then 0.13383 N-m = 1.33837725 kg-cm = 13.3837725 kg-mm

As seen below we want to remain on the efficiency curve, when the torque is equal to 13.38 kg-mm the corresponding RPM is about 98. At this RPM the velocity will be

$$V = \frac{RPM}{60} * 2\pi * r \rightarrow = \frac{98rpm}{60} * 2\pi * 0.035m = 0.3591\frac{m}{s}$$

This velocity would yield an efficiency of around 39% as seen below.

Operating the motors according to their characteristics to reach max efficiency while maintaining an adequate safety factor.

$$S.F = \frac{capacity}{Actual\ required} = \frac{(42kg-mm)}{(13.38kg-mm)} = 3.13$$

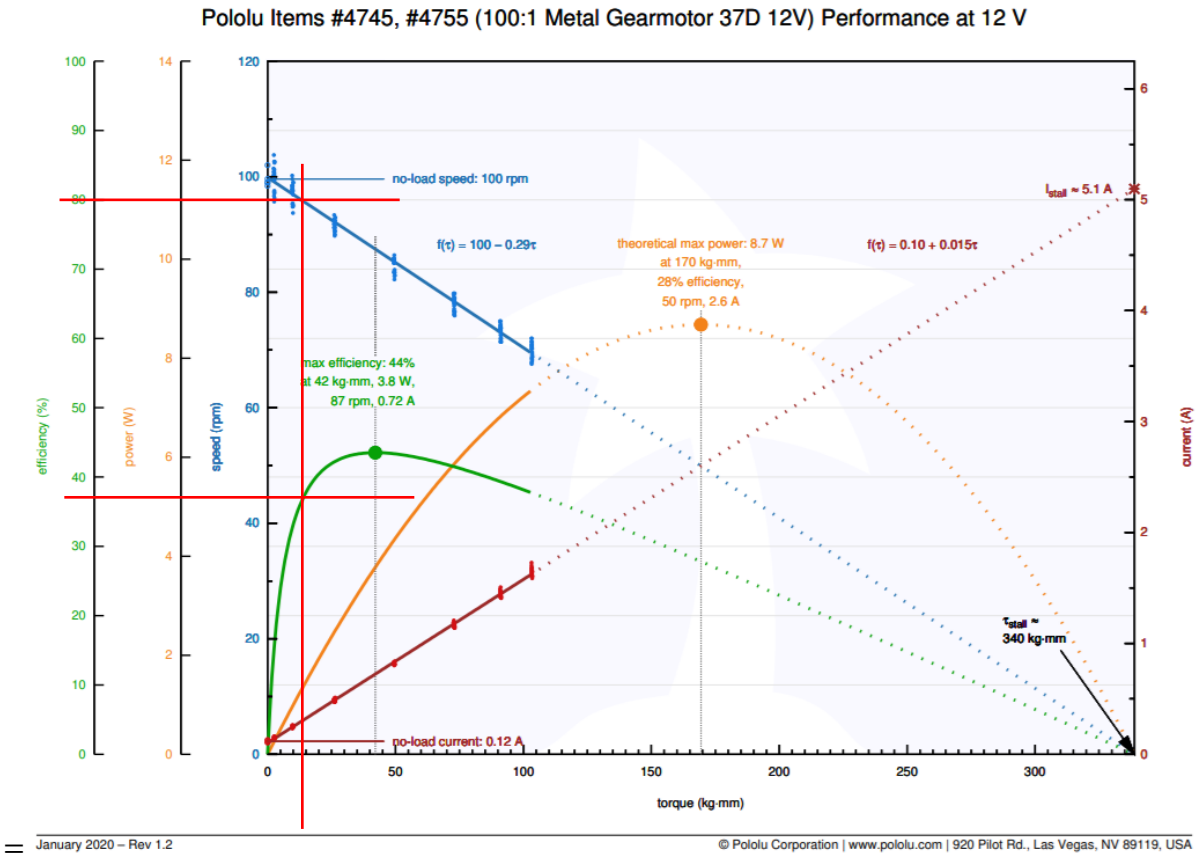
At the motors' maximum efficiency, the rpm value is 87 while at 42 kg-mm this is beneficial since the torque should be higher to account for losses in the gear box and circuit interferences.

Therefore, the desired velocity will be.

$$V = \frac{RPM}{60} * 2\pi * r \rightarrow = \frac{87\ rpm}{60} * 2\pi * 0.035m = 0.3188\frac{m}{s}$$

Since the no load rpm is equivalent to the maximum rpm the motor can reach, the maximum velocity can be found,

$$V_{max} = \frac{RPM}{60} * 2\pi * r \rightarrow = \frac{100\ rpm}{60} * 2\pi * 0.035m = 0.366519142\frac{m}{s}$$



=

For the battery,

The battery was calculated through the amount of stall current the robot would use; the assumption is that it uses 50% of the stall current. Stall torque is the ampere the motor draws when the armature is prevented from turning. An hour is more than enough for this project's goals and scope, but an 8-hour day is more realistic. Implementing this 66 Amp-hour battery would increase the weight and decrease precious space inside the robot chassis for other critical components. The price of such battery is also an issue; therefore a 1-hour requirement is more than enough for testing and demonstration purposes. The battery below is 8 Ah which even extends the battery life longer. This is a good example of a cost and space constraint decision to decrease the chances of complications in the future when technical difficulties arise during the building stage.

$$Power_{out} = (T_{wheel})(W_{wheel})$$

$$= (0.13383\ N)(87\ rpm)\left(\frac{2\pi}{60}\right) = 1.2192741\ Watts$$

Therefore, we would ideally use 50% stall current

For an ideal case: $(2\ motors)(5.5\ A)(0.5 - stall\ current)(12\ hours) = 66\ amp - hour$

For Feasibility: $(2\ motors)(5.5\ A)(0.5 - stall\ current)(1\ hour) = 5.5\ amp - hour$

Battery Specifications			
Voltage	Capacity	Size	Weight
12 V	8 Ah	70x47x101 mm	1.596 kg

This battery being 8 Ah then, $\frac{5.5Ah\ (for\ feasibility)}{8Ah(\ actutal\ battery)} = 1.454545455$

Actual: $(2\ motors)(5.5\ A)(0.5 - stall\ current)(1.454545455\ hour) = 8\ amp - hour$

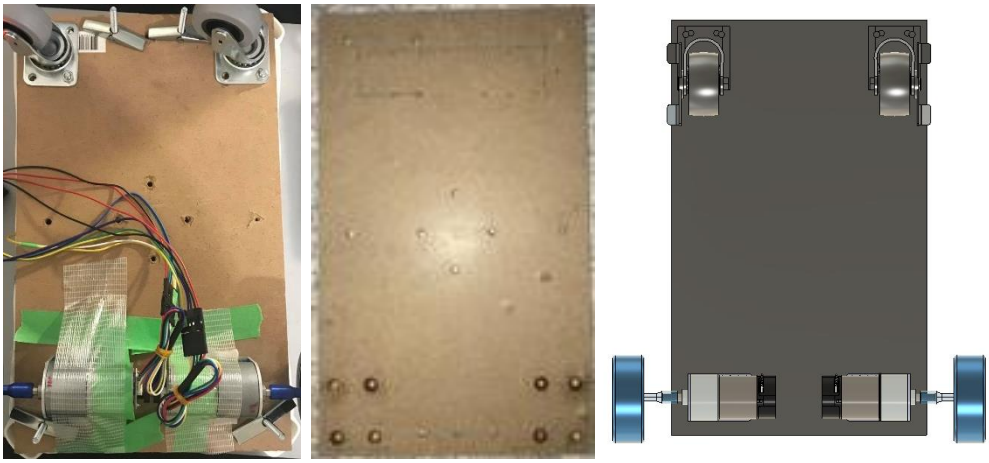
Therefore, the battery can last 1 hour and 45 minutes on a single charge.

1.4.2 Design

1.4.2.1 Plywood Chassis

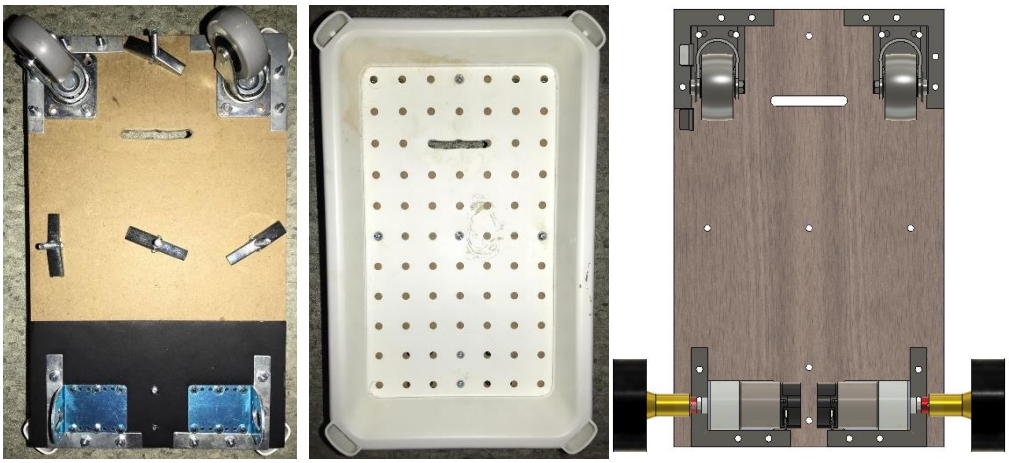
The unit shelf trays' design allowed for endless possibilities to mount the motors and sensors, using a 8x12.5 in plywood with a thickness of 0.25. Through FUSION360 the placement of the motors was visualized before beginning manufacturing, certain tolerances and height disparities between the front and rear were considered as a consequences of source caster wheels for the front. At first the turning mechanism was going to consist of a servo motor with an axle turning mechanism like RC cars, but caster wheels were decided to be used instead since most food serving robot on the market function with two wheels to steer and provide torque to move. The design should allow the robot to turn in a full circle minimal turning radius which is critical to a robot in small tight spaces.

1.4.2.1.1 Revision 1



This first prototype design was manufactured using a hand saw, drill, and rotary tool to make the countersunk holes for a flush finish on the top surface to connect with the white tray. The accuracy of the placement motor and caster wheel brackets was noted through the measuring tool in FUSION360, the holes in the prototype are aligned to the holes on the tray this allowed for a simpler assembly. The distance the wheels were sticking out on the CAD model was not desirable and was adjusted with respect to the real physical prototype.

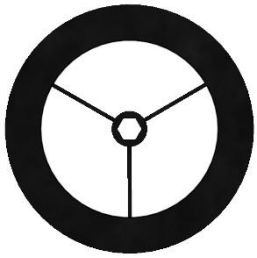
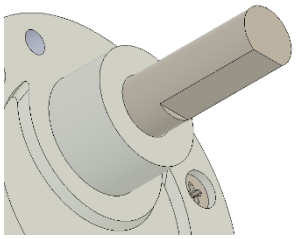
1.4.2.1.2 Revision 2



During the 1st revision's construction and assembly many lessons learned, as a result different techniques were now done more efficiently and with better accuracy. The 2nd revision manufactured using the same techniques as the 1st revision, manufacturing equipment included a hand saw, a drill, rotary tool, and a c-clamp. The datum to mount the white tray on the plywood chassis was chosen to be the center hole of the tray, the rest of the holes to hold the two structural components together were

derived from this center hole. Using the white tray's holes as a datum/grid for the placement of cable pass throughs and mounting holes allowed for a high accuracy difficult to replicate using a ruler and marker to pinpoint hole locations to drill. The clearance between the two rotating magnetic discs of each motors' encoder was the constraint that dictated the use and placement of the L-brackets. To ensure these brackets were equally spaced from either edge, another bracket was placed flush with the edge to ensure an inward offset of 0.5 in. The use of already machined distances and shapes guaranteed accurate tolerances and equal of the placement of the wheel casters and motors, with keeping in mind that this geometry would affect the mobility of the robot.

1.4.2.2 Motor/Wheel Connecting Shaft

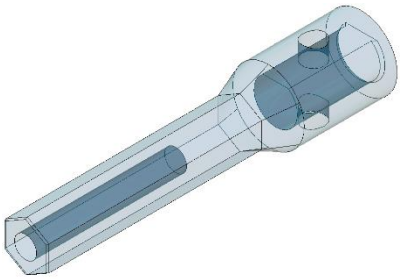
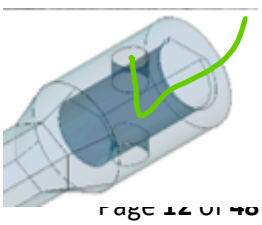





The motor/wheel connecting shaft had to be fully custom, this was achieved using a 3-D printer and FUSION360 to design the component. From the images below a 6 mm hex key was able to fit with ease with the wheel and the motor's shaft was also a 6 mm in diameter shaft with a flat surface to allow the transfer of motion. Revision 1 is the blue shaft that sheared during the disassembly process, the visual inspection of this shaft while under load did not seem safe reliable for long term operation. The piece sheared vertically since the way the filament was fused together

1.4.2.2.1 Revision 1





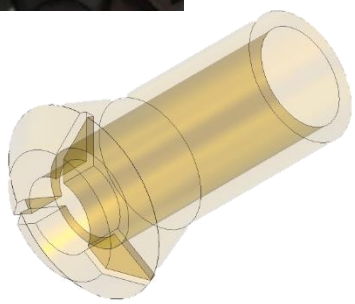
This 1st prototype was design with material use in mind and the using the previous shaft system the old RC car used that the wheels were sourced from. Material had to be removed using a rotary tool for the diameter to increase to allow the motor shaft to slide in and out easily for re-assembly. During a disassembly after testing and trouble shooting the semi autonomous capabilities with other group members this shaft sheared from the force being applied to pull it out of the motor shaft. This was likely due to the small shaft diameter since there is a M3 x 12mm bolt separating the continuity of the filament fusion. A small rubber string (seen in green above) was also used to ensure a secure fit, one end was fed through one of the 3 mm vertical holes on the motor end of the shaft and then fed out of the entrance hole of the motor shaft. This provided great friction and made sure the shaft was not going to separate during operation but complicated the disassembly process which ultimately led to the shearing of revision 1.

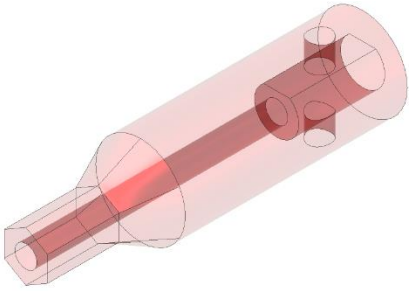
1.4.2.2.2 Revision 2

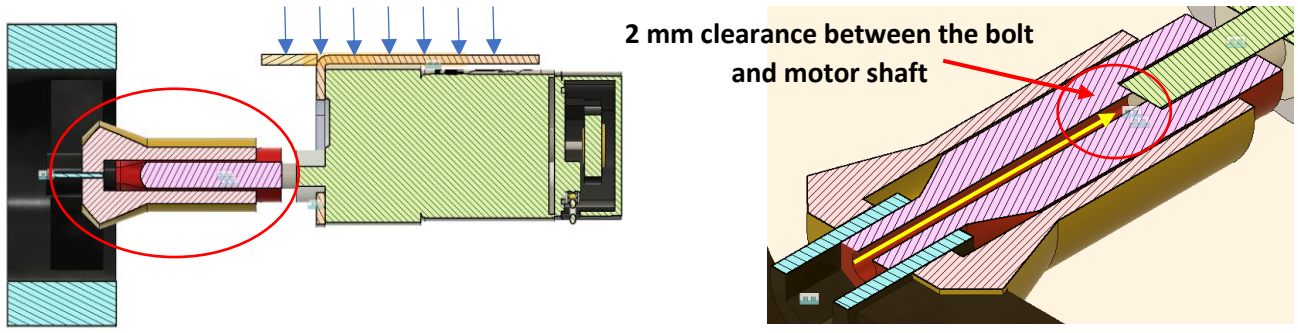
Revision 2 is a modified shaft with a larger diameter to sustain the repeated and change in torsion force direction and downward loads. This increase in cross-sectional area allows for more filament to be fused together in a larger contact area, this decreases the chances of the inner shaft shearing like the first revision did. This new design includes a sleeve to provide more structural rigidity since it increases the contact points with the wheel and maintains the shaft straight along the x direction; under operating conditions the sleeve will take most of the vertical load under until the small tolerance of 0.25 mm becomes 0 mm.





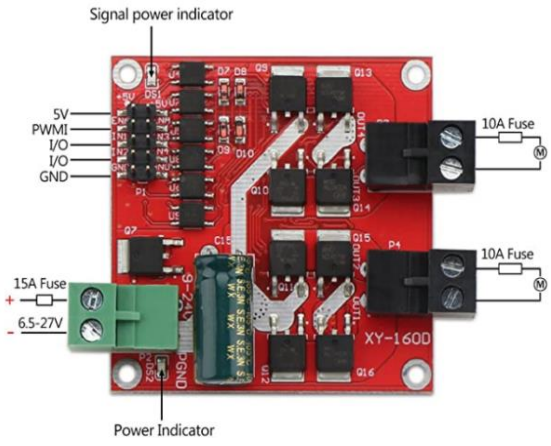






The image above depicts where and in what direction the new improve shaft design will experience the downward load. The outer connecting shaft is connected to the wheel via three contact point slots that are equally spaced, this provides reinforcement to sustain vertical loads. Then the inner connecting shaft is held in place via a M3 bolt that provides another metal skeleton shaft in the center. These combined two shafts with a bolt long enough going through the center to meet the motor shaft with a small clearance of about 1-2 mm ensures the torsion forces and weight of the robot do not shear the connection between the motor and wheels. Other notable tolerances that were revised was the diameter of the mating hole with the motor shaft, it was adjusted to be 0.05 mm larger since on revision 1; material had to be removed using a rotary tool to increase the diameter.

2. Experimental Apparatus

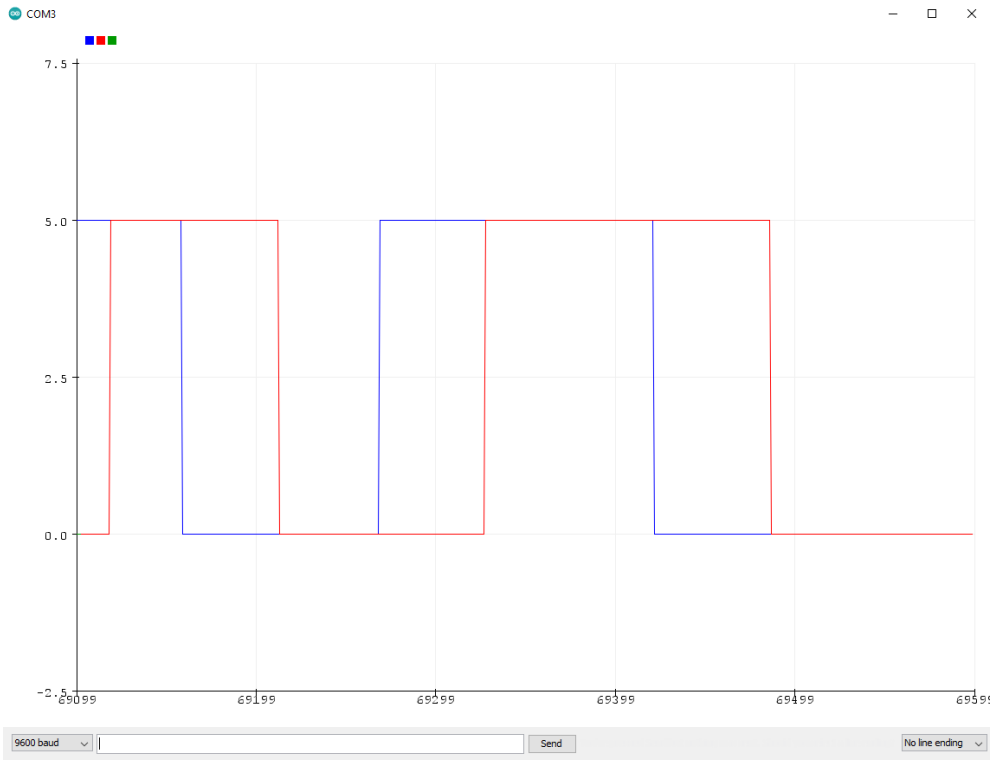


From section *1.4.1 Powertrain Selection* of this report the 100:1 Pololu Metal Gearmotors 12V and appropriate battery were sourced. Since these motors can draw 5.5 A continuously, a more capable motor driver was needed over the basic L298N motor driver included in the Arduino kit which can only provide 1.2 A. The DROK L298 Dual H Bridge Motor Driver has the capability to operate with DC 6.5V-27V and 7A, with a total output of 160 W this motor driver is more than capable. An equivalent Pololu motor driver was not cost effective, therefore the DROK motors driver was purchased from Amazon was delivered within days; the delivery time for the Pololu motor drivers' estimated delivery time was weeks. The appropriate fuses were found at Canadian Tire, which were not necessary but were implemented to avoid damaging motor driver and Pololu motors.

2.1.1 Encoder Calibration

The robot must be able to reach a speed of less than 0.5 m/s , to calibrate the proper PWM value that must be send to each motor, the encoder must be read to know how fast the wheel is rotating. The 100:1 Pololu features a rotary magnetic encoder, the resolution of this encoder is 64 counts per revolution. Using a *attachinterrupt* function, where the ENCA referring to the pin on the Arduino, the *readEncoder* is the function, and finally the *RISING* means it will interrupt when the signal is rising. The *readEncoder* function the is triggered every time ENCA's is RISING, then inside the function of the other output is read to determine weather ENCB is RISING or FALLING. When the encoder is rotated clockwise the ENCA (blue) is RISING first and ENCB is FALLING and counter clockwise is the same but reversed, this simple function contributes to calculating the speed of the wheel on the robot. According to the pinout schematic of the Arduino Mega 2560 the digital pins that are usable for interrupts are 2,3,18,19,20,21. These pins are classified as "external interrupts" which can be programmed to trigger an interrupt how ever deemed necessary.

```
attachInterrupt(digitalPinToInterrupt(ENCA), readEncoder, RISING);
```



This can be found in the code below in the appendix.

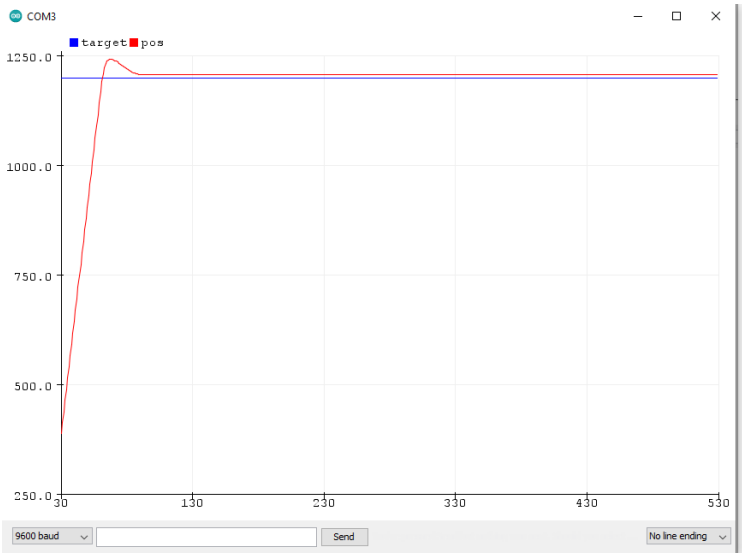
$$\frac{1\ motor\ rotation}{16\ counts}\ x\ \frac{1\ output\ shaft\ rotation}{102.083\ motor\ rotations} = \frac{1\ motor\ shaft\ rotation}{1632\ counts}$$

```
//convert count/s to RPM
float v1 = velocity1 / 1632.00 * 60.00;
float v2r = velocity1r / 1632.00 * 60.00;
//convert to m/s
float v1ms = (v1 / 60) * (2 * 3.14) * (0.035);
float v1msr = (v2r / 60) * (2 * 3.14) * (0.035);
```

The full code above can be found in the appendix, Velocity1 is in counts/second and then converted to RPM by the v1 equation, and finally to m/s multiplying by 2 Pi and the diameter of the wheel. After the counts per second the second is converted to rpm converted to rpm the conversion to m/s takes place.

2.1.2 PID Calibration

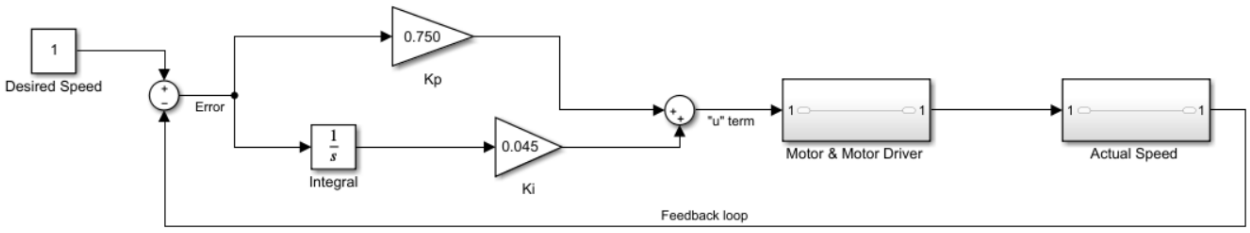
Since the encoder provides a 64 counts per revolution resolution setting a target location of resolution position of 1200 counts means that the wheel will rotate 18.75 times until the motor shaft stops abruptly.



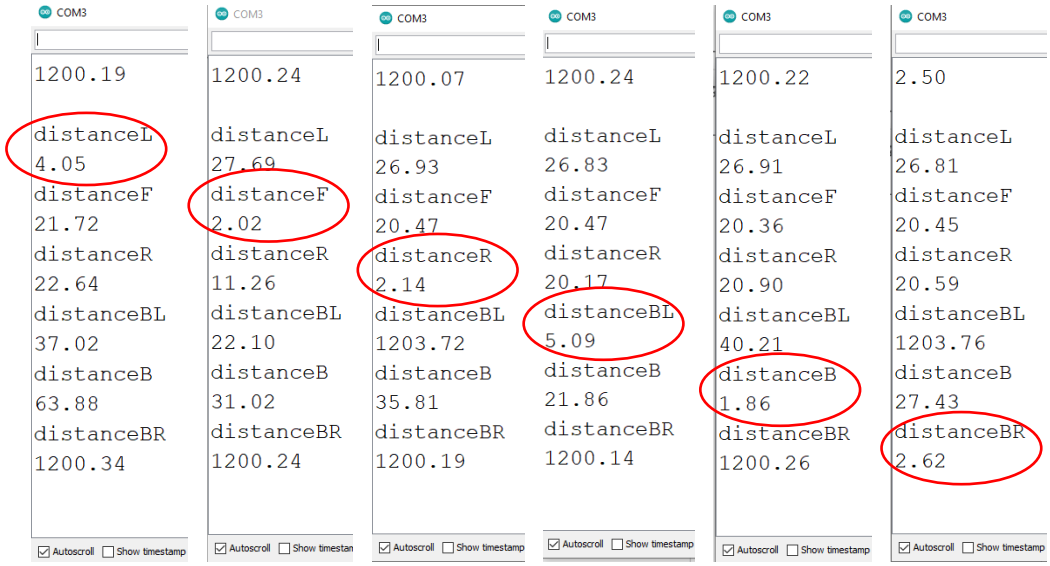

```
27
28 void loop() {
29
30   // set target position
31   int target = 1200;
32   //int target = 500*sin(prevT/1e6);
33
34   // PID constants
35   float kp = 1;
36   float kd = 0.025;
37   float ki = 0.0;
38
39   // time difference
40   long currT = micros();
41   float deltaT = ((float) (currT - p
42   prevT = currT;
43
44   // Read the position in an atomic
```


```
27
28 void loop() {
29
30   // set target position
31   int target = 1200;
32   //int target = 500*sin(prevT/1e6);
33
34   // PID constants
35   float kp = 0.750;
36   float kd = 0.045;
37   float ki = 0.0;
38
39   // time difference
40   long currT = micros();
41   float deltaT = ((float) (currT - pr
42   prevT = currT;
43
44   // Read the position in an atomic b
```

From above this calibration was achieved through trial and error, the Ki and d constant were manipulated until the steady state error was completely removed. Observing position of the motor responds to when Kp= 0.750 and Kd= 0.045 it is evident that this is the desired response for this application. If there is overshoot the robot will accelerate fast and then reach the desired speed during direction changes, the plate on the tray mechanism must also be kept level. Since the robot is transporting food, the gradual acceleration and deceleration is critical to avoid spills and damage to other mechanical mechanisms. To control the speed of the motors a proportional controller is used to reduce steady state error. Kp is the proportional term and kd is the integral term. The proportional controller works to control and monitor the speed of the motor by adjusting the voltage feed to the motor to match the desired speed. The difference between the desired speed and the actual speed is the error, this error is the multiplied by the Kp term to address the speed according to the positive or negative voltage that must be applied to control the speed. This feedback control can be seen below where the u term in the Arduino code the term “u” is seen coming into the motor is the error multiplied by voltage to be applied. The Kd value remained at zero but with more tuning and changing the values of kp and kd, this term can be tuned with a value greater than 0, for simplicity and efficiency of the programing and calibration purposes it was left at 0.



2.1.2 Ultrasonic Sensor Calibration

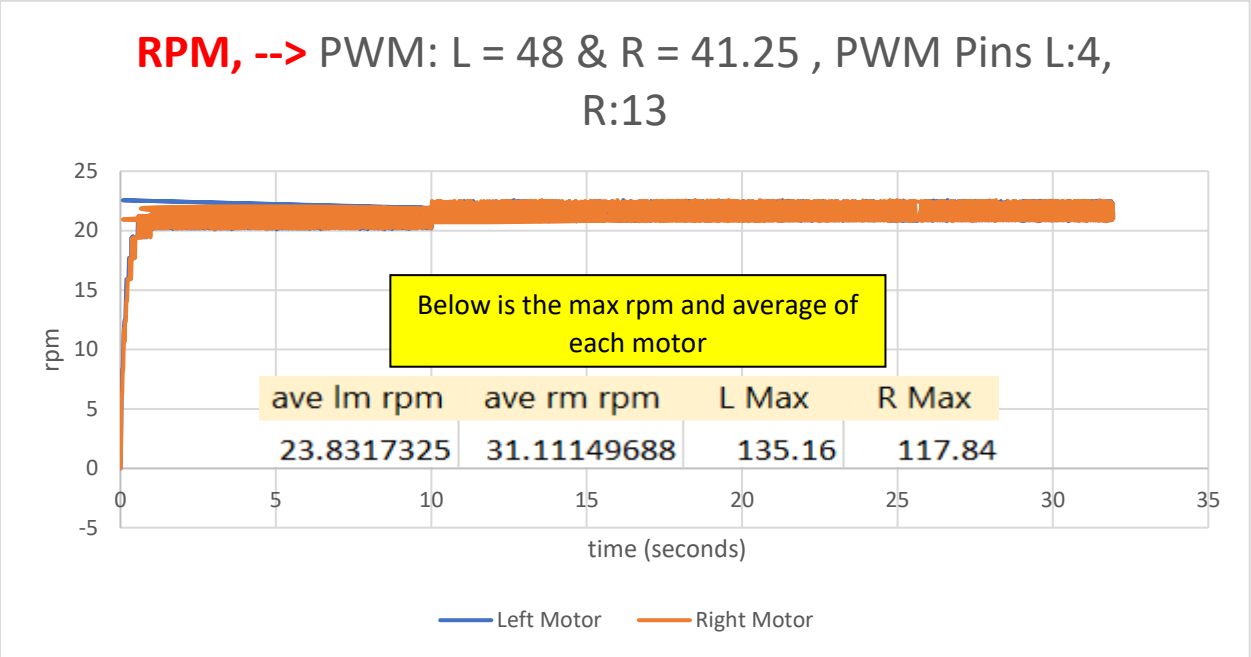


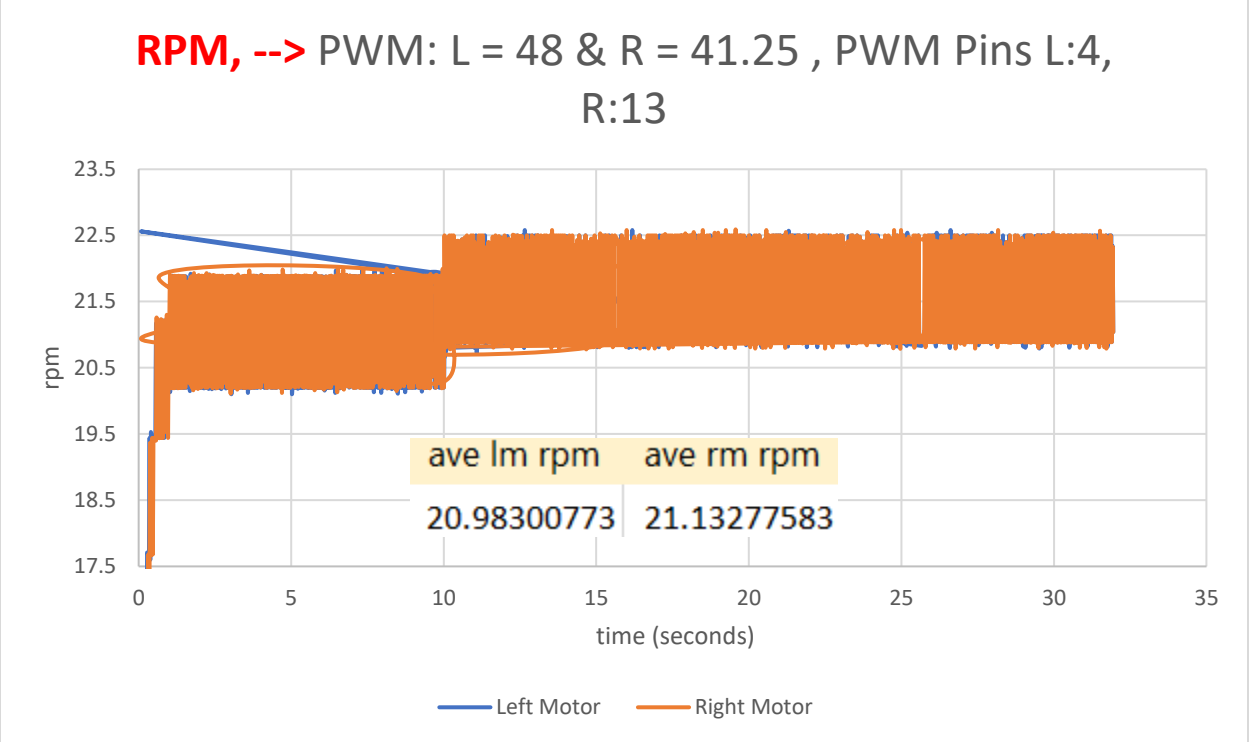
There are total of six ultra sonic sensors placed on robot, at each corner and at the front and back of the robot there is a sensor. The image above illustrates a paper no less than 5 cm way from each sensor. The placement of these sensors should avoid colliding with obstacles along its path back to the kitchen. From previous course work involving ultrasonic sensors, these were programmed to measure distance in centimeters. The *pulseIn()* sends a pulse for 10 microseconds to trigger the trig pin, then listens for a pulse in the echo pin; this pulse' duration is equal to the first pulse sent out. The time taken for the ultrasonic sound to return is then divided by 58. Since the speed of sound is 340 m/s or 29 microseconds per centimeter, this is multiplied by 2 since only the duration of one way is only required to determine how far an object is.

```
float realSensorDataF() {
  digitalWrite(trigPinF, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPinF, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPinF, LOW);
  float distanceF = pulseIn(echoPinF, HIGH) / 58.00;
  return distanceF;
}
```

2.1.2 Speed Calibration / Trouble Shooting

Below are graphs of the rpm vs time of using different parameters





Since Pins 2-13, 44-46 are designated PWM PINS on the Arduino Mega 2560, where PINS 4 and 13 are 980 Hz compared to the 480 Hz frequency of the other pins. These discrepancies can be due to losses in

the gear ratio in one of the motors, signal interference coming from the motor driver or inconsistent voltage from the battery. Using a PWM values of 48 for the left and 41.25 for the right motor allows both motors to produce similar rpms. Using the datasheet of the motor, the relationship derived from the no load speed 100 rpm at 12V, this represents the maximum speed of the motor is 100 rpm at PWM value of 255. The relation can aid in calculating the PWM value, based on the desired rpm and vise versa. Since the desired speed is 0.3188 m/s, and the wheel torque is 13.38 kg-mm at approximately 98 rpm. Then,

$$\frac{12\ V}{100\ RPM} = \frac{V_m}{RPM_{new}} \rightarrow Where, V_m = \frac{PWM}{255} * 12V$$

$$0.12 \frac{V}{RPM} = \frac{V_m}{98\ RPM_{new}} \rightarrow 11.76\ V = \ V_m$$

$$11.76\ V = \frac{PWM}{255} * 12V \rightarrow \frac{PWM}{255} = 0.98 \rightarrow PWM = 249.9$$

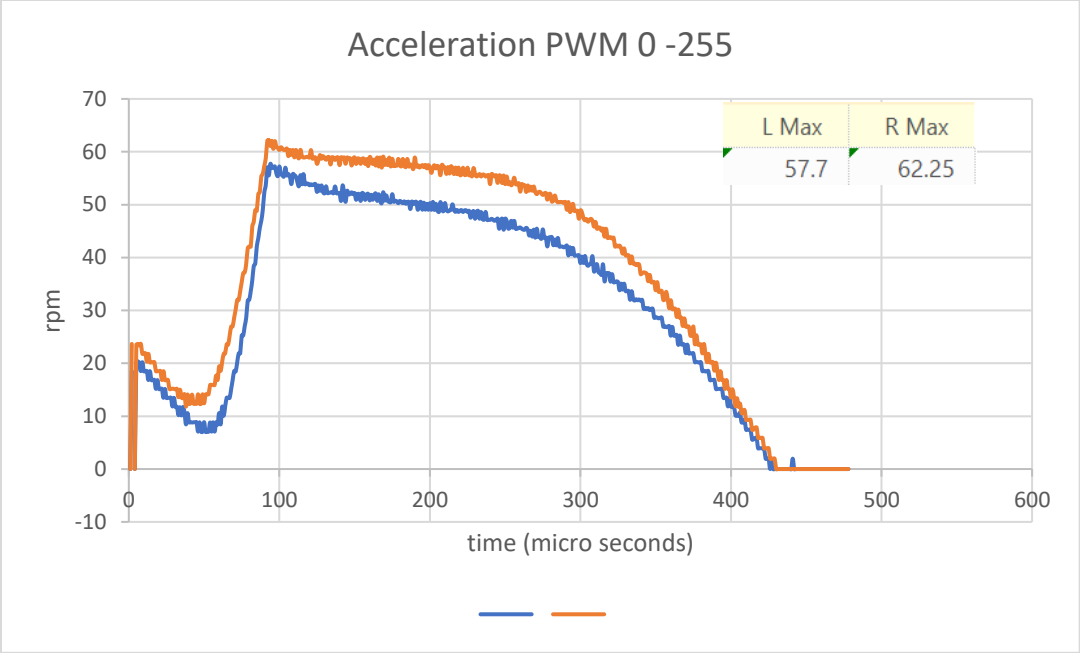
This PWM value does not reproduce the desire rpm of 98, the gear box of each motor is slightly different hence the different PWM values that achieved the same speed. Below is the same calculation but simplified for the PWM of 87 rpm,

Since the no load speed is 100 rpm,

$$\frac{100\ RPM}{87\ RPM} = 87\%\ of\ the\ rpm\ is\ desired, simply\ multiply\ tyhe\ PWM\ by\ this\ percentage;$$

$$255 * 0.87 = 221.85\ PWM\ value$$

This is a method to estimate the PWM value in the Arduino code but as is illustrated above a PWM values, below illustrates this discrepancy even further. Below both motors were accelerated from PWM values of 0-255 over 5 seconds, the orange is the right motor, and the left is the blue. It is evident both motors produce different RPMs based on the same PWM value.

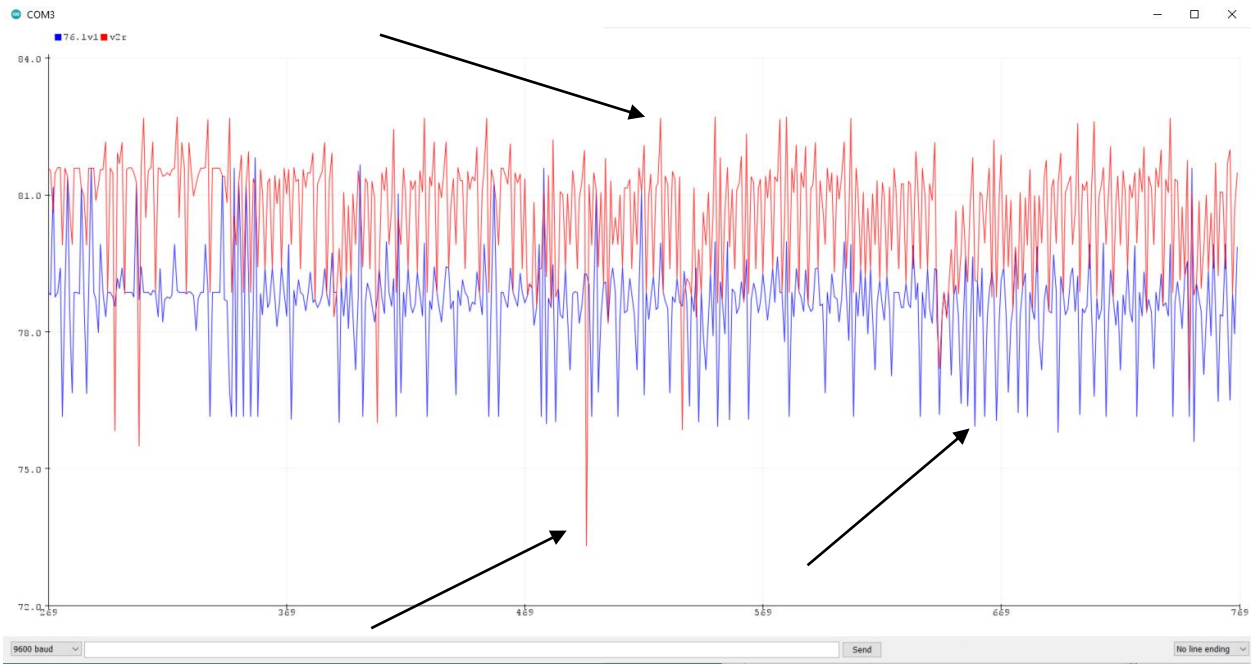


The PWM values used in the code are very distinct then the anticipated PWM values form the relationship established, a good response but when running the motors under these PWM they can be damaged during these PWM values hence the scraping sound coming from the right motor, pushing such high voltage causes unbalanced and voltage spike

```
  pinMode(IN2r, OUTPUT);

  // intrerrupt function is us
  attachInterrupt(digitalPinTo
  attachInterrupt(digitalPinTo

  //Serial.println("v1,v2r");
}

void loop() {

 myTime = millis();// * 0.001;
  // pwm value and to run the


  int pwr = 255;// left motor
int pwrr =246.5;//right motor
```

```
COM3
Send

Revolutions per minute: Left Motor: 79.23 rpm
                        Right Motor: 81.37 rpm

Velocity: Left Motor: 0.29 m/s
          Right Motor: 0.30 m/s

Revolutions per minute: Left Motor: 79.03 rpm
                        Right Motor: 81.61 rpm

Velocity: Left Motor: 0.29 m/s
          Right Motor: 0.30 m/s

Revolutions per minute: Left Motor: 78.75 rpm
                        Right Motor: 81.23 rpm

Velocity: Left Motor: 0.29 m/s
```

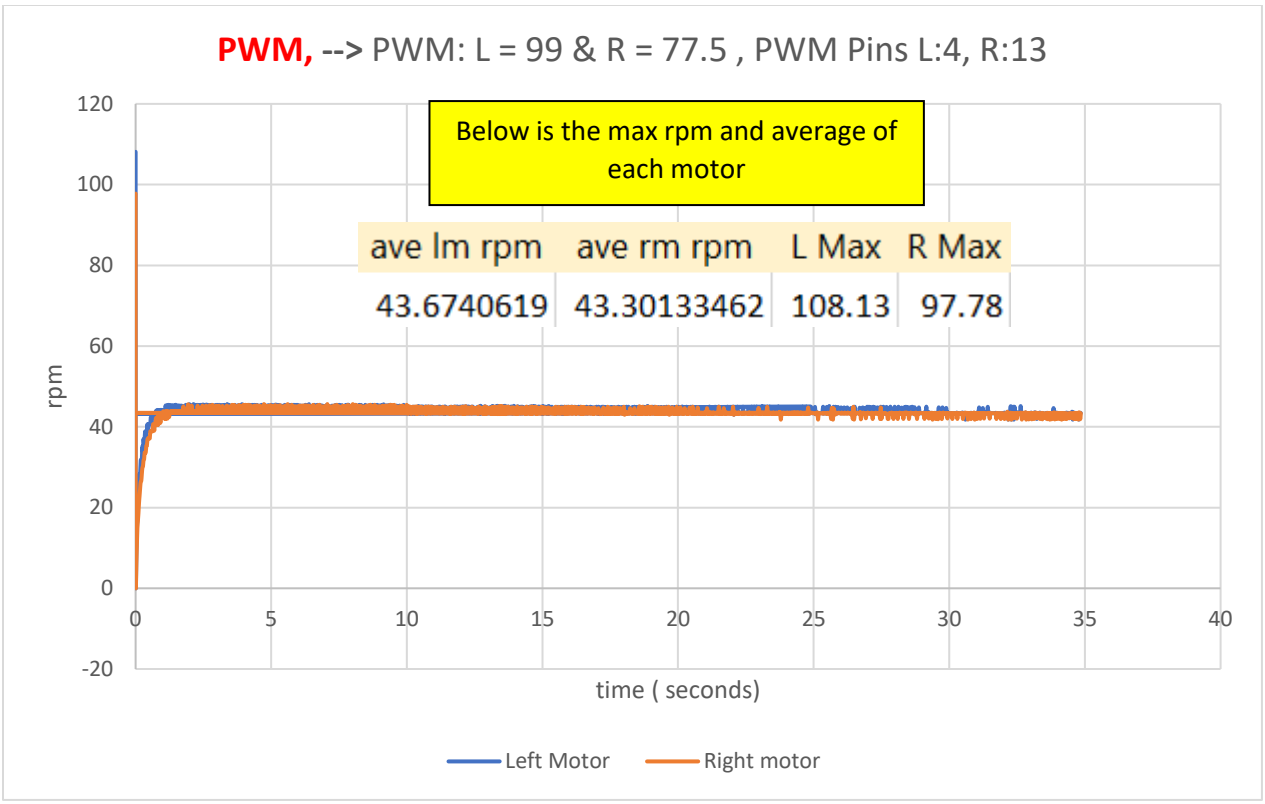
s.

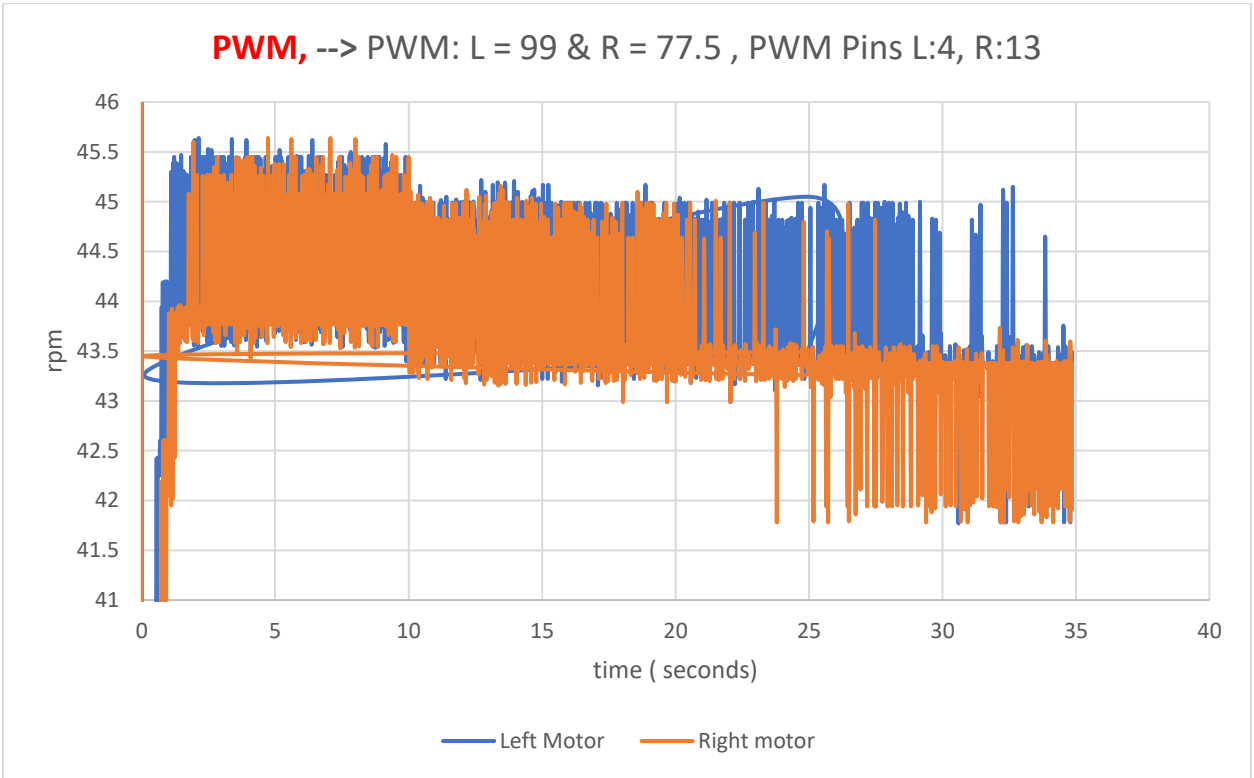


Ultimately for the longevity of the DC motors and other electrical components the speed that was used on the ground with the semi autonomous capabilities was 0.16 m/s, with the following PWM values circled in blue.

```
  pinMode(IN1r, OUTPUT);
  pinMode(IN2r, OUTPUT);

  // intrerrupt function is u
  attachInterrupt(digitalPinT
  attachInterrupt(digitalPinT

  Serial.println("v1,v2r");
}

void loop() {

 myTime = millis();// * 0.001
  // pwm value and to run the


  int pwr = 99;// left motor
int pwrr =77.5;//right motor
```

```
COM3

Revolutions per minute: Left Motor: 44.60 rpm
                        , Right Motor: 44.60 rpm ,

Velocity: Left Motor: 0.16 m/s
          Right Motor: 0.16 m/s

Revolutions per minute: Left Motor: 44.81 rpm
                        , Right Motor: 44.60 rpm ,

Velocity: Left Motor: 0.16 m/s
          Right Motor: 0.16 m/s

Revolutions per minute: Left Motor: 44.83 rpm
                        , Right Motor: 44.83 rpm ,

Velocity: Left Motor
Autoscroll  Show timestamp                No line ending
```


PWM, --> PWM: L = 99 & R = 77.5 , PWM Pins L:4, R:13
Below is the max rpm and average of each motor
ave lm rpm
ave rm rpm
L Max
R Max
43.6740619
43.30133462
108.13
97.78
rpm
time (seconds)
Left Motor
Right motor


PWM, --> PWM: L = 99 & R = 77.5 , PWM Pins L:4, R:13
rpm
time (seconds)
Left Motor
Right motor

3. Conclusion

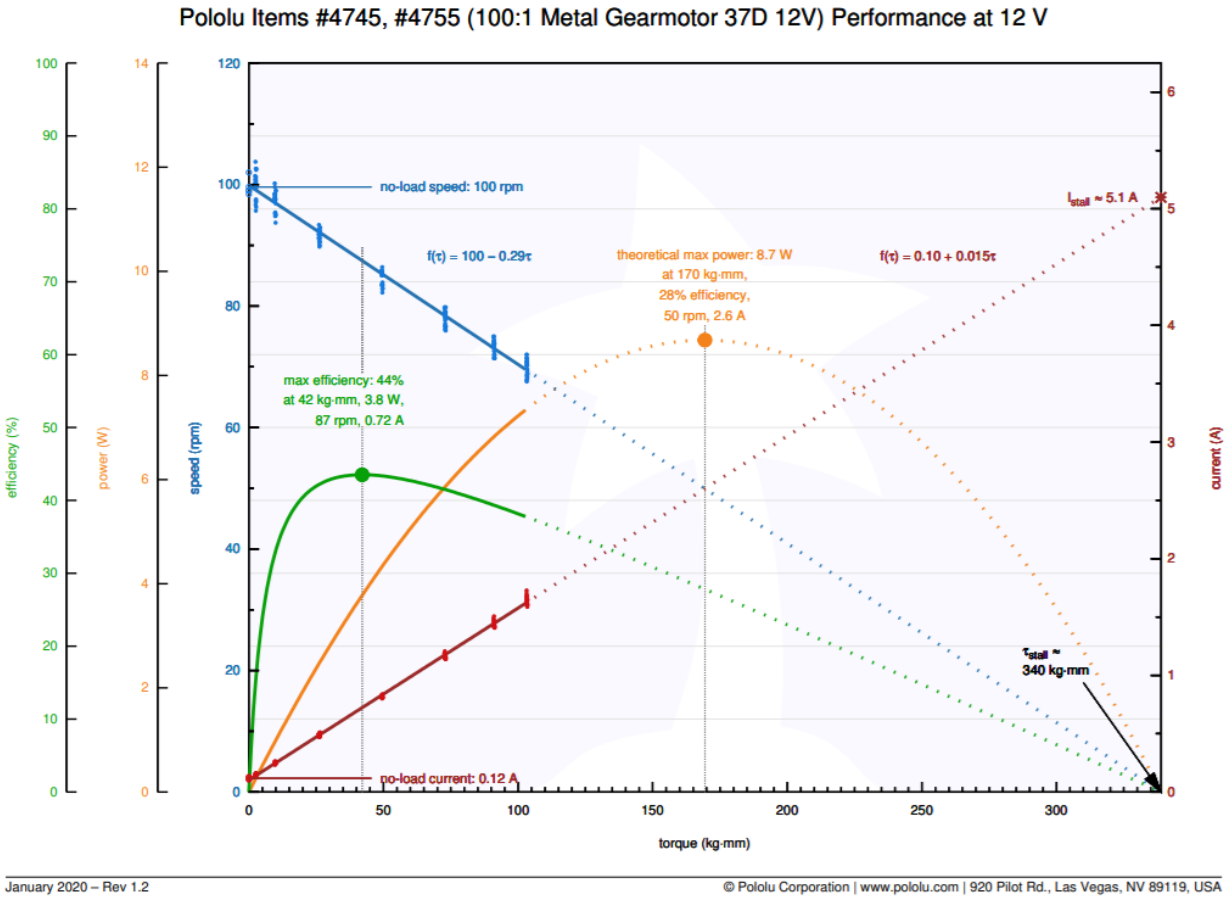
The robot reached the adequate speed of 0.16 m/s safely and without running the risk of injury or spilling food, this was done through reading an encoder and multiple days of troubleshooting. The use of PID control improved the motors' response and achieved zero overshoot. Design principles pertaining additive manufacturing and its tolerances were implemented to create a connecting shaft to provide the transfer of power. Skills and knowledge pertaining to electric motors, trouble shooting, 3D design, PID control, machine health monitoring and mechatronics a played a critical role in the development of this robot. New skills were also acquired during the processes of trouble shooting, coding a complex program in IDE and wood working. Undertaking a complex task like designing, manufacturing, assembling, and programing a semi autonomous robot while using PID control and many mechatronics principles filled the gap by forcing a review of such topics mention in this report.

4. Recommendations

The velocity of the robot could be faster, but this would introduce voltage spikes and instability and chances of the robot electronic components to be damaged, a higher operating velocity could also tip over the robot when the tray mechanism is mounted. A low past filter to remove high frequencies during reading the encoder, decreasing the number of small voltage spikes thus delivering a linear rpm graph and improve the clarity of velocity readings. The powertrain system could be redesigned with AWD capabilities, one motor per two wheels, front and rear will provide improved maneuverability and eliminate the turning radius of this design. Depicting a pin layout would have conveyed more to the reader but the code states the pins used. Implementing the use of a variable power supply would improve results on rpm readings and allow for testing the true limits of the motor. Areas for further study include implementing a low pass filter and coding the robot to map the room and to avoid objects without having to approach the distance limit, will save battery life from having to reverse and improve usability. The automatous robot industry is continuing to reach milestones with AI and using next generation sensor and camera technologies, in the future an engineering student can design, manufacture and program a fully autonomous robot when the technology becomes more widespread and affordable for broader academic use.

5. Appendices

Motor Datasheet

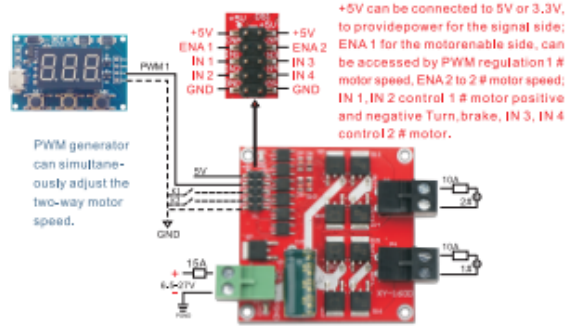


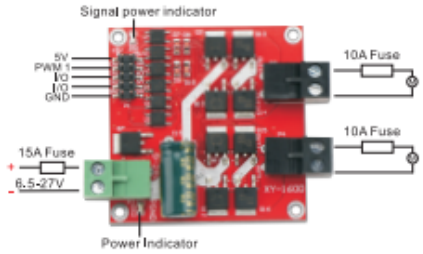
Drok http://www.Droking.com SKU:200206

7A / 160W dual DC motor drive module, high-power industrial grade, positive and negative, PWM speed, L298 logic

Product parameters:

- Power supply voltage DC 6.5V-27V, the power cannot be connected to reverse or over DC 27V, or may burn the module, it is recommended connecting in series 15A fuse in the power input.
- Dual motor interface, each rated output current 7A, peak current 50A, the motor interface cannot short-circuit, the proposed series 10A fuse.
- Control signal voltage DC 3-6.5V, respectively, is the enable signal and the positive inversion control signal.
- Enable signal terminal (ENA) input PWM adjustable speed, PWM frequency range 0-10KHZ, PWM minimum pulse width 10us.
- Working temperature -25 ℃ ~80 ℃.
- Product size: 55 * 55 * 16mm (length and width).
- Installation hole diameter: 3mm, pay attention to prevent short circuit on the back of the circuit, you can add insulation pad or copper column to raise the circuit board.





Product Highlights:

- Double H bridge, which can drive two DC motors, single 7A current, high power;
- Wide voltage input range: DC 6.5V - 27V.
- Signal optocoupler isolation input, IO port can be directly controlled, without interference;
- Under voltage protection, to prevent instantaneous high-current burned module;
- High power TVS and electrostatic discharge circuit, inhibit the transient interference pulse and static, enhanced EMC performance, made stable and reliable product, which are industrial design.

Suitable for motor parameters:

- Rated voltage 24V of the motor, the appropriate identification of rated power 115W and below or rated current of 7A or less the following long-term work of the motor.
- Rated voltage 12V of the motor, the appropriate identification of rated power 40W and below or rated current 7A rated below the motor for a long time full of work.

Precautions:

- Drive power must not be reversed, the proposed power connector in series 15A fuse, the voltage should be between DC 6.5 ~ 27V. If the voltage is over-voltage, power-on may burn the drive module.
- It is recommended that the rated output current of the power supply be at least 2 times of the rated power supply, in case the power supply voltage cannot reach the required input voltage of the motor when it is started, which will lead to off output, then causing motor pause for under voltage protection.
- Do not short-circuit the motor interface, or may burn the driver module, the proposed electrical interface in series 10A fuse.
- When switching the forward and reverse, the motor needs to brake more than 0.1S then reverse; cannot reverse when the motor is not stopped, otherwise it may damage the drive.
- When the drive module is powered down, don't rotate the motor directly or indirectly at high speed, otherwise the electromotive force generated by the motor may burn the drive module. If the application requires a high-speed rotation of the motor when the drive module is powered down, it is recommended that a relay (NO and COM) be connected in series to the motor interface of the drive, and then the relay coil and the drive are connected in common. In this way, when the power down, the relay will disconnect the drive from the motor.
- CAUTION Do not expose the drive to moisture. Do not short-circuit the components on the drive board. Do not touch the pins and pads on the board.

Control signal logic

Note: Where 0 is low, 1 is high level, × is any level, when floating high.

- 1 # motor interface control signal logic

IN 1	IN 2	ENA 1	OUT1、OUT2
0	0	×	brake
1	1	×	Floating
1	0	PWM	Forward to speed
0	1	PWM	Reverse speed
1	0	1	Full speed forward
0	1	1	Full speed reversal

- 2 # motor interface control signal logic

IN 3	IN 4	ENA 2	OUT3、OUT4
0	0	×	brake
1	1	×	Floating
1	0	PWM	Forward to speed
0	1	PWM	Reverse speed
1	0	1	Full speed forward
0	1	1	Full speed reversal

BOM

MATERIAL REQUIREMENTS		PRICE
Chassis	8x12.5 in plywood, 0.25 in thick, 4 x L-brackets	L-brackets - $2
Bolts	12x M3x12mm, 16 x M4x16mm bolts and nuts, 4x 3/16x 2-inch Toggle Bolts(one was cut)	Bolts - $8
Household miscellaneous	Popsicle sticks, electrical tape, hot glue, double sided tape, rubber bracelet string, rubber stoppers, nails, zip ties,	n/a
Arduino & motor driver mounting base	9x5.5 in wood plate, 0.5 in thick	n/a
POWERTRAIN REQUIREMENTS		
DC Motors	2x 100:1 Pololu Metal Gearmotor 37D 12 V with corresponding brackets	Motors & brackets - $110
Motor Driver	DROK L298 Dual H Bridge Motor Driver DC 6.5V-27V 7A Motor Control Board	$25
Fuses	AGC 3AG Littelfuse - 1x15 A, 2x10 A, 1x Omni Block Fuse Block	$15
Battery	2x Long Way 6V 4Ah, Sealed Lead Acid	n/a
Wheels	1x Kids RC car	$8
Shafts	4 x 3D printed shafts (2 protypte,2 final design)	$10
Battery – Arduino	Small mobile battery pack	n/a

6. Acknowledgments

I wish to thank Dr. Timber Yuen for his critical questions that shaped my critical thinking skills to seek solutions during the complications of this project.

I would also like to extend my sincere thanks to Jack Gillies for his undivided attention and guidance during the electrical powertrain aspects of this project.

7. References

Canada restaurants staff shortage

https://www.restaurantscanada.org/resources/reopening-of-restaurants-leading-to-labour-shortages/

keenon robot

https://www.keenonrobot.com/EN/index/Lists/index/catid/1.html

BearRobotics

https://www.bearrobotics.ai/products

bear robotics Forbes article

https://www.forbes.com/sites/lanabandoim/2020/01/24/mass-production-of-a-self-driving-restaurant-robot-is-coming/

motor driver in arudino kit specs

https://www.ti.com/product/L293D

Arduino Mega 2560 specs

https://store-usa.arduino.cc/products/arduino-mega-2560-rev3?selectedStore=us

Arduino Serial Plotter

https://www.youtube.com/watch?v=WnxBNxX_WDc

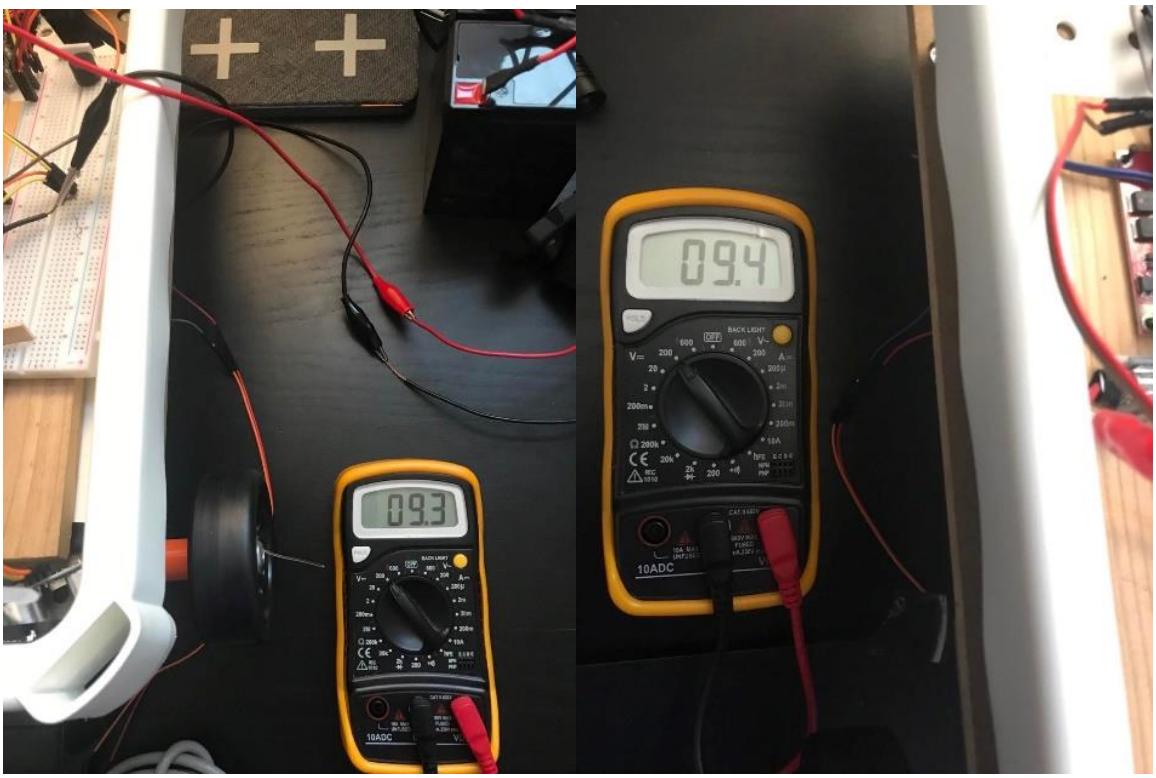
DC Motor control

https://www.youtube.com/watch?v=dTGITLnYAY0

7. Appendix

Testing

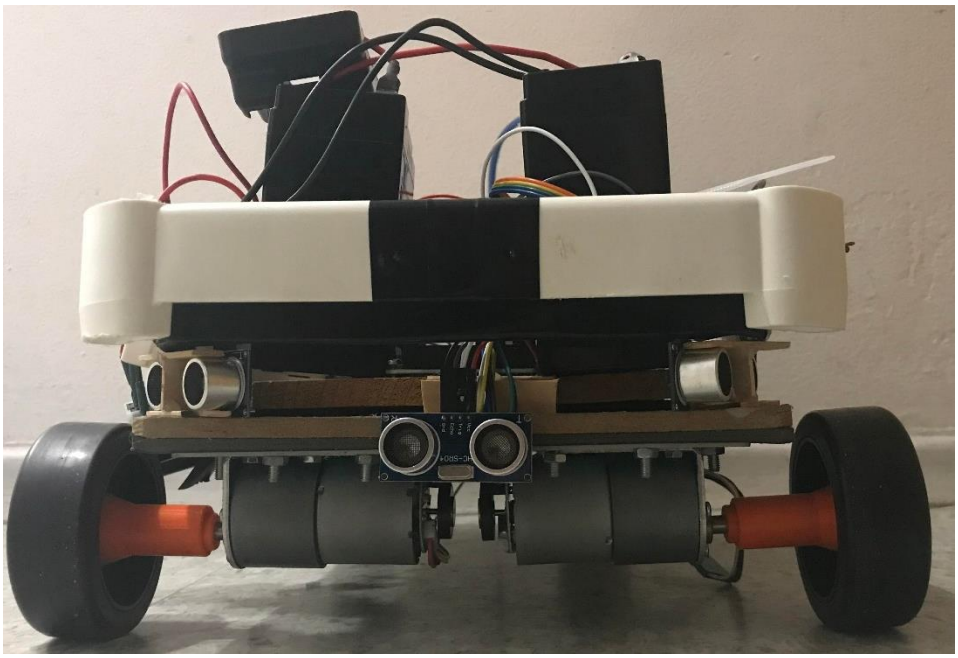
These are picture taken during troubleshooting, below is a picture involved testing the voltage being provided for the 87 RPM speed setting, 9.4 volts were provided to the left wheel and 9.3 to the right wheel.



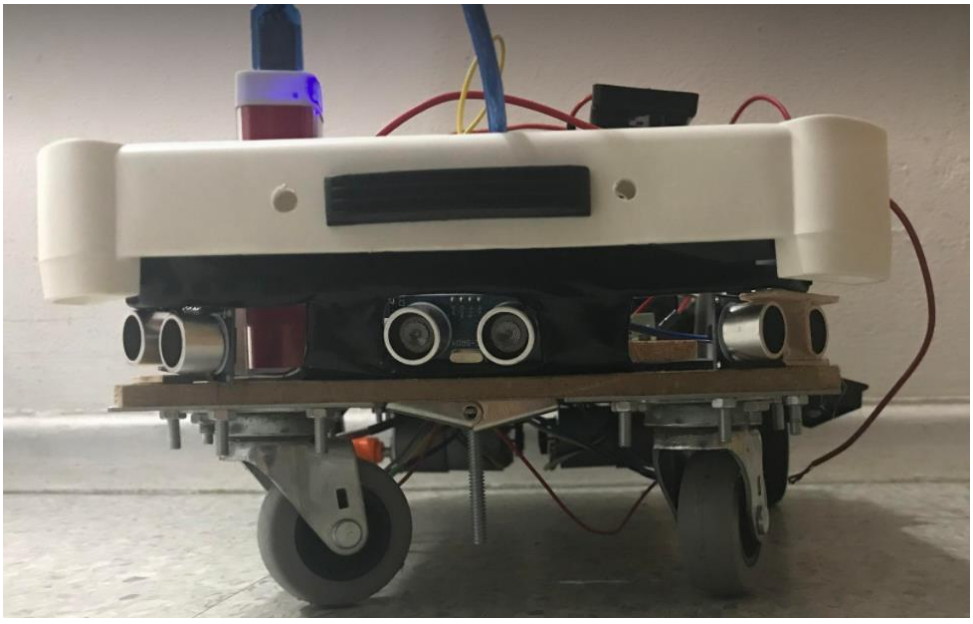
Battery

Apparatus

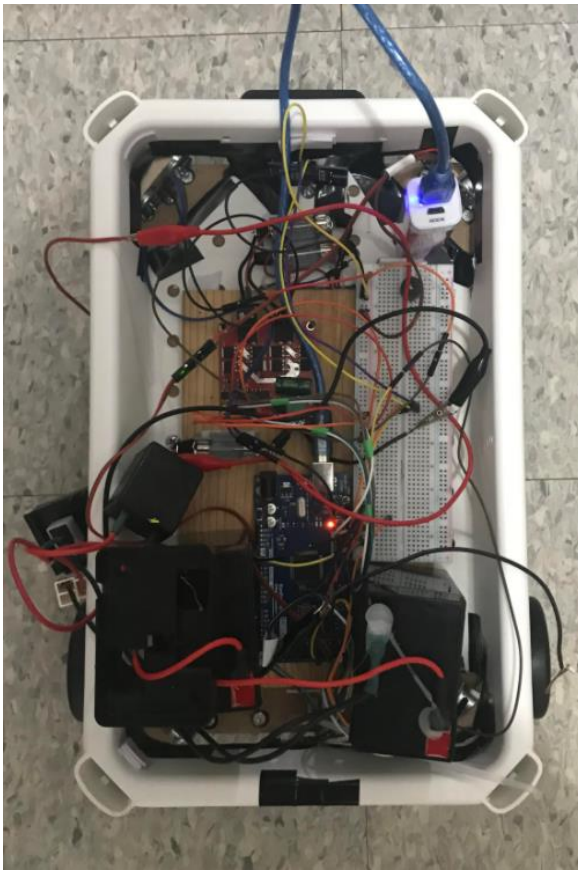
Back view



Front view

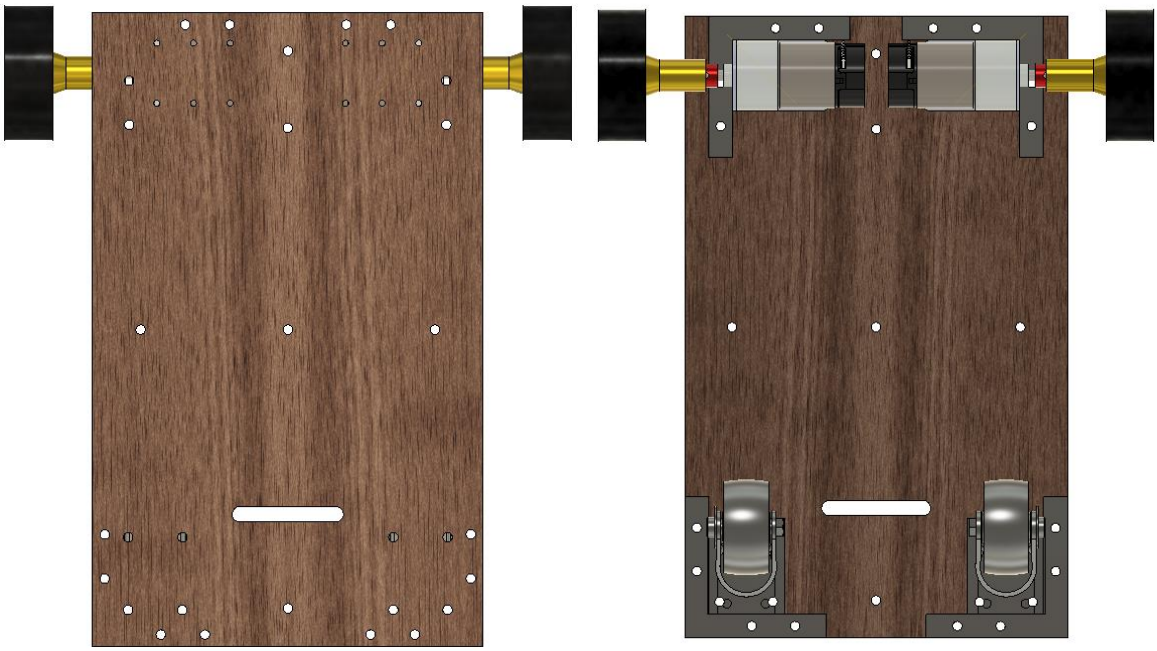


Top view

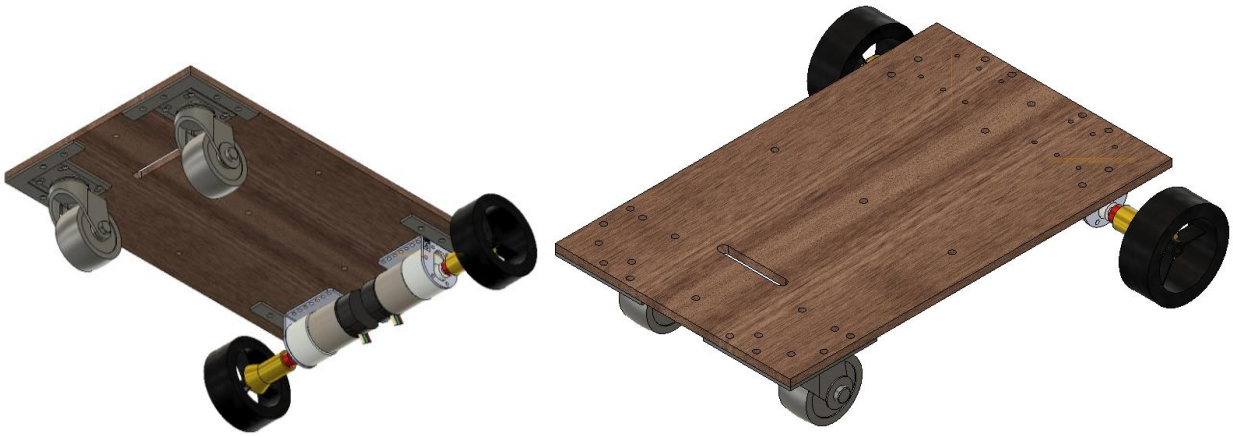


Side view

Top & Bottom View



Bottom Left side & Isometric View



Left Side View

Code

Code to find counts /sec, rpm, m/s

```
#include <util/atomic.h>

//pins that the motor and motor driver
//Left motor
#define PWM 4
#define ENCA 18 //yellow
#define ENCB 19  //white
#define IN1 3
#define IN2 2

//right motor
#define PWMr 13
#define ENCAr 20 //yellow
#define ENCBr 21  //white
#define IN1r  6
#define IN2r 7
//globals for velocity calcualtions
long prevT = 0;//previous time
long posPrev = 0;//previous position
volatile int pos_i = 0;
volatile float velocity_i = 0;
volatile long prevT_i = 0;
long prevTr = 0;//previous time
long posPrevr = 0;//previous position
volatile int pos_ir = 0;
volatile float velocity_ir = 0;
volatile long prevT_ir = 0;

float eprev = 0;//previous error left motor
float eintegral = 0;
float eprev2 = 0;//previous error right motor
float eintegral2 = 0;
unsigned long myTime;

void setup() {
```

```
  Serial.begin(9600);
  pinMode(ENCA, INPUT);
  pinMode(ENCB, INPUT);
  pinMode(PWM, OUTPUT);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(ENCAr, INPUT);
  pinMode(ENCBr, INPUT);
  pinMode(PWMr, OUTPUT);
  pinMode(IN1r, OUTPUT);
  pinMode(IN2r, OUTPUT);
  // intrerrupt function is used to ensure the program can read the pulses from the encoder
  attachInterrupt(digitalPinToInterrupt(ENCA), readEncoder, RISING);
  attachInterrupt(digitalPinToInterrupt(ENCAr), readEncoderR, RISING);

  Serial.println("v1,v2r");
}

void loop() {

  myTime = millis();// * 0.001;
  // pwm value and to run the motor for three seconds

  int pwr = 99;// left motor 48
  int pwrr = 77.5; //right motor 41.25
  // int pwrr = 255/ 8.0 * micros()/ 1.0e6;
  //int pwr = 255/ 8.0 * micros()/ 1.0e6;
  int dir = 1; // this is the direction of the motor (FWR or BWD)
  setMotor(dir, pwr, PWM, IN1, IN2); // calling motor function for it to fun in the loop section
  int dirr = 1; // this is the direction of the motor (FWR or BWD)
  setMotorR(dirr, pwrr, PWMr, IN1r, IN2r); // calling motor function for it to fun in the loop section

  //read the position in an atomic block to avoid potential misreads
  int pos = 0;
  float velocity2 = 0;
  int posr = 0;
  float velocity2r = 0;
  // time difference
  long currT = micros();
  float deltaT = ((float) (currT - prevT)) / ( 1.0e6 );
  prevT = currT;
  // time difference
  long currTr = micros();
  float deltaTr = ((float) (currTr - prevTr)) / ( 1.0e6 );
  prevTr = currTr;
  ATOMIC_BLOCK(ATOMIC_RESTORESTATE) {
    pos = pos_i;
```

```
  velocity2 = velocity_i;
  posr = pos_ir;
  velocity2r = velocity_ir;
}

//compute velocity using method 1
//long currT = micros();
// float deltaT = ((float)(currT - prevT)) / (1.0e6);
float velocity1 = (pos - posPrev) / deltaT;
posPrev = pos;
prevT = currT;
//long currTr = micros();
//float deltaTr = ((float)(currTr - prevTr)) / (1.0e6);
float velocity1r = (posr - posPrevr) / deltaTr;
posPrevr = posr;
prevTr = currTr;
//convert count/s to RPM
float v1 = velocity1 / 1632.00 * 60.00;
float v2r = velocity1r / 1632.00 * 60.00;
//convert to m/s
float v1ms = (v1 / 60) * (2 * 3.14) * (0.035);
float v1msr = (v2r / 60) * (2 * 3.14) * (0.035);

//Serial.println(pos);
//Serial.print(posr);
/*Serial.print("Countspersecond: ");
  Serial.print("Left Motor: ");
  Serial.print(velocity1); Serial.println(" counts/sec"); //left motor
  Serial.print("               ");
  Serial.print("Right Motor: ");
  Serial.print(velocity1r); Serial.print(" counts/sec"); //right motor
  Serial.println(currT);
  Serial.println();
  Serial.println();
  Serial.print("Revolutions per minute: ");
  Serial.print("Left Motor: ");
  Serial.print(v1); Serial.println(" rpm"); //left motor
  Serial.print("                     ");
  Serial.print(" , ");
  Serial.print("Right Motor: ");
  Serial.print(v2r); Serial.print(" rpm"); //right motor
  Serial.print(" , ");
  Serial.println();
  Serial.println();
  Serial.print("Velocity: ");
  Serial.print("Left Motor: ");
  Serial.print(v1ms); Serial.println(" m/s");
  Serial.print("        ");
```

```
   Serial.print("Right Motor: ");
   Serial.print(v1msr); Serial.print(" m/s");
   Serial.println();
    //Serial.print(pwr);
   //Serial.print(pwrr);
  */

  //Serial.println();
  //delay(1000.00);
  plot("", myTime, false);
  plot("", v1, false);
  plot("", v2r, true);

}
void setMotor(int dir, int pwmVal, int pwm, int in1, int in2) {
  analogWrite(pwm, pwmVal);
  if (dir == 1) {
    digitalWrite(in1, HIGH);
    digitalWrite(in2, LOW);
  }
  else if (dir == -1) {
    digitalWrite(in1, LOW);
    digitalWrite(in2, HIGH);
  }
  else {
    digitalWrite(in1, LOW);
    digitalWrite(in2, LOW);
  }
}
void setMotorR(int dirr, int pwmValr, int pwm, int in1r, int in2r) {
  analogWrite(pwm, pwmValr);
  if (dirr == 1) {
    digitalWrite(in1r, HIGH);
    digitalWrite(in2r, LOW);
  }
  else if (dirr == -1) {
    digitalWrite(in1r, LOW);
    digitalWrite(in2r, HIGH);
  }
  else {
    digitalWrite(in1r, LOW);
    digitalWrite(in2r, LOW);
  }
}
//reads the direction of the motor
void readEncoder() {
  int b = digitalRead(ENCB);
  int increment = 0;
```

```
  if (b > 0) {
    increment = 1;
  }
  else {
    increment = -1;
  }
  pos_i = pos_i + increment;

  //compute velocity with method 2
  long currT = micros();
  float deltaT = ((float)(currT - prevT_i)) / (1.0e6);
  velocity_i = increment / deltaT;
  prevT_i = currT;
}
//reads the direction of the motor
void readEncoderR() {
  int c = digitalRead(ENCBr);
  int incrementr = 0;
  if (c > 0) {
    incrementr = 1;
  }
  else {
    incrementr = -1;
  }
  pos_ir = pos_ir + incrementr;

  //compute velocity with method 2
  long currTr = micros();
  float deltaTr = ((float)(currTr - prevT_ir)) / (1.0e6);
  velocity_ir = incrementr / deltaTr;
  prevT_ir = currTr;
}
void plot(String label, float value, bool last) {//ploting purposes
  Serial.print(label);//maybe an emoty string
  if (label != "")Serial.print(":");
  Serial.print(value);
  if (last == false)Serial.print(",");
  else Serial.println();
}
```

Code to find PID constants

```
#include <util/atomic.h> // For the ATOMIC_BLOCK macro

//pins that the motor and motor driver
#define PWM 4
#define ENCA 18 //yellow
#define ENCB 19  //white
```

```
#define IN1 3
#define IN2 2

volatile int posi = 0; // specify posi as volatile:
https://www.arduino.cc/reference/en/language/variables/variable-scope-qualifiers/volatile/
long prevT = 0;
float eprev = 0;
float eintegral = 0;

void setup() {
  Serial.begin(9600);
  pinMode(ENCA,INPUT);
  pinMode(ENCB,INPUT);
  attachInterrupt(digitalPinToInterrupt(ENCA),readEncoder,RISING);
  pinMode(PWM,OUTPUT);
  pinMode(IN1,OUTPUT);
  pinMode(IN2,OUTPUT);
  Serial.println("target pos");
}

void loop() {

  // set target position
  int target = 1200;
  //int target = 500*sin(prevT/1e6);

  // PID constants
  float kp = 0.750;
  float kd = 0.045;
  float ki = 0.0;

  // time difference
  long currT = micros();
  float deltaT = ((float) (currT - prevT))/( 1.0e6 );
  prevT = currT;

  // Read the position in an atomic block to avoid a potential
  // misread if the interrupt coincides with this code running
  // see: https://www.arduino.cc/reference/en/language/variables/variable-scope-qualifiers/volatile/
  int pos = 0;
  ATOMIC_BLOCK(ATOMIC_RESTORESTATE) {
    pos = posi;
  }

  // error
  int e = pos - target;

  // derivative
```

```
  float dedt = (e-eprev)/(deltaT);

  // integral
  eintegral = eintegral + e*deltaT;

  // control signal
  float u = kp*e + kd*dedt + ki*eintegral;

  // motor power
  float pwr = fabs(u);
  if( pwr > 255 ){
    pwr = 255;
  }

  // motor direction
  int dir = 1;
  if(u<0){
    dir = -1;
  }

  // signal the motor
  setMotor(dir,pwr,PWM,IN1,IN2);

  // store previous error
  eprev = e;

  Serial.print(target);
  Serial.print(" , ");
  Serial.print(pos);
  Serial.println();
}

void setMotor(int dir, int pwmVal, int pwm, int in1, int in2){
  analogWrite(pwm,pwmVal);
  if(dir == 1){
    digitalWrite(in1,HIGH);
    digitalWrite(in2,LOW);
  }
  else if(dir == -1){
    digitalWrite(in1,LOW);
    digitalWrite(in2,HIGH);
  }
  else{
    digitalWrite(in1,LOW);
    digitalWrite(in2,LOW);
  }
}
```

```
void readEncoder(){
  int b = digitalRead(ENCB);
  if(b > 0){
    posi++;
  }
  else{
    posi--;
  }
}
```

Code to test Six ultrasonic sensors

```
const int echoPinL = 9;
const int trigPinL = 8;
const int echoPinR = 29;
const int trigPinR = 28;
const int echoPinF = 10;
const int trigPinF = 11;
const int echoPinBL = 25;
const int trigPinBL = 24;
const int echoPinBR = 27;
const int trigPinBR = 26;
const int echoPinB = 23;
const int trigPinB = 22;
unsigned long myTime;
void setup() {
  Serial.begin(9600);

  pinMode(echoPinL, INPUT);
  pinMode(trigPinL, OUTPUT);
  pinMode(echoPinR, INPUT);
  pinMode(trigPinR, OUTPUT);
  pinMode(echoPinF, INPUT);
  pinMode(trigPinF, OUTPUT);
  pinMode(echoPinBL, INPUT);
  pinMode(trigPinBL, OUTPUT);
  pinMode(echoPinBR, INPUT);
  pinMode(trigPinBR, OUTPUT);
  pinMode(echoPinB, INPUT);
  pinMode(trigPinB, OUTPUT);
  //Serial.print("distanceL, distanceF, distanceR, distanceLB, distanceB, distanceBR");
}

void loop() {
  myTime = millis();
  float distanceF = realSensorDataF();
  float distanceL = realSensorDataL();
  float distanceR = realSensorDataR();
  float distanceB = realSensorDataB();
```

```
 float distanceBL = realSensorDataBL();
 float distanceBR = realSensorDataBR();
// Serial.print(myTime);
 Serial.println("distanceL");
 //Serial.print(" , ");
 Serial.println(distanceL);
// Serial.print(" , ");
 Serial.println("distanceF");
 Serial.println(distanceF);
 //Serial.print(" , ");
 Serial.println("distanceR");
 Serial.println(distanceR);
 //Serial.print(" , ");
  Serial.println("distanceBL");
 Serial.println(distanceBL);
// Serial.print(" , ");
 Serial.println("distanceB");
 Serial.println(distanceB);
 //Serial.print(" , ");
 Serial.println("distanceBR");
 Serial.println(distanceBR);
 //Serial.print(" , ");

 Serial.println("");
 delay(1000);
}
float realSensorDataR() {
 digitalWrite(trigPinR, LOW);
 delayMicroseconds(2);
 digitalWrite(trigPinR, HIGH);
 delayMicroseconds(10);
 digitalWrite(trigPinR, LOW);
 float distanceR = pulseIn(echoPinR, HIGH) / 58.00;
 return distanceR;
}
float realSensorDataL() {
 digitalWrite(trigPinL, LOW);
 delayMicroseconds(2);
 digitalWrite(trigPinL, HIGH);
 delayMicroseconds(10);
 digitalWrite(trigPinL, LOW);
 float distanceL = pulseIn(echoPinL, HIGH) / 58.00;
 return distanceL;
}
float realSensorDataF() {
 digitalWrite(trigPinF, LOW);
 delayMicroseconds(2);
 digitalWrite(trigPinF, HIGH);
```

```
  delayMicroseconds(10);
  digitalWrite(trigPinF, LOW);
  float distanceF = pulseIn(echoPinF, HIGH) / 58.00;
  return distanceF;
}
float realSensorDataB() {
  digitalWrite(trigPinB, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPinB, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPinB, LOW);
  float distanceB = pulseIn(echoPinB, HIGH) / 58.00;
  return distanceB;
} float realSensorDataBR() {
  digitalWrite(trigPinBR, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPinBR, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPinBR, LOW);
  float distanceBR = pulseIn(echoPinBR, HIGH) / 58.00;
  return distanceBR;
} float realSensorDataBL() {
  digitalWrite(trigPinBL, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPinBL, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPinBL, LOW);
  float distanceBL = pulseIn(echoPinBL, HIGH) / 58.00;
  return distanceBL;
}
void plot(String label, float value, bool last) {
  Serial.print(label);//maybe an emoty string
  if (label != "")Serial.print(":");
  Serial.print(value);
  if (last == false)Serial.print(",");
  else Serial.println();
}
```

Final operating code

```
#include <util/atomic.h>

// A class to compute the control signal
class SimplePID {
  private:
    float kp, kd, ki, umax; // Parameters
    float eprev, eintegral; // Storage

  public:
```

```
  // Constructor
  SimplePID() : kp(1), kd(0.025), ki(0), umax(255), eprev(0.0), eintegral(0.0) {}

  // A function to set the parameters
  void setParams(float kpIn, float kdIn, float kiIn, float umaxIn) {
    kp = kpIn; kd = kdIn; ki = kiIn; umax = umaxIn;
  }

  // A function to compute the control signal
  void evalu(int value, int target, float deltaT, int &pwr, int &dir) {
    // error
    int e = target - value;

    // derivative
    float dedt = (e - eprev) / (deltaT);

    // integral
    eintegral = eintegral + e * deltaT;

    // control signal
    float u = kp * e + kd * dedt + ki * eintegral;

    // motor power
    pwr = (int) fabs(u);
    if ( pwr > umax ) {
      pwr = umax;
    }

    // motor direction
    dir = 1;
    if (u < 0) {
      dir = -1;
    }

    // store previous error
    eprev = e;
  }

};

// How many motors
#define NMOTORS 2

// Pins
const int enca[] = {18, 20};
const int encb[] = {19, 21};
const int pwm[] = {4, 13};
const int in1[] = {3, 6};
```

```
const int in2[] = {2, 7};

// Globals
long prevT = 0;
volatile int posi[] = {0, 0};

// PID class instances
SimplePID pid[NMOTORS];
const int echoPinL = 9;
const int trigPinL = 8;
const int echoPinR = 29;
const int trigPinR = 28;
const int echoPinF = 10;
const int trigPinF = 11;
const int echoPinBL = 25;
const int trigPinBL = 24;
const int echoPinBR = 35;
const int trigPinBR = 34;
const int echoPinB = 23;
const int trigPinB = 22;
void setup() {
  Serial.begin(9600);
  pinMode(echoPinL, INPUT);
  pinMode(trigPinL, OUTPUT);
  pinMode(echoPinR, INPUT);
  pinMode(trigPinR, OUTPUT);
  pinMode(echoPinF, INPUT);
  pinMode(trigPinF, OUTPUT);
  pinMode(echoPinBL, INPUT);
  pinMode(trigPinBL, OUTPUT);
  pinMode(echoPinBR, INPUT);
  pinMode(trigPinBR, OUTPUT);
  pinMode(echoPinB, INPUT);
  pinMode(trigPinB, OUTPUT);
  for (int k = 0; k < NMOTORS; k++) {
    pinMode(enca[k], INPUT);
    pinMode(encb[k], INPUT);
    pinMode(pwm[k], OUTPUT);
    pinMode(in1[k], OUTPUT);
    pinMode(in2[k], OUTPUT);

    pid[k].setParams(1, 0.025, 0, 255);
  }



}
int target[NMOTORS];
```

```
void loop() {
  float distanceF = realSensorDataF();
  float distanceL = realSensorDataL();
  float distanceR = realSensorDataR();
  float distanceB = realSensorDataB();
  float distanceBL = realSensorDataBL();
  float distanceBR = realSensorDataBR();
  /*
    Serial.println("distanceL");
    Serial.println(distanceL);
    Serial.println("distanceR");
    Serial.println(distanceR);
    Serial.println("distanceF");
    Serial.println(distanceF);
    Serial.println("distanceBL");
    Serial.println(distanceBL);
    Serial.println("distanceBR");
    Serial.println(distanceBR);
    Serial.println("distanceB");
    Serial.println(distanceB);
    Serial.println("cm");
    delay(2000);
  */
  // set target position

  //target[0] = 99; //42.6
  //target[1] = 77.5; //40


  if (distanceF < 15) {//move backwards
    target[0] = -99;
    target[1] = -77.5;
    //if (distanceBL < 15) { //turn right if the distance of back left is too close
      //target[0] = 99;
      //target[1] = -77.5;
    //}
    //if (distanceBL < 15 && distanceB < 15 ) { //turn right if the distance of back left and back  is too close
     // target[0] = 99;
      //target[1] = -77.5;
    //}

    //if (distanceBR < 15 ) { //turn left if the distnace of back left is too close
    //  target[0] = -99;
     // target[1] = 77.5;
    //}
    //if (distanceBR < 15 && distanceB < 15 ) { //turn left if the distance of back left and back  is too close
     // target[0] = -99;
     // target[1] = 77.5;
```

```
  //}
  // if (distanceB < 15) { //move foward slowly if the distance of the back is low
    // target[0] = 99;
    // target[1] = 77.5;

 } else if (distanceR < 8 ) {//turn right
   target[0] = -120;
   target[1] = 77.5;


 } else if (distanceL < 8) { // move left
   target[0] = 120;
   target[1] = -77.5;
 }

 else {

   target[0] = 99;
   target[1] = 77.5;
 }

// target[0] = 99;//750*sin(prevT/1e6);
// target[1] = 77.5;//-750*sin(prevT/1e6);

 // time difference
 long currT = micros();
 float deltaT = ((float) (currT - prevT)) / ( 1.0e6 );
 prevT = currT;


 // loop through the motors
 for (int k = 0; k < NMOTORS; k++) {
   int pwr, dir;
   // evaluate the control signal
   pid[k].evalu(posi[k], target[k], deltaT, pwr, dir);
   // signal the motor
   setMotor(dir, pwr, pwm[k], in1[k], in2[k]);
 }


}

void setMotor(int dir, int pwmVal, int pwm, int in1, int in2) {
 analogWrite(pwm, pwmVal);
 if (dir == 1) {
   digitalWrite(in1, HIGH);
   digitalWrite(in2, LOW);
 }
```

```
  else if (dir == -1) {
    digitalWrite(in1, LOW);
    digitalWrite(in2, HIGH);
  }
  else {
    digitalWrite(in1, LOW);
    digitalWrite(in2, LOW);
  }
}
float realSensorDataR() {
  digitalWrite(trigPinR, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPinR, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPinR, LOW);
  float distanceR = pulseIn(echoPinR, HIGH) / 58.00;
  return distanceR;
}
float realSensorDataL() {
  digitalWrite(trigPinL, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPinL, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPinL, LOW);
  float distanceL = pulseIn(echoPinL, HIGH) / 58.00;
  return distanceL;
}
float realSensorDataF() {
  digitalWrite(trigPinF, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPinF, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPinF, LOW);
  float distanceF = pulseIn(echoPinF, HIGH) / 58.00;
  return distanceF;
}
float realSensorDataB() {
  digitalWrite(trigPinB, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPinB, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPinB, LOW);
  float distanceB = pulseIn(echoPinB, HIGH) / 58.00;
  return distanceB;
} float realSensorDataBR() {
  digitalWrite(trigPinBR, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPinBR, HIGH);
```

```
  delayMicroseconds(10);
  digitalWrite(trigPinBR, LOW);
  float distanceBR = pulseIn(echoPinBR, HIGH) / 58.00;
  return distanceBR;
} float realSensorDataBL() {
  digitalWrite(trigPinBL, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPinBL, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPinBL, LOW);
  float distanceBL = pulseIn(echoPinBL, HIGH) / 58.00;
  return distanceBL;
}
```