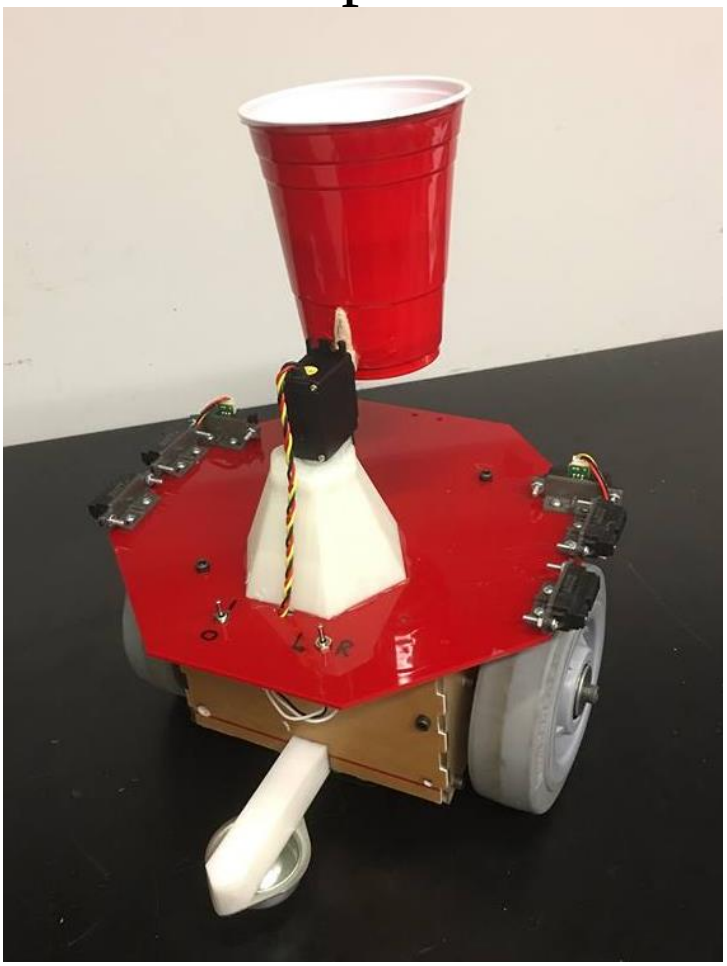
StopBot



Pranav Joneja, Jihyun Jung, Jihu Kim, Qing Xu

ME 412 - Autonomous Mobile Robots
Professor Ericson Mar
The Cooper Union for the Advancement of Science and Art
December 19, 2017

Abstract

"StopBot" is an autonomous mobile robot that can maneuver in response to its surroundings without external user command. StopBot is created to win the competitive 2017 Robot Tank Battle for the class ME412 taught by Professor Mar. In this battle, StopBot will safely navigate around the arena safely and shoot ping pong balls at the enemies or their base. StopBot is challenged to navigate away from home base, past obstacles, and shoot ping pong ball at an IR beacon at the opposing robot or its base, and then successfully return home.

Our strategy is to design StopBot to follow the wall until it arrives at the opposing robot's base. Sensors along the edges of the robot help it sense the arena with a cup launcher holding the ping pong ball. The strategy proved to be very effective. There were some small changes to the octagon roof and a change in our launcher design. Regardless, StopBot won the Tank Battle for 2017.

Table of Contents

Introduction

The Robot Tank Battle is an annual autonomous robotics competition by The Cooper Union. The battle is one on one battle held in a six by ten feet arena. Each robot starts at each end of the arena. Two robots will verse each other to see which can better avoid obstacles and shoot ping pong balls. The robot with the most points in five minutes wins. To learn more about the specific rules of the competition, they are listed in the Appendix.

Robots that participate in the Robot Tank Battle have the choice to design a strategy that follows the rules of the competition. Some strategies include randomly running until it spots an IR beacon, electronically mapping the entire arena, or wall-following. After some discussion, wall-following seemed to be the most reliable strategy, as it did not waste any time.

Robots also are required to have a platform to place the IR beacon. IR beacon is placed for the robots to sense the enemy and enemy territory.

Strategy

The goal of StobBot's strategic design is to reach to the opposing base as fast as possible and earn points by hitting the opposing base with the launcher. "Wall-following" strategy is integrated to StopBot to achieve this goal. This strategy minimizes the chances for the robot to be lost and confused in the middle of arena. Wall-following also allows for simple code in robot movements. The robot is programmed to mainly focus on maintaining a certain distance away from the wall. Therefore, StopBot will always follow the arena. If it finds itself approaching a corner, it will turn at an angle and find the perpendicular wall. It will continue on this way until it sees an IR beacon, either from opposing robot or from the opposing base. The sensors are placed on the edges of an octagon shaped platform. Octagon shape allows the robot to have enough sensors to avoid hitting corners while turning. StopBot is also designed to have smallest possible dimensions to not only efficiently avoid physical obstacles in the arena but also to expose a smaller target for the opposing robots to hit.

Once the infrared beacon on the enemy or enemy base is sensed, StopBot starts its launcher. The launcher is placed at the very top of the robot, where it can be easily reloaded with the next ping pong ball. The launcher tilts to let go of the ball, which rolls into the opposing base or the robot.

Mechanical Design

Body

The dimensions of the body can fit in a 12” x 12” x 10” box, which is the maximum allowed robot size. The body of the robot was made with .125-inch-thick acrylic. Shown in Figure 3.1, the body is made up of a rectangular prism with an octagon roof. For more CAD files, please refer to the Figures 5.2 - 5.8 in the Appendix. Different colored acrylic, like red and white, were used for decoration to go with the name StopBot.

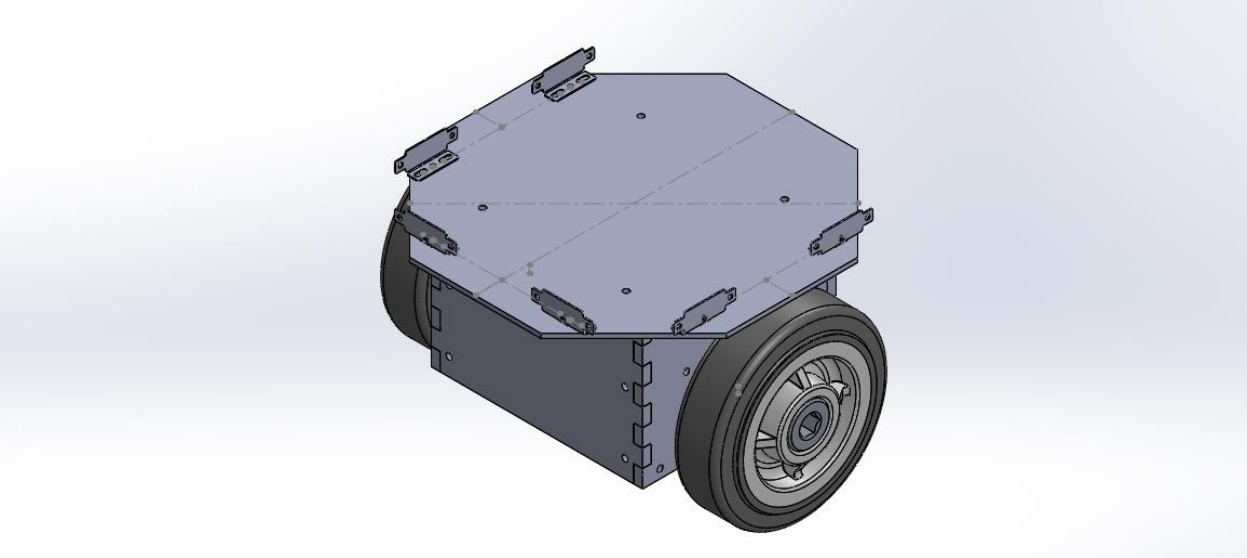


Figure 3.1: *An early 3D model of the body of StopBot*

The walls are rectangles laser cut out of acrylic to build into a box. Figure 3.2, a picture of one of the side walls, will show that jigsaw patterns were attached to the sides to fit the walls better together. Little holes were placed near the edges of the walls for screws and wires to go through.

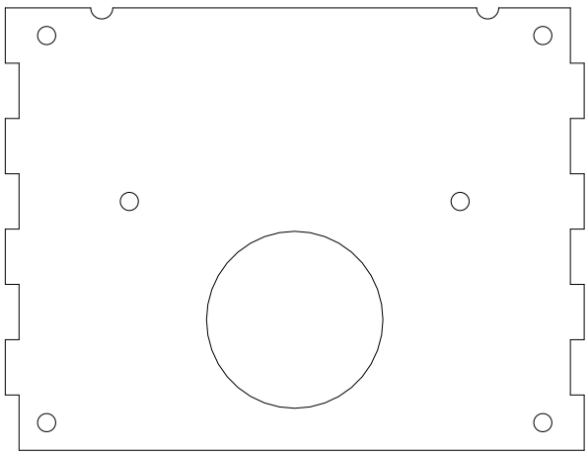


Figure 3.2: *The side wall, with holes to give space to screws, wires, and a wheel*

Just for the side walls, a large hole was also placed in the middle of the wall to give space for the wheels. The back wall also included a rectangular hole, shown in Figure 3.3, to have better access to the Arduino.

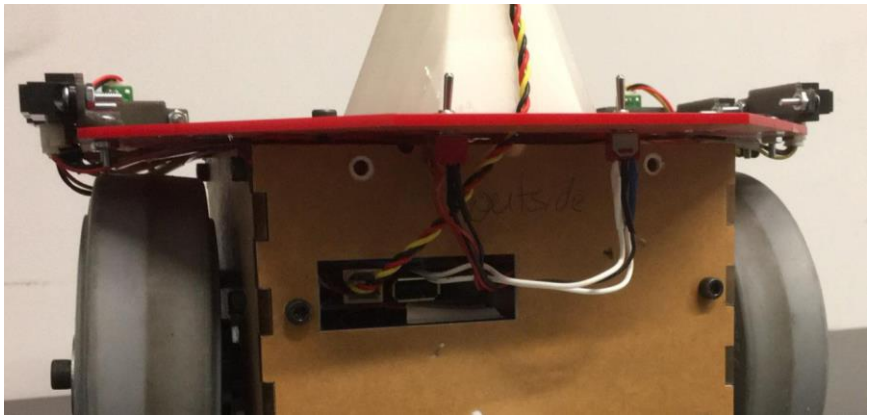


Figure 3.3: *The back wall has a rectangular hole to allow for access to Arduino*

All of the walls are attached using delrin connectors. The connectors were able to be drilled into and tapped to allow for stronger connections. This eliminated the need for nuts as the delrin acted as the nut itself holding onto the threaded bolts. An example of the delrin connector is shown in Figure 3.4.

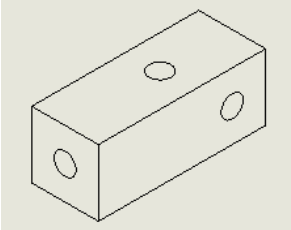


Figure 3.4: *An example of one of the delrin connectors connecting three walls*

To hold all the electronics, motors, wheels, and batteries, StopBot needs to have a lot of space. This leaded to having a multiple-layered robot. Figure 3.5 shows a cross section of StopBot. The cross-section view shows the different levels within the body to allow for spacing of motors, batteries, electronics, sensors, and the launcher.

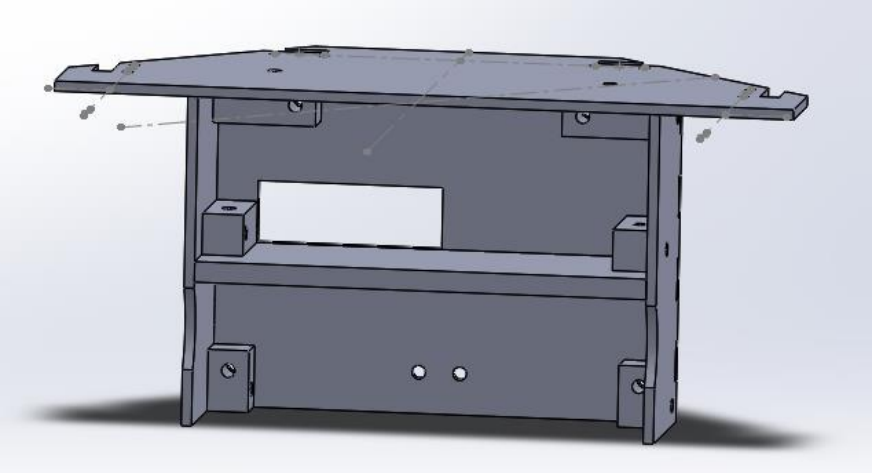


Figure 3.5: *Cross section of StopBot, showing two levels*

The batteries, motors, and wheels are all attached at the bottom side of the lowest level. This level is also made of delrin to eliminate the need for nuts. The electronics, including an Arduino and breadboard, is attached to the top of this delrin piece.

The top of StopBot is a red acrylic regular-octagon roof with 3.73-inch sides. On it lies the launcher with multiple sensors around it. Figure 3.6 shows that the laser cut drawing for the roof. The drawing shows that the roof had many holes. The holes served to allow for easy and clean wiring for the sensors and switches.

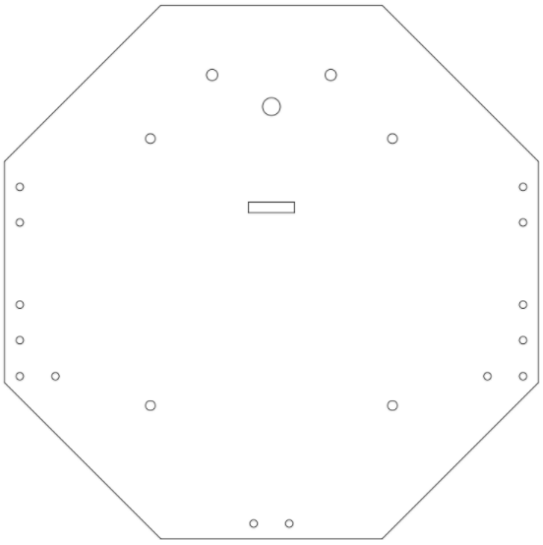


Figure 3.6: *The octagon roof with holes allowing for the sensors, switches, and reattachment*

Some of the holes on the roof were also used to mount the sensors. These were mounted on the roof using 3D printed mounts, shown in Figure 3.7. The mounts hold the sensors up perpendicularly so that the sensors could detect the world around them.

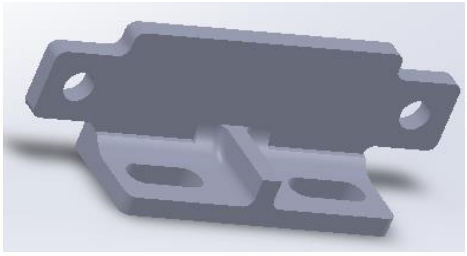


Figure 3.7: *The sensor mount*

Motors and Wheels

The motors we chose were 12V medium power DC motors from Pololu. The motors have a good tradeoff between torque (18 kg-cm) and speed (43 RPM). Torque is necessary to accelerate the robot, as the robot is heavy. Speed is necessary to compensate for the inefficient wall-following strategy (which necessitates taking a long path around the whole arena.

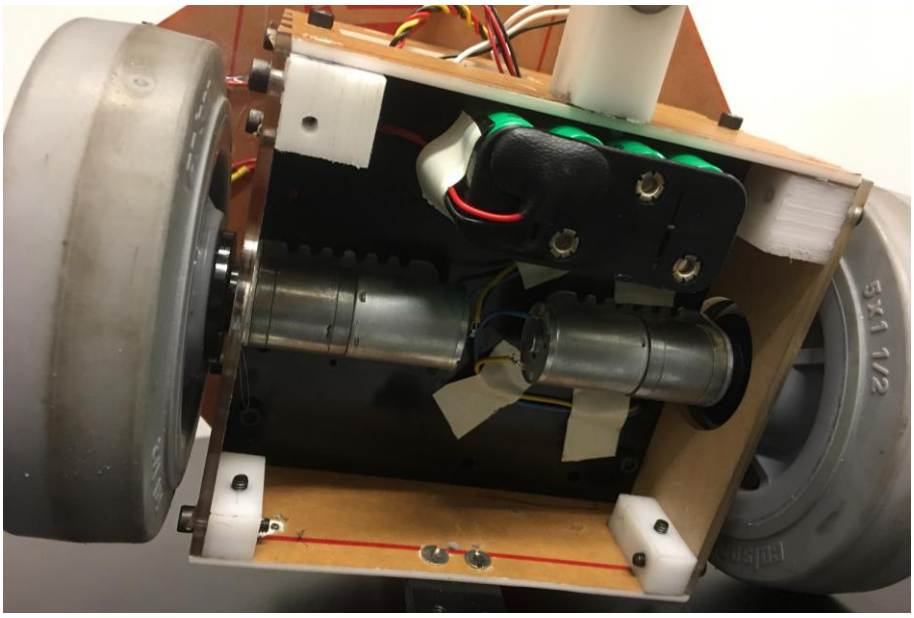


Figure 3.8: *Motor and wheel assembly*

The motors were attached to two Colson Performa wheels, shown in Figure 3.8. These 5 inch wheels are heavy-duty, high-grip rubber wheels. This means that they will have a high friction with the floor. A hexagonal aluminum shaft connected the wheels and motors while the motors were held by mounting brackets. The shaft is shown in Figure 3.9.

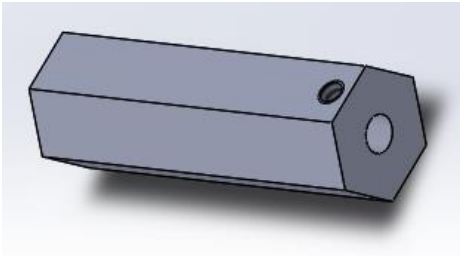


Figure 3.9: *The hexagonal aluminum shaft used to connect the motors and wheels*

Launcher

The launching mechanism sits on the top of StopBot. The launcher is made up of a base, a servo motor, a cup, which are all shown in Figure 3.10. A hexagonal cone shape acts as the launcher base and is attached to the octagon roof to rise the launching mechanism higher. The base holds the cup up higher to give the cup more room to rotate.

Figure 3.10: *The launcher*

A servo motor lays on top of the base and can move 180 degrees. The arm sticking out of the motor is attached to the cup so that when the motor turns, the cup turns as well.

Electronics

Hardware Components

Arduino Mega

The Arduino Mega is a powerful microcontroller that serves as the main component to control the movement of the robot and its reactions to obstacles. The number of input and output pins on the Mega were enough to receive input from all sensors and to transmit consistent output flow.

H-Bridge

An integrated dual h-bridge (model l298n) is connected to Arduino Mega to send StopBot's control input to the motors.

Infrared Range Finders

StopBot has total of six IR range finders to achieve wall-following strategy. These sensors detect the wall for the robot to navigate at a constant distance away from the wall.

Four Sharp GP2Y0A41SK0F Infrared Range Finders are used to detect with measurement distance of 4cm - 30cm. This specification is ideal to successfully achieve a follow the wall with a small distance, leading to better obstacle avoidance because StopBot can choose a Two of these sensors are mounted on the left edge of the top platform and oriented to the left side of the robot. They sense the wall from left side when StopBot is selected to navigate in clockwise direction around the arena. The other two sensors are mounted on the right-side edge of the platform. They will sense the wall from the right side when StopBot is selected to navigate counter clockwise around the arena.

Two more IR range finders were mounted and oriented to the front of the robot to detect any obstacles within a selected threshold parameter. These have a sensing distance of 10cm-80cm, which is better suited for “seeing” front obstacles early and successfully avoiding a collision. When these sensors detect that StopBot will encounter any obstacle ahead, StopBot rotates until the sensors no longer detect the obstacle. Once front sensors view is cleared from obstacles, StopBot will continue to perform the wall-following algorithm.

Infrared Phototransistor

IR phototransistors are used to detect infrared from the IR beacon on opposing robot and its base. One phototransistor is placed on the right and left edge of the platform (two in total).

Circuit Layout

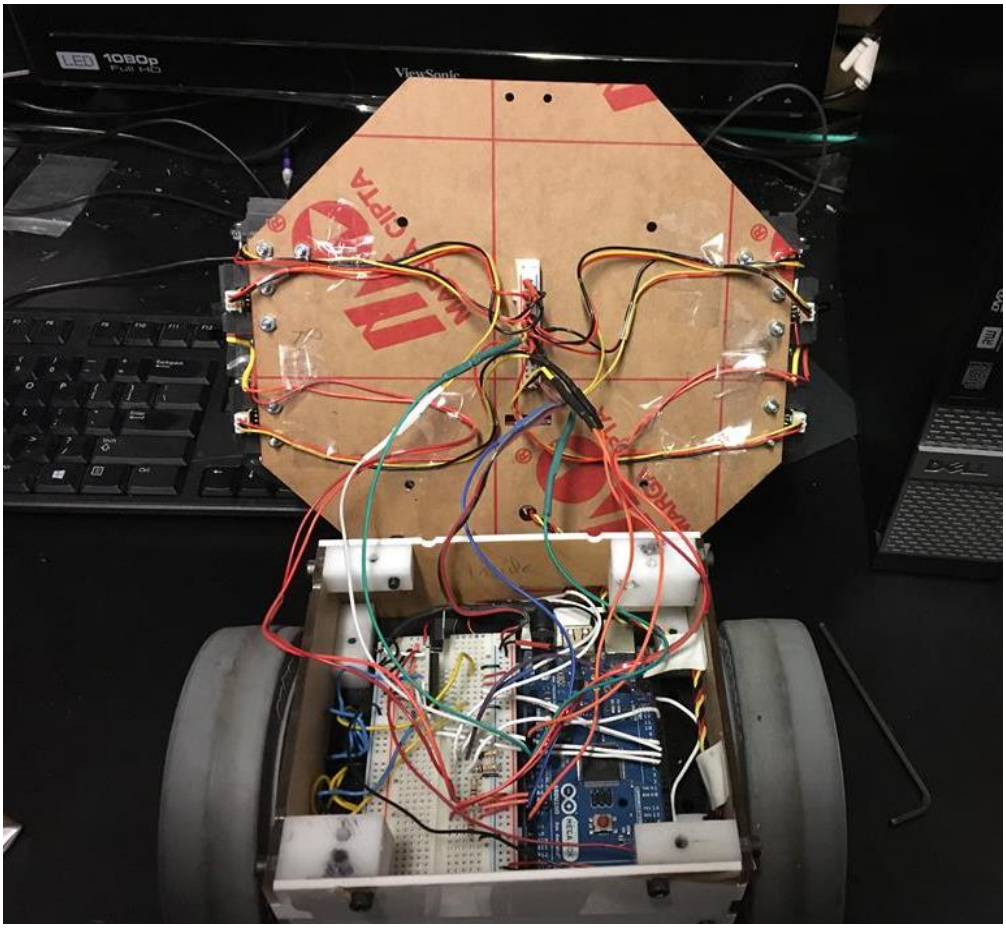


Figure 4.1: *The inside of StopBot, showing the electrical components*

Figure 4.1 above demonstrates how electrical components are connected to inside the platform. Breadboard for ground and power is placed on the bottom of the top platform to organize wires better. The wires are also color coded to help identify where each wire belongs. (e.g. orange wires connect the IR photoresistor to the board, green wires connect front IR sensors) The white and blue wires are for the switch that changes the direction of wall-following to either right or left. Wires for the on/off switch cut off current completely from the battery.

Algorithm and Programming

The entire code could be found in the Appendix.

Process Flow

Figure 4.2 clearly illustrates the logic StopBot follows. Note that the loop does not need to go to a stop since it the robot will be picked up when it returns to the arena. Also note that the ball-launching loop is only going to be performed once, because ping-pong ball has to be reloaded when the robot returns to the home base.

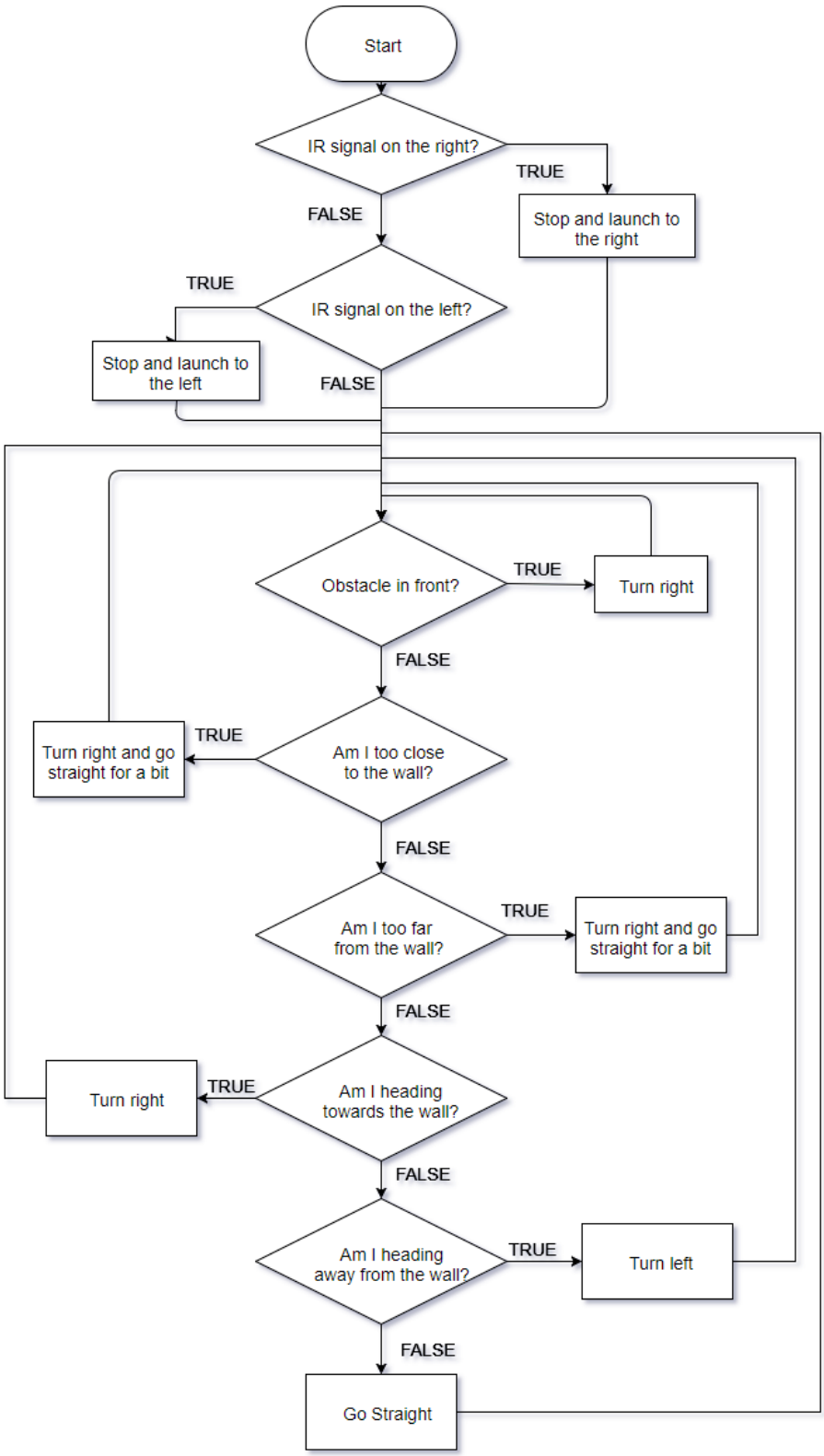


Figure 4.2 Program Flowchart

Assembly

Performance

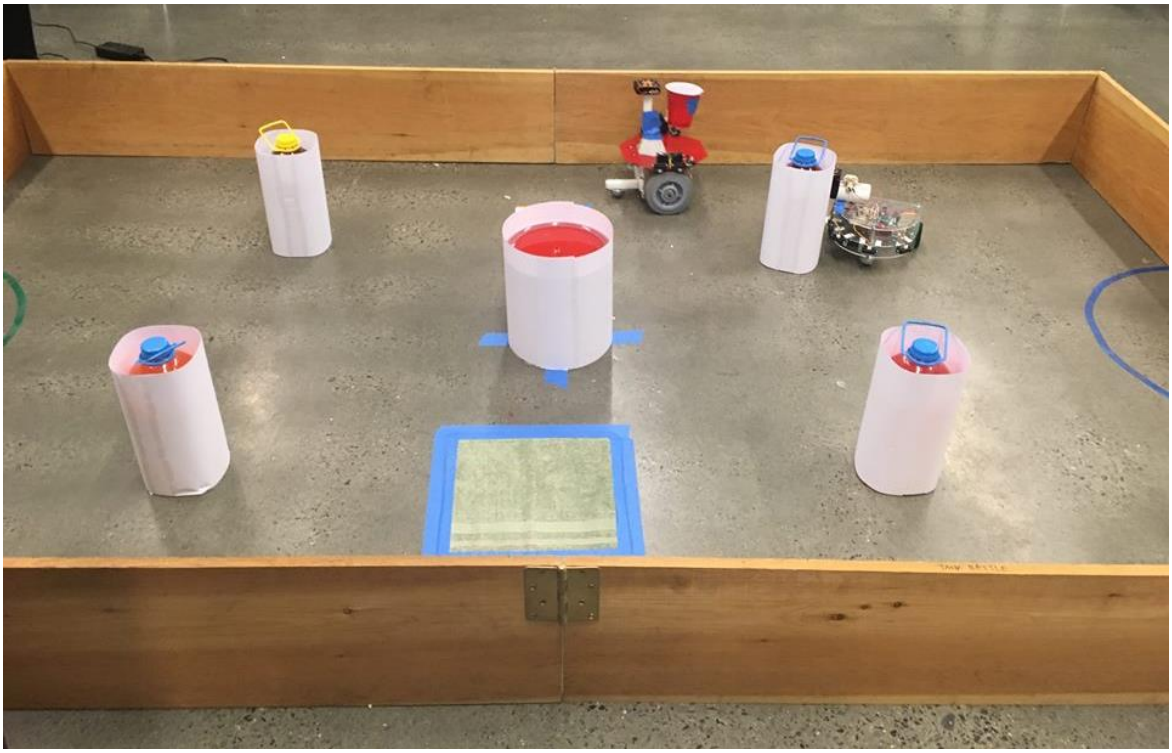


Figure 5.1: *StopBot going around the arena with another enemy robot in its path*

StopBot was undefeated in the competition, with four straight wins. A photo taken during one of those battles is shown in Figure 5.1. It successfully completed the required tasks: obstacle avoidance, shoot ping pong ball at opposing base, and return to its home base. In fact, in the last round, StopBot achieved all of these tasks in the same round.

However, because StopBot was designed to focused on shooting ping pong ball at opponent's base, the IR phototransistors were not mounted at an ideal place to shoot the ping pong ball at opposing robot. In order to achieve this task, IR phototransistors were replaced couple inches higher by simply pulling out the connected wires. Also, during the first round, StopBot was stuck in between the wall and the obstacle and lost points for collision. However,

after adjusting the thresholds for the sensors, StopBot was able to navigate through the obstacles without a problem from then on.

In the last round, StopBot managed to get 70 points by hitting the enemy robot twice as well as the enemy home base three times. By the last round, the pit stop used to reload the ping pong ball was used as a chance to lower and higher the IR phototransistors so that StopBot could hit both the enemy robot and the enemy base. Using this tactic, the last round was the most successful of all of StopBot's four rounds

StopBot placed first for the Robot Tank Battle 2017. It earned an average score of 30 with 70 being the highest score. Its average score per round was the highest ever compared to average scores from past Robot Tank Battles.

Fixes

StopBot had some changes before becoming the robot it is now. The front caster wheel originally decided for the robot proved to be too small. A larger caster wheel gave the robot and increased ability to go over floor obstacles. A tail made of delrin juts out from the back, shown in Figure 5.2. The new caster wheel was attached to the end of the tail to allow for more stability.

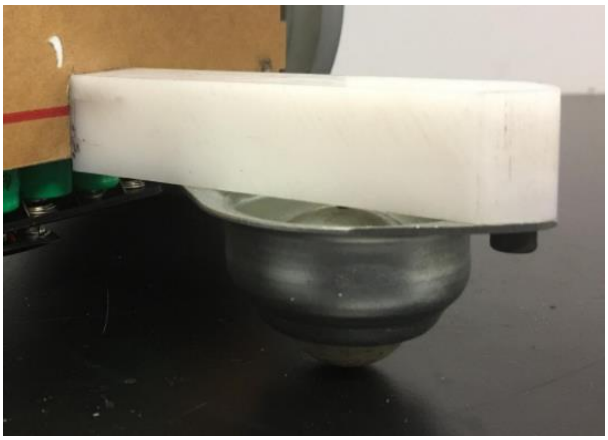


Figure 5.2: The back caster wheel connected with a delrin tail

The roof design had changed throughout the process of making StopBot. The original roof had space for six sharp IR sensors, two on the right, two on the left, and two on the front. StopBot, however, was not able to detect obstacles that appeared in the front left or front right corners from StopBot. To fix this, the two front sharp IR sensors were put right next and perpendicular to the left and right sensors, shown in Figure 5.3. Additional holes were also cut into the roof to allow room for the new switches.

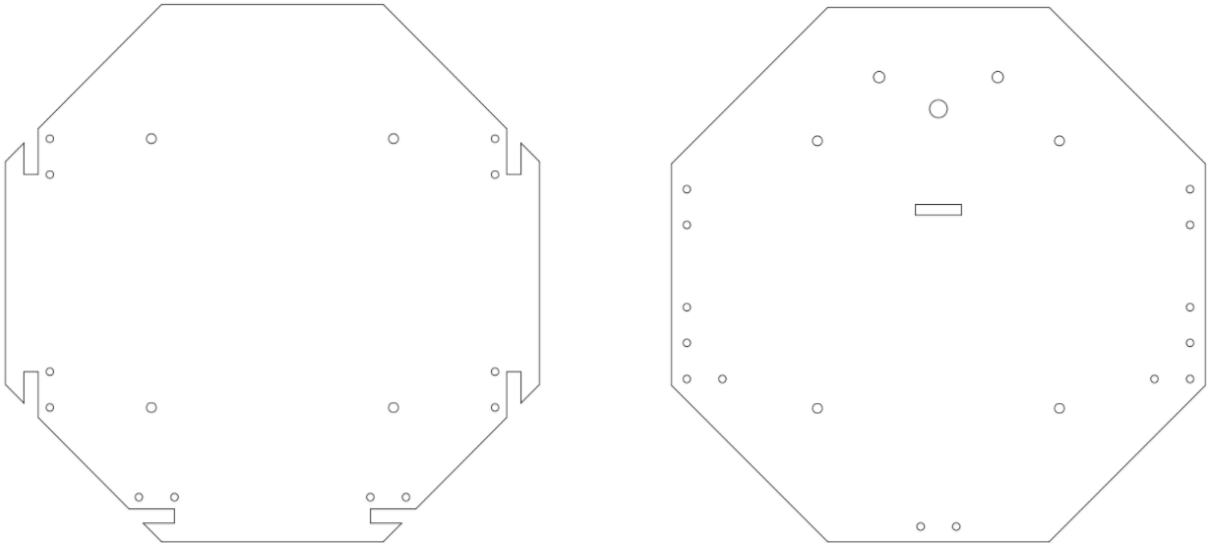


Figure 5.2: *The old (left) and new (right) octagonal roofs*

The launcher had initially been designed to be put on a rectangular platform that held the servo motor and a hexagonal cup. When 3D-printed, however, the space between the cup and the platform was too small to allow for the cup to rotate and drop the ping pong ball. The new launcher used the hexagon cup as the base to hold the launcher up. The new ping pong ball holder was replaced with a regular plastic cup.

Appendix

Robot Tank Battle Rules

The rules are taken directly from this link, shared by Professor Ericson Mar, the host of the Robot Tank Battle Rules.

Game Objective

The objective of the game is to have your autonomous robot score as many points as it can by shooting 40 mm ping pong balls at the opponent's home base and/or robot. In order to accomplish this, your robot has to navigate the arena to get within range of the opponent's robot or the opponent's home base, fire a shot, and return to its own home dock for human reloading. Obstacles will be present in the arena at random locations for each round. Each robot team will play 4 rounds of 5 minutes each with a 5-minute rest period between rounds.

Arena Specifications

The battlefield will be a 6 feet x 10 feet rectangle enclosed by wooden walls approximately 1 foot high. The wall will be recessed slightly (0.25" to 1" deep) for lengths of 1 foot at the locations of the bases. The floor will be the floor usual classroom.

Home Base Specifications

Each home base will be a recessed 1-foot wall (as described above). There will be a hole cut out in center of the 1 foot by 1-foot home base wall where IR LED light source will shine through. There may be a plastic shield over this hole if a robot can shoot a ping pong ball at a velocity that may break the LED light source.

Obstacle Specifications

Obstacles will be placed at random areas in the arena. Each obstacle will be a white obstructive object with a height of approximately 1 foot or a small rug or other flat floor obstacle with a height of at most 3/8" inches (formerly 1/4" in 2015). Each obstructive obstacle will be a solid shaped approximately like a vertical projection of an oval or a rectangle (it can be circular or square). The floor obstacles will be any shape. There will be no direct line-of-sight from one base to the other without an obstructive obstacle in the way.

Dock Specifications

Each robot team will have a home dock. This will be a black 1-foot radius semicircle on the ground surrounding its home base.

Robot IR Heat Signature

A small ring (footprint of approximately 2.5 square inches) of IR LED's (Model: TSAL6400, peak emission = 940 nm) will be mounted on top of each robot. Each robot is required to have a small "platform" or "pole" to hold the simulated IR heat signature where the ring can be viewed from anywhere 360 degrees around the robot. This object will be counted as part of the robot for scoring (described below) purposes.

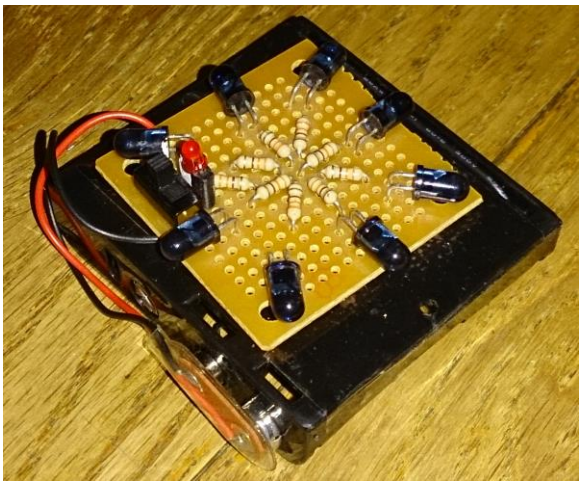


Figure 5.1: *An IR heat signature put on top of each robot*

Robot Rules

A robot must fit into a box 1 foot long by 1 foot wide by 10 inches high in all modes of operation. There may be no extension of robot parts beyond this space. If a team wishes to place a decorative item that has no effect on the game, then that item will not be counted as part of the robot for size purposes, but will be counted as part of the robot for scoring (described below) purposes.

A robot may not intentionally separate into multiple parts or leave any parts in the arena other than its ping pong ball.

A robot may not have any parts or devices that are designed to intentionally damage the arena, another robot, or humans.

Game Rules

A robot can carry one ball maximum at any given time.

A robot is considered "docked" when any part of the robot covers part of its home dock from a vertical perspective.

A robot may be re-loaded by the human team only when it is docked as described in "Robot Docking and Reloading" below.

A ball released by a robot is considered "dead" (i.e. it cannot score any points):

- after it touches an object other than the ground (e.g. it may be rolled but may not ricocheted off the wall towards the opponent)
- after it touches the releaser itself (e.g. the robot may not "guide" the ball with its own body)
- after it comes to a complete stop.
- after it has reached a height of 1 foot from the ground (no shooting over obstacles!)

All other times, the released ball is considered "live" (i.e. it can score points).

There may be ping pong balls strewn around the arena during the game. These are counted as part of the game. Consider this "debris". There will be an attempt to remove any debris from the arena if it is approximately 4 feet or a greater distance from any fielded robot. A team may remove debris from its home dock to clear the area for their robot to be placed, but

they cannot place it back somewhere else. Once dock debris is "disturbed" by the team's human actions it must be removed for good.

Ping Pong Ball Scoring

Opponent Base Hit (10 pts): When a robot releases a ball that touches its opponent's base while the ball is live, this is considered an opponent base hit.

Docked Robot Hit (10 pts): When a robot releases a ball that touches the docked opponent robot while the ball is live, this is considered a docked robot hit.

Fielded Robot Hit (20 pts): When a non-docked robot releases a ball that touches the non-docked opponent robot while the ball is live, this is considered a fielded robot hit. If the robot releasing the ball is docked, the team will NOT receive points. So, get your robot out into the field!

Robot self-hit (0 pts): If the live ball touches the robot where the ball originated from, that ball is considered "dead" (no penalty will be given to a robot hit by its own ball by catching up to it, but the robot may not "guide" the ball with its own body, either. See above Game Rules section).

Penalties

Home Base Hit (-10 pts): When a robot releases a ball that touches its own home base while the ball is live, this is considered a home base hit. Be careful not to shoot your own base! Collision (-10 pts. / round): When a robot comes into contact with anything other than a ball or the opponent robot, it is considered a collision. The penalty will only be assessed once per round.

Emergency Teleport (-10 pts.): If desired, a team may recover their non-docked robot as if it has docked (as in Robot Docking and Reloading below).

Robot Docking and Reloading

The human team may pick up its own robot when any part of the robot covers part of its home dock (from a vertical perspective). After the robot is "docked", the human team may perform any form of maintenance and reloading of the robot. The robot may then be placed anywhere entirely within the boundaries of its home dock and restarted for its next attack as long as the round has not expired. In the case where the opponent robot is stalled or roaming around the home dock such that the robot cannot be placed entirely within the dock area, it is allowed that the robot be placed in a location closest to the home base possible (even if part of it may protrude outside the dock boundary) and restarted for its next attack.

Parts Listing

Quantity	Part Number	Part	Model	Provider
2	3232	DC Motors	172:1 Metal Gearmotor	Pololu
1	2676	Motor Brackets	Metal Gearmotor Bracket Pair	Pololu
1	953	Caster Wheel	Metal Ball Caster	Pololu
1	2691	Caster Wheel	1" Plastic Ball Caster	Pololu
1	2462	IR Sensor	Vishay TSSP58038 IR Detector Module	Pololu
6	2464	Sharp Sensor	Sharp Analog Distance Sensor	Pololu
3	2678	Sensor Brackets	Sensor Bracket Pairs	Pololu
1	1040	Servo	Micro Servo	Pololu
1	2191	Arduino	Arduino Uno	Pololu
2	217-3430	Wheel Hub	Colson Live Hub	VEX Robotics
1	89845K81	Wheel Holder	Aluminum 1/2" Hex Size, 6" length	McMaster-Carr
2	CO4-5-446	Wheel	Colson Performa Wheel 5 x 1.5	RobotMarketplace
1	ACTLBLK0.250X12X12	Delrin Sheet	Delrin Sheet 0.250" X 12" X 12"	ePlastics
1		Red Acrylic	12" x 18" 1/8" Red Acrylic	Canal Plastics
1		Clear Acrylic	18" x 32" 1/8" Clear Acrylic	Canal Plastics
1		Ivory Acrylic	12" x 18" 1/8" Ivory Acrylic	Canal Plastics
1	47333	Connector	1/2" square Acetron Rod	US Plastic Corp
1	47335	Connector	3/4" square Acetron Rod	US Plastic Corp
1	BD070052	Battery	16 AA 3300 mAh 1.2 V Batteries	Tmart
1	2700382	Battery Holder	2 AA Battery Holder	Radio Shack
3	2700391	Battery Holder	4 AA Battery Holder	Radio Shack

CAD Files

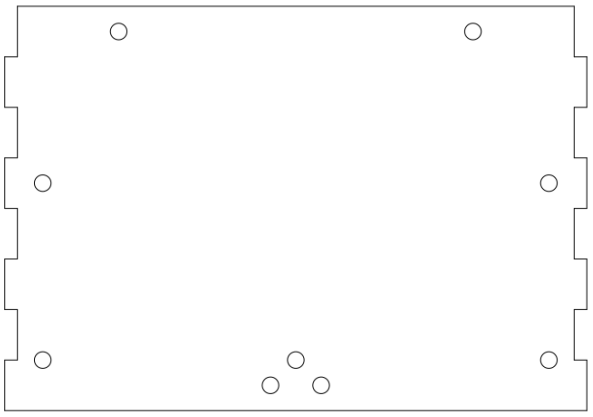


Figure 5.2: *The back wall*

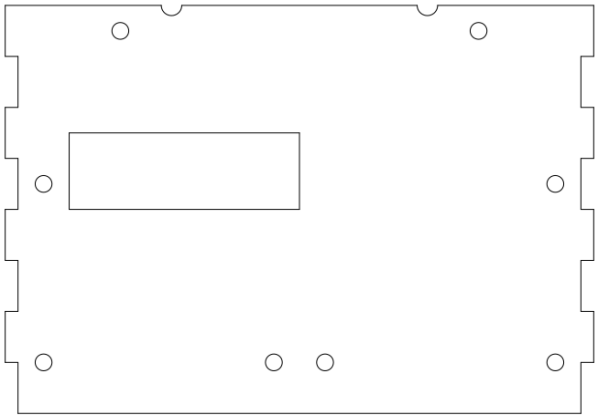


Figure 5.3: *The front wall*

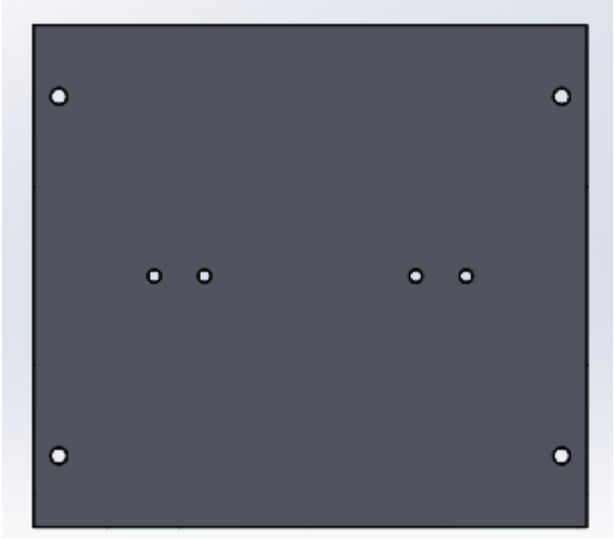


Figure 5.4: *Delrin sheet*

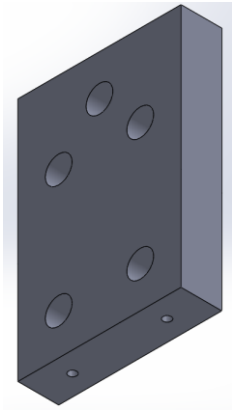


Figure 5.5: *Delrin connector for front casting wheel*

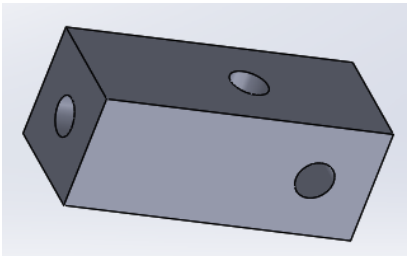


Figure 5.6: Delrin connector for three walls

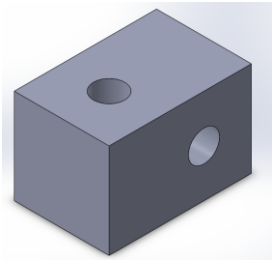


Figure 5.7: *Delrin for two walls*

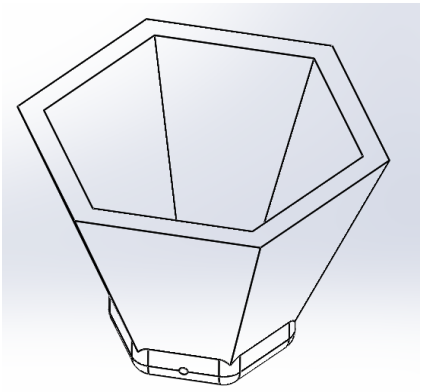


Figure 5.8: *Launcher base*

Code

```
//include servo library
#include <Servo.h>

// #DEFINE ALGORITHM PARAMETERS
//side distance threshholds
#define tooclose 6   //cm
#define toofar 14    //cm
#define diffthresh 2 //cm
#define diffcrazy 40
//front distance threshholds
#define frontthresh 20 //cm
//photo transistor threshholds
#define PTthresh 60
#define debounceArrayLength 10 //
//Servo positions
#define ServoLeft 200
#define ServoCenter 90 //experimentally determined center!1
#define ServoRight 0

// #DEFINE PINS
//switches
#define RWallFollowPin 10
#define LWallFollowPin 11
//hbridge
#define input1 5
#define input2 6
#define input3 3
#define input4 4
#define ServoPin 13
//sensors
#define sensor0 A0 // Right Rear Sharp IR
#define sensor1 A1 // Right Front Sharp IR
#define sensor2 A2 // Front Right Sharp IR
#define sensor3 A3 // Front Middle Sharp IR
#define sensor4 A4 // Front Left Sharp IR
#define sensor5 A10 // Left Front Sharp IR
#define sensor6 A9 // Left Rear Sharp IR
#define RptPin A6 // Right phototransistor
#define LptPin A7 // Left phototransistor

// #initialize global variables
Servo launcher;
int RWallFollowSwitch; int LWallFollowSwitch;

float volt0; float volt1; float volt2;
float volt3;
float volt4; float volt5; float volt6;

int distance0; int distance1; int distance2;
int distance3;
int distance4; int distance5; int distance6;

int diff;
```

```
int Rpt;
int Lpt;
int Rpt_avg;
int Lpt_avg;
int Rptdebounce[debounceArrayLength];
int Lptdebounce[debounceArrayLength];
int shotcount = 0;
bool takeshot = true;

void setup() {
  Serial.begin(9600);

  pinMode(RWallFollowPin, INPUT);
  pinMode(LWallFollowPin, INPUT);
  pinMode(input1, OUTPUT);
  pinMode(input2, OUTPUT);
  pinMode(input3, OUTPUT);
  pinMode(input4, OUTPUT);
  launcher.attach(ServoPin);
  delay(25);
  launcher.write(ServoCenter);
}

void loop() {

  ReadWorld();
  Serial.print(shotcount); Serial.print("\t");
  Serial.print(Lpt_avg); Serial.print("\t"); Serial.println(Rpt_avg);
  delay(20); //settle down

  if (shotcount > 15) {
    takeshot = false; //
  }

  if ((Rpt_avg < PTthresh && Lpt_avg < PTthresh) || not(takeshot)) {
    launcher.write(ServoCenter);
    shotcount = 0; //reset shotcount~!
  }

  if (Rpt_avg > PTthresh && takeshot) { //IR on the right
    RMotorStop();
    LMotorStop();
    launcher.write(ServoRight);
    shotcount += 1;
    delay(50);
  }
  else if (Lpt_avg > PTthresh && takeshot) {//IR on the left
    RMotorStop();
    LMotorStop();
    launcher.write(ServoLeft);
    shotcount += 1;
    delay(50);
  }
```

```
else if (RWallFollowSwitch == 0 && LWallFollowSwitch == 0) {
  RMotorStop();
  LMotorStop();
}

else if (RWallFollowSwitch == 1) {//Right wall follow switch
  if (diff > 40) {// sensors are crazy, stop doing anything TODO: parameterize diff
    RMotorStop();
    LMotorStop();
  }
  else if (SomethingInFront()) { //obstacle in front
    Serial.println("obstacle in front");
    while (diff < -1 * diffthresh || diff > diffthresh || SomethingInFront()) {
      ReadWorld();
      //rotate right
      RMotorForward();
      LMotorReverse();
    }
  }
 // NO OBSTACLES IN FRONT, IS THERE ANYTHING ON THE SIDE?
 /* else if (distance0 >= toofar || distance1 >= toofar) {
    //if there's nothing to the side either, turn sharp!
    LMotorForward();
    RMotorStop();
    delay(20);
    RMotorForward();
    } */

  // NO OBSTACLES IN FRONT, THERE'S SOME WALL DETECTED ON THE SIDE
  else { //reasonable distance difference on the right side

    //do not get too close
    if (distance1 <= tooclose) {
      Serial.println("Move away from the wall");
      RMotorForward();
      LMotorReverse();
      delay(10);
      LMotorForward();
    }
    //do not get too far
    else if (distance1 >= toofar) {
      Serial.println("Move closer to the wall");
      RMotorReverse();
      LMotorForward();
      delay(10);
      RMotorForward();
    }

    //WALL FOLLOW ON THE RIGHT
    else { //not too far, not too close, let's WALL FOLLOW
      //if the back is closer than the front, turn right a bit
      if (diff <= -1 * diffthresh) {
        Serial.println("Right");   // right turn command
        RMotorStop();
```

```
      LMotorForward();
      delay(10);
      RMotorForward();
     }
     else if (diff >= diffthresh) {
      Serial.println("Left"); //left turn command
      RMotorForward();
      LMotorStop();
      delay(10);
      LMotorForward();
     }
     else {
      Serial.println("Go"); //going straight command
      RMotorForward();
      LMotorForward();
     }
    }
   }
}

else if (LWallFollowSwitch == 1) {//Left wall follow switch
  if (diff > 40) {// sensors are crazy, stop doing anything
    RMotorStop();
    LMotorStop();
  }
  else if (SomethingInFront()) { //obstacle in front
    Serial.println("obstacle in front");
    while (diff < -1 * diffthresh || diff > diffthresh || SomethingInFront()) {
      ReadWorld();
      //rotate right
      LMotorForward();
      RMotorReverse();
    }
  }


  // NO OBSTACLES IN FRONT, IS THERE ANYTHING ON THE SIDE?
  /* else if (distance5 >= toofar || distance6 >= toofar) {
    //if there's nothing to the side either, turn sharp!
    LMotorStop();
    RMotorForward();
    delay(20);
    LMotorForward();
    } */

  // NO OBSTACLES IN FRONT, THERE'S SOME WALL DETECTED ON THE SIDE
  else { //reasonable distance difference on the left side

    //do not get too close
    if (distance5  <= tooclose) {
      Serial.println("Move away from the wall");
      LMotorForward();
      RMotorReverse();
      delay(10);
```

```
      RMotorForward();
    }
    //do not get too far
    else if (distance5 >= toofar) {
      Serial.println("Move closer to the wall");
      LMotorReverse();
      RMotorForward();
      delay(10);
      LMotorForward();
    }

    //WALL FOLLOW
    else { //not too far, not too close, let's WALL FOLLOW
      //if the back is closer than the front, turn right a bit
      if (diff <= -1 * diffthresh) {
        Serial.println("Right");   // right turn command
        RMotorStop();
        LMotorForward();
        delay(10);
        RMotorForward();
      }
      else if (diff >= diffthresh) {
        Serial.println("Left"); //left turn command
        RMotorForward();
        LMotorStop();
        delay(10);
        LMotorForward();
      }
      else {
        Serial.println("Go"); //going straight command
        RMotorForward();
        LMotorForward();
      }
    }
   }
  }
}
// HELPER FUNCTIONS

void ReadWorld() {
  RWallFollowSwitch = digitalRead(RWallFollowPin);
  LWallFollowSwitch = digitalRead(LWallFollowPin);

  volt0 = analogRead(sensor0) * 0.0048828125; // value from sensor * (5/1024)
  volt1 = analogRead(sensor1) * 0.0048828125; // value from sensor * (5/1024)
  volt2 = analogRead(sensor2) * 0.0048828125; // value from sensor * (5/1024)
  volt3 = analogRead(sensor3) * 0.0048828125; // value from sensor * (5/1024)
  volt4 = analogRead(sensor4) * 0.0048828125; // value from sensor * (5/1024)
  volt5 = analogRead(sensor5) * 0.0048828125; // value from sensor * (5/1024)
  volt6 = analogRead(sensor6) * 0.0048828125; // value from sensor * (5/1024)

  Rpt = analogRead(RptPin);
  Lpt = analogRead(LptPin);
```

```
  Rpt_avg = moving_avg(Rpt,Rptdebounce);
  Lpt_avg = moving_avg(Lpt,Lptdebounce);

  //distance to wall in cm
  distance0 = 13 * pow(volt0, -1);
  distance1 = 13 * pow(volt1, -1);
  distance2 = 27.51 * pow(volt2, -1.18);
  distance3 = 27.51 * pow(volt3, -1.18);
  distance4 = 27.51 * pow(volt4, -1.18);
  distance5 = 13 * pow(volt5, -1);
  distance6 = 13 * pow(volt6, -1);

  if (RWallFollowSwitch == 1) {
    diff = distance0 - distance1;
  }
  else if (LWallFollowSwitch == 1) {
    diff = distance5 - distance6;
  }
  else {
    diff = 0;
  }
}

int moving_avg(int *pt, int *debounceArray) { //multi-use function for Left and Right side
  int sum = 0;
  int i;
  for (i = 0; i < debounceArrayLength-1; i++) {
    debounceArray[i] = debounceArray[i+1];
    sum += debounceArray[i];
  }
  debounceArray[debounceArrayLength-1] = pt;
  sum += pt;

  return sum/debounceArrayLength;
}

bool SomethingInFront() {
  if (distance2 < frontthresh || distance3 < frontthresh || distance4 < frontthresh) {
    return true;
  }
  else {
    return false;
  }
}

void RMotorReverse() {
  digitalWrite(input1, HIGH);
  digitalWrite(input2, LOW);
}

void RMotorForward() {
  digitalWrite(input1, LOW);
  digitalWrite(input2, HIGH);
}
```

```
void RMotorStop() {
  digitalWrite(input1, LOW);
  digitalWrite(input2, LOW);
}

void LMotorReverse() {
  digitalWrite(input3, HIGH);
  digitalWrite(input4, LOW);
}

void LMotorForward() {
  digitalWrite(input3, LOW);
  digitalWrite(input4, HIGH);
}

void LMotorStop() {
  digitalWrite(input3, LOW);
  digitalWrite(input4, LOW);
}
```