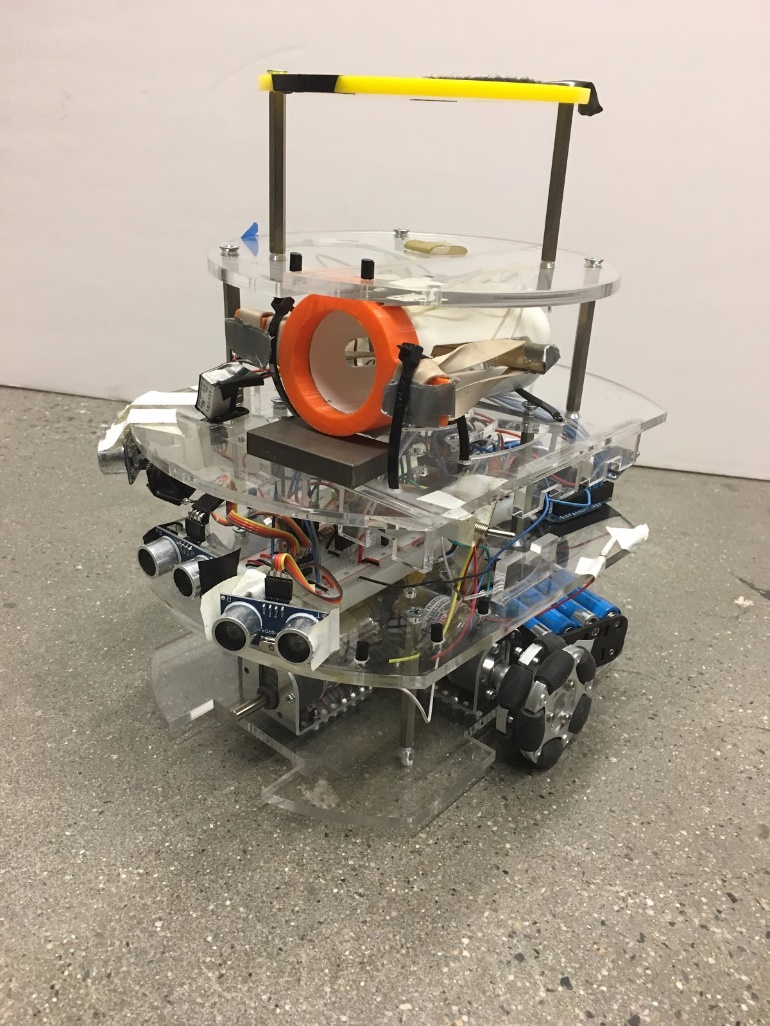
BumbleBee

Darwin Huang, Jessie Wu, Tony Zhao

Professor Mar

ME412: Autonomous Mobile Robots



Contents

[1. Abstract 3](#_Toc470112293)

[2. Introduction 4](#_Toc470112294)

[3. Strategy and Design Overview 5](#_Toc470112295)

[4. Implementation and Fabrication 8](#_Toc470112296)

[5. Electronics 10](#_Toc470112297)

[5.1 Arduino Mega 10](#_Toc470112298)

[5.2 Voltage Regulator 10](#_Toc470112299)

[5.3 Ultrasonic Range Finder 10](#_Toc470112300)

[5.4 Infrared Radiation Sensor 10](#_Toc470112301)

[5.6 H-Bridge 10](#_Toc470112302)

[5.7 Motors 10](#_Toc470112303)

[5.8 Motors (Launcher) 11](#_Toc470112304)

[6. Discussion 12](#_Toc470112305)

[7. Modifications 13](#_Toc470112306)

[7.1 Strategy 13](#_Toc470112307)

[7.2 Drivetrain 13](#_Toc470112308)

[7.3 Sensors 13](#_Toc470112309)

[7.4 Layer 5 15](#_Toc470112310)

[7.5 Code 15](#_Toc470112311)

[8. Conclusion 17](#_Toc470112312)

[Appendix A: Cost Report 18](#_Toc470112313)

[Appendix B: CAD Files 19](#_Toc470112314)

[Appendix C: Code 24](#_Toc470112315)

# 1. Abstract

Bumblebee is an omni directional Autorobot consisting of four layers designed to participate in The Cooper Union Annual Autonomous Mobile Robots Tank Battle. Due to the omni directional wheels, Bumblebee excels in roaming about the arena along an x-y coordinate axis. Ideally, Bumblebee will only move forward, backward, left, and right, not making any turns. This will let it easily navigate the arena and only turn when it detects another robot by picking up its IR beacon. Bumblebee will turn to the opposing robot and fire a ping-pong ball at it, eliminating the Decepticon. However, the robot was modified to a three wheel drive that relied on the strategy of hugging the wall on it’s right side to navigate the arena.

# 2. Introduction

The Tank Battle Robot Competition is an annual robotics competition held by The Cooper Union Autonomous Mobile Robots class. The competition is a one on one battle between two opposing robots. The objective of the competition is to score as many points as possible by shooting a 38mm ping pong ball at the opposing robot or the opposing home base. The robot must traverse the arena from its own home base to find the opposing robot or the opposing robot’s base. The arena is 6 feet wide and 10 feet long with a 1 foot tall wall height. The home base section of the wall will be recessed 0.25 inch to 1 inch deep.

Each robot is required to fit within the dimensions of 12” x 12” x 10”. There is no weight limit to the robot and no cost limit. The only requirement of the robot is that there needs to be a platform for the IR beacon. The IR beacon is necessary for the robots to track one another. The cannon may also only shoot one ping pong ball at a time. After shooting the ball, it must return to base to reload. The arena that the competition will be held in a 6’ x 10’ box with walls that are 1’ tall. The home base of the wall will be recessed 0.25” to 1” deep.

# 3. Strategy and Design Overview

The goal of Bumblebee’s robot design is to reach the opposing base as fast as possible and accrue points by base hits. In order to simplify how the robot traverses the field, Bumblebee was given omni directional wheels. As shown in Figure 1, a pair of wheels would be responsible for front and back movement. A pair of wheels would be responsible for lateral movement. Ideally, all movement of the robot will be in the same XY coordinate plane. It will move forward and back or strafe left and right.

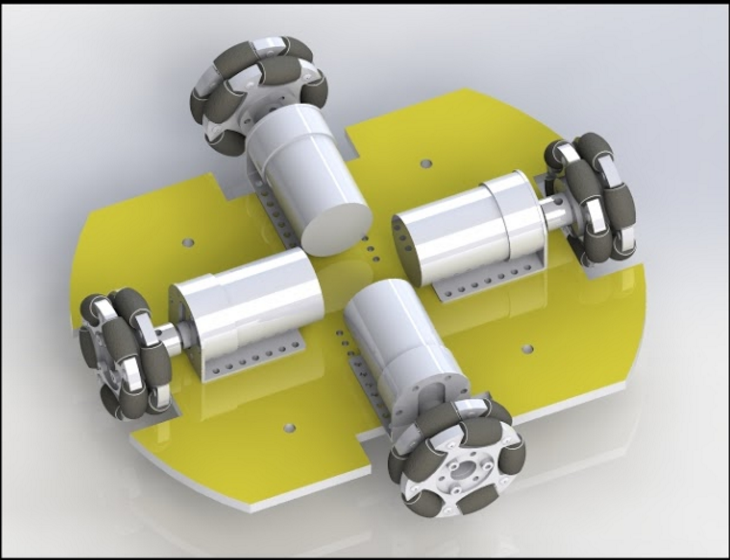


Figure 1: A rendering of Bumblebee’s omni directional drive train

This design allows for simpler code in robot maneuvering. The 2nd layer of Bumblebee has IR sensors and Range Finder sensors for the robot to “see”. The only time Bumblebee would perform a turn would be when it detects the IR beacon of an opposing robot. Bumblebee would fire the ping pong ball, using the canon mounted on the 3rd layer, at the robot once the center IR lines up with the opposing robot and then return to facing forward. The compass on Bumblebee will allow it to know it’s orientation and return to facing forward when necessary. If the range finders detect a wall or obstacle, Bumblebee will strafe away from that obstacle.

Bumblebee is by no means a small robot, because typical autobots are very large. However Bumblebee is thinner in its frontal profile than its side profile. Figure 2 shows the aspect ratio of Bumblebee.

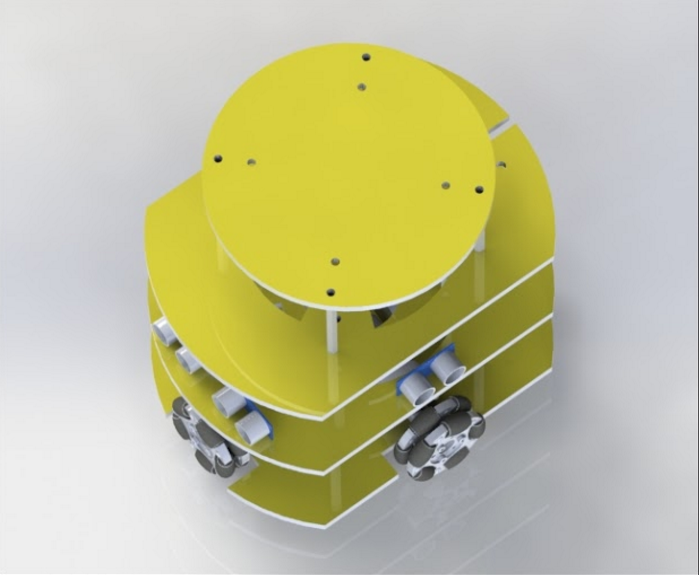


Figure 2: Overhead view showing Bumblebee is thinner from the front

This makes Bumblebee appear thinner when facing robots directly in front of it, giving a smaller target for opposing robots to hit.

The IR sensors are arranged in a circular array along the front of Bumblebee (see Figure 3). Each IR sensor has a 30 degree of vision. By placing the in this formation, there is not an excessive amount overlap in vision and it will ensure that Bumblebee will be able to see any IR beacon that is in front of it.

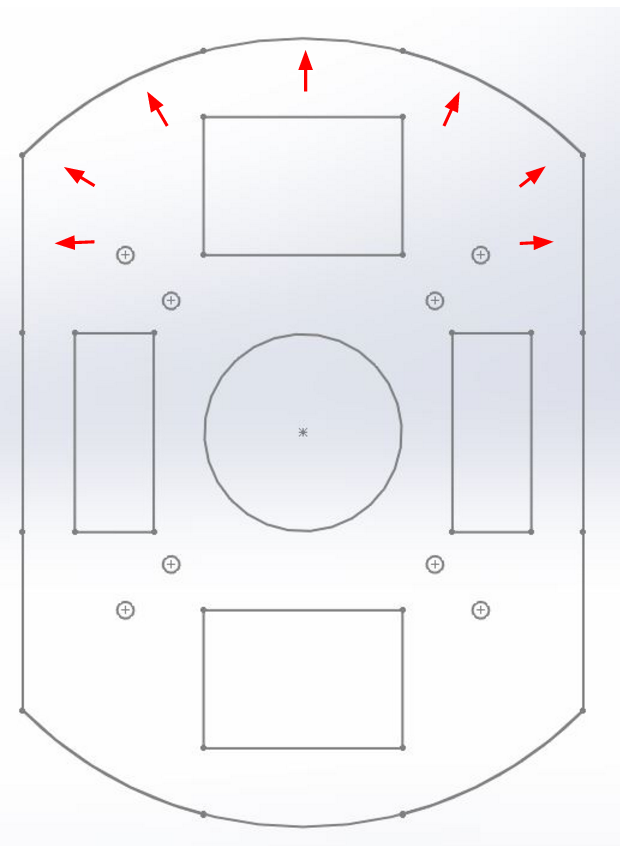


Figure 3. IR sensor layout on the 2nd layer

The Range Finder sensors are placed in cross formation. Two IR are placed in the front to ensure there is no frontal collision. Since Bumblebee is only supposed to move forward, backwards, and sideways, the other two Range Finder sensors would go on the side of Bumblebee. One facing the left side and one facing the right side as shown in Figure 4.

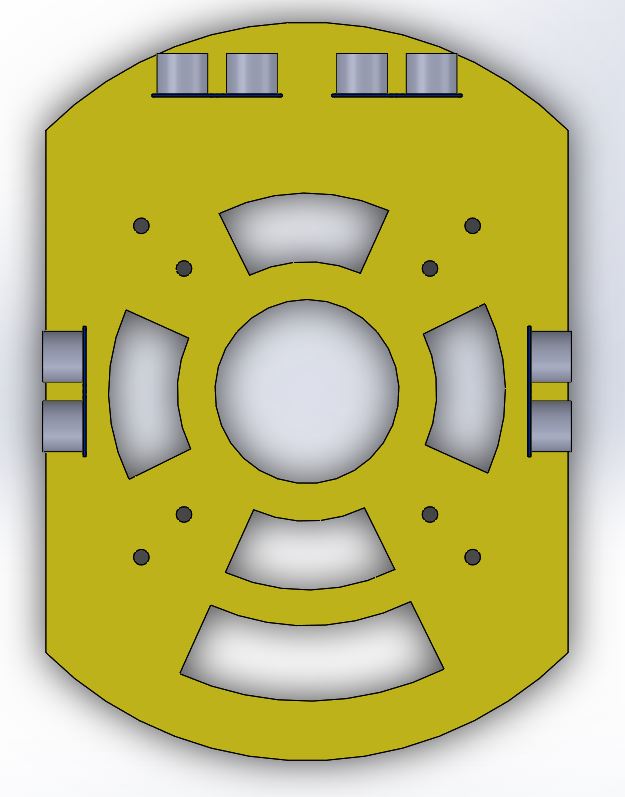
**

Figure 4. Ultrasonic Range Finder layout on 2nd layer

# 4. Implementation and Fabrication

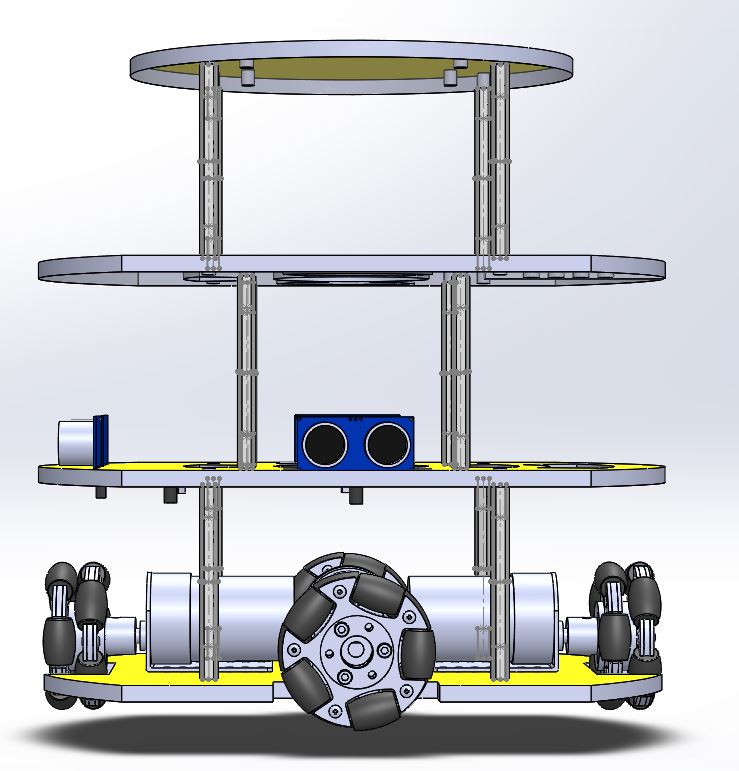


Figure 5. Side view of Bumblebee to illustrate the four layers in design.

The chassis consists of a laser cut body made of acrylic. The material of choice is picked for its ease of manufacturing. Each of the 4 layers of Bumblebee is cut from ¼” acrylic. The first level houses the drive train. The second layer houses the sensors and electronics. The third layer houses the ping pong cannon and the trigger mechanism. The fourth layer is to fulfill the competition requirement for an IR beacon.

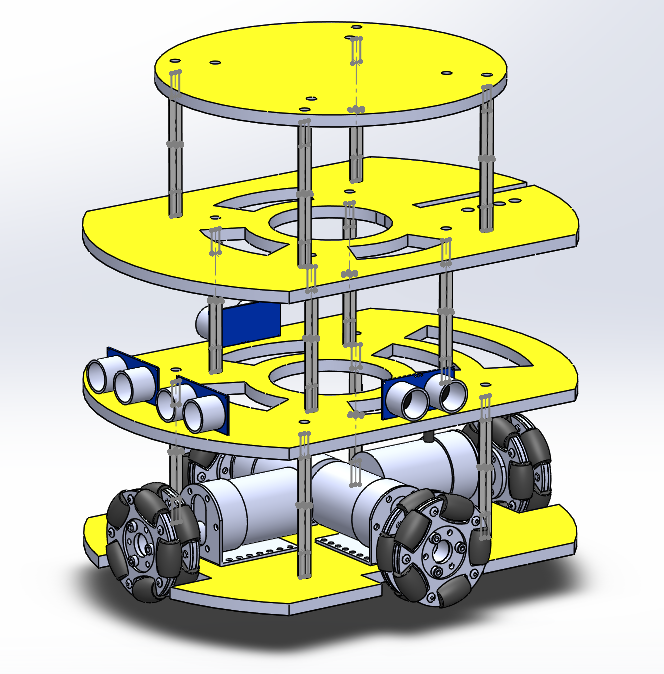
**

Figure 6. An isometric view of Bumblebee illustrating the layer design

Each of the layers are connected by four 2” long standoffs. Each layer has designs cut out in order to reduce overall robot weight and for spaces to run wires through. The standoffs are 2” long and can be secured to the plates by using quarter inch size screws. Note, an additional 5th layer was added in order to maximize the height of the IR beacons, which would make it harder for other robots to detect Bumblebee.

The canon design is a simple sling shot system. The rubber band is held back storing potential energy. When the rubber band is released, all the stored potential energy is converted to kinetic energy and will hit the ping pong ball launching it forward. The rubber band is held by a 3D printed part that will be referred to as the all spark. The allspark is designed to be a cap that is press fit to one end of the PVC tube or canon. It contains a notch on each side in order to hold the rubber band. A string pulls the wooden dowel that the rubber bands are connected to back. The end of the string is hooked to a motor attachment, as shown in Figure 7. The string is released, or slips off, when the motor rotates the attachment 90 degrees.

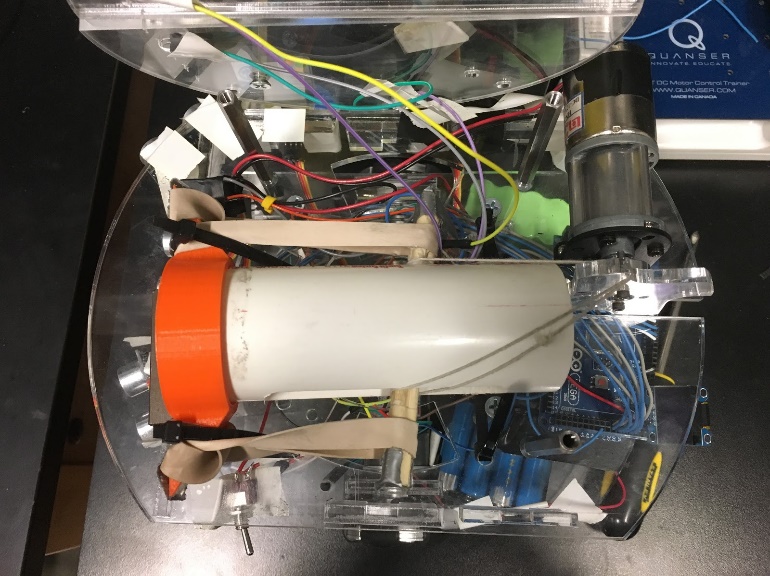
~~~~

Figure 7.  The loaded ping pong ball launcher. The string is pulled back and over the motor attachment indicating it is armed.

# 5. Electronics

## 5.1 Arduino Mega

The Arduino Mega is a very powerful microcontroller that takes 9 inputs, the 5 IR sensors and 4 Range Finders, and outputs 8 signals to the H-bridges. In addition it also outputs a signal to the motor that controls the cannon trigger.

## 5.2 Voltage Regulator

The L7805CV has three pins and is used to limit the output voltage to be 5V. The 12V battery supply is connected to the input pin of the L7805CV. The middle pin is connected to the ground rail. The output pin will output the constant 5V. The output is connected to the power rail to serve as logic supply voltage for all other circuit parts. A connection is made to the 12V line in parallel in order to supply the motors with the 12V power.

## 5.3 Ultrasonic Range Finder

The HC-SR04 Range Finder sensor works on a similar principle of sonar echolocation. The sensor emits an ultrasonic signal(trig) from one of its ports and receiving the signal after it bounces back(echo). It uses the time between when the signal was sent and when it was received to determine its distance from any wall or obstacle. The HC-SR04 sensor is good for ranges from 2cm to 4m.

## 5.4 Infrared Radiation Sensor

The LTR-516 AD IR sensor used is for detecting the wavelength of around 900 nm. It has an approximate 30 degree of vision directly in front of it.  The resistance across the IR sensor changes depending on its distance of the IR signal. The change in current is used to determine the distance of the sensor to the signal. An array of these IR sensors are used to find the direction of the IR signal.

## 5.6 H-Bridge

The L298N H-bridge takes in two inputs from the microcontroller and returns two outputs to each motor. There are two sets of inputs and outputs from the H-bridge, one set for each motor. The inputs to the H-bridge are 5V or 0V signalling a (1) or a (0). When the inputs to the H-bridge are (0)-(1) or (1)-(0), then H-bridge outputs the (1), or an operating voltage,  to make the motor spin. If the inputs are (1)-(1) or (0)-(0), then the output to the motor will be (0), and the motor will not spin.

## 5.7 Motors

Bumblebee uses two pairs of gear head motors that run on 12V with a 50:1 gear ratio. The motors are connected to the H-bridge. If the input to the motors is (1) - (0), the motors will spin forward, and if the inputs are (0) - (1), then the motors will spin in reverse. If the signal to the two pins on each motor are the same, (0) - (0) or (1) - (1), then the motors will not move.

5.8 Motors (Launcher)

Bumblebee uses one gear head motor that runs on 12V with a 410:1gear ratio. The motor is connected via a double pole double throw relay as opposed to an H-bridge, due to wanting the motor to be connected without using more breadboard space, or using a perforated board. Additionally, the only real need for this motor was to allow a control signal to turn the motor in one direction, so any additional features of the H-bridge wouldn’t have been too helpful. Basically, wiring the relay as a single pole double throw relay was the simplest circuit, that could use minimal space on the breadboard.

# 6. Discussion

Throughout the semester, Bumblebee was able to complete the IR tracking and Obstacle avoidance requirements using omni drive. Using an array of IR sensors on the same level, Bumblebee would turn to face the IR beacon and follow it. Using the ultrasonic range finders, Bumblebee would back up from the wall and avoid the obstacles that were placed in front of the sensors. There were no issues getting the sensors to work along with the drive train.

Bumblebee’s design was very straightforward and had a simple drive control. Unfortunately, the design did not take into account the uneven floors of the classrooms. Due to the use of 4 omni wheels, 4 points of contact are required for Bumblebee to move. The additional constraints added from using a 4 wheel drive prevented Bumblebee from strafing sideways effectively if the floor was uneven and a single wheel would not be in contact with the ground.

A potential fix for this issue is to implement a suspension system that will push the wheels down no matter the curvature of the ground, ensuring all points of contact necessary for driving. Unfortunately due to the late discovery of this problem, a suspension system could not be designed and built in time. Bumblebee’s designed was modified to a 3 wheel drive robot that would turn, rather than strafe left and right.

# 7. Modifications

Upon realizing the faults in Bumblebee’s 4 wheel drive design, a series of changes were implemented to ensure that Bumblebee would be competition ready.

## 7.1 Strategy

Due to the inability of Bumblebee to consistently strafe left and right, the strategy changed to edge hugging. Edge hugging is the most consistent method for a robot to navigate the arena. Bumblebee would start off facing the right, with the right side of the robot already against the wall. He would then stay close to that wall and turn back towards the wall when it detects itself moving too far away from the wall.

## 7.2 Drivetrain

Having four wheels prevented Bumblebee from going over some obstacles because of the wheel dimension. In addition there was no need for four wheels now that the robot won’t be using its omnidrive capability. The front wheel was removed from the robot, leaving the two side wheels to be the main driving force. The third wheel in the back served as a rudder to help the robot turn faster. The three wheel drive also allows Bumblebee to traverse over obstacles much more easily than he did with four wheels.

## 7.3 Sensors

Since Bumblebee would be edge hugging on the right side, there was no longer a need for a Range Finder to be on the left of the robot. Instead, two Ultrasonic Range Finders were placed on the right side of the robot, and one facing directly forward. Two more rangefinders were placed in the front of the robot. One is angled to the right, to cover the blind spot between the front facing sensor and the right facing sensors. The other is angled slightly to the left to prevent our robot from accidentally bumping into and obstacles on our left while turning. It was to ensure Bumblebee was not completely blind one the left side.

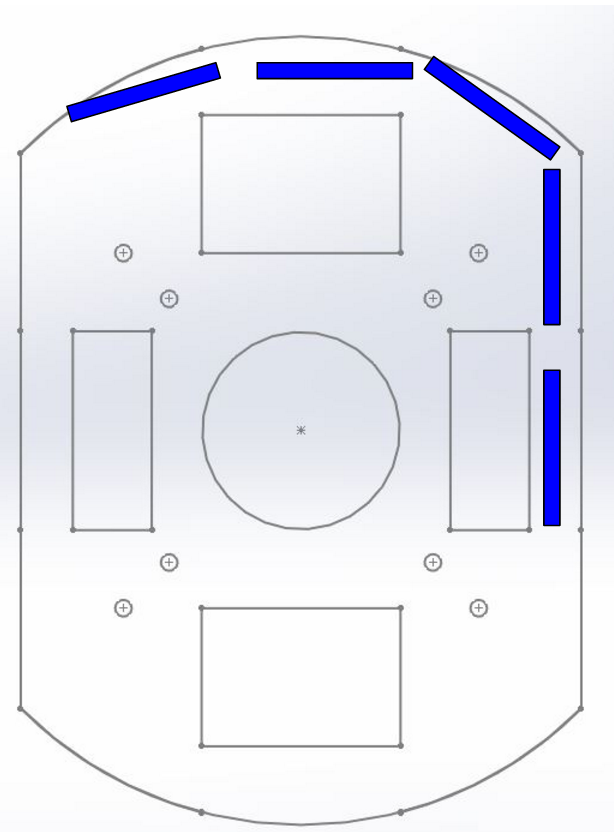


Figure 9. New Ultrasonic Range Finder configuration on Bumblebee

The IR sensors on the second layer of Bumblebee remained unchanged. Bumblebee would still use that array of sensors to turn until the front most is facing the IR beacon that it detects. But in addition to the lower array of IR sensors, two sensors were added on a fourth layer. There was a possibility that opposing robots would try to mount the competition IR beacon as tall as possible on the robot. Bumblebee needed to ensure he could see the IR beacons at all heights. However, since only two IR sensors were used on the upper layer, tracking could not really be done. The upper IRs were only used to trigger the ping pong ball launcher when it sensed an IR signal directly in front of it.

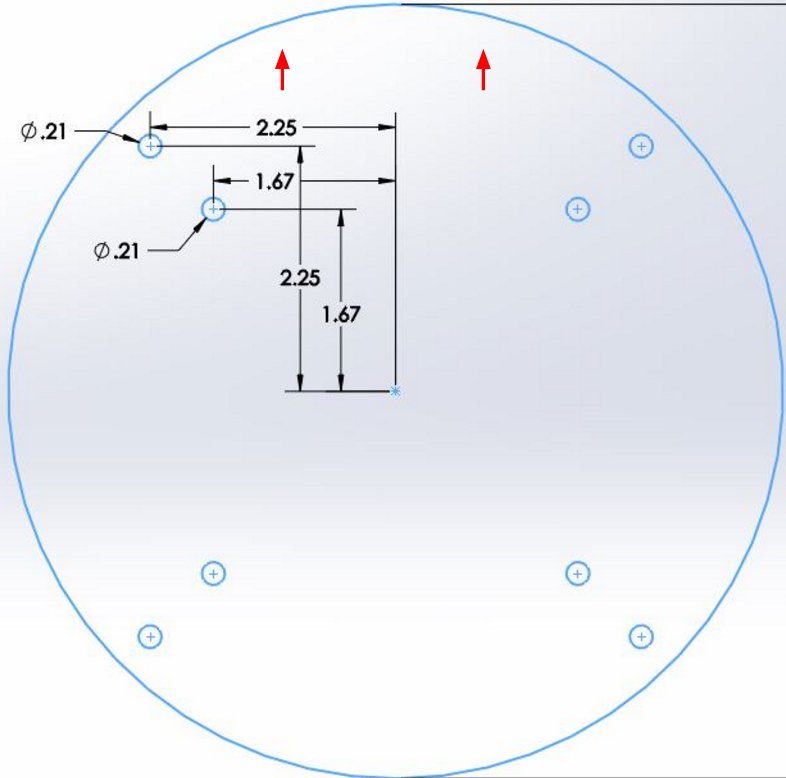


Figure 10. IR sensor layout on Layer 4

## 7.4 Layer 5

A fifth layer was added to Bumblebee in attempts to make the placement of the IR beacon as tall as possible. Robots who mounted their IR sensors too low will not be able to detect the IR beacon on the fifth layer of Bumblebee.

## 7.5 Code

Flow Diagram**:**

**Original StrategyC:\Users\huang\Downloads\strat1.png**

**Revised Strategy**

C:\Users\huang\Downloads\strat2 (1).png

Revised Code

The code, as included in Appendix C, was written for use on an Arduino Mega, with debugging output values written to an external OLED display, and is written with a wall-hugging strategy as shown above. There are several main portions of the code: definitions, setup, loop, ultrasonic-based functions, IR-based/tracking functions, motor functions, movement functions, and OLED display functions. The definitions portion contains thresholds and timings that were tweaked during testing to optimize performance. The setup portion set up the display and all relevant sensors. The loop portion contained the overall logic of the code, which was abstracted through use of functions. The ultrasonic-based functions contained functions for determining whether objects were too close on various axis, through use of the ultrasonic sensors. The IR-based/tracking functions included functions to determining IR values, interpreting them, and using them to track the enemy to shoot. The motor part of the code had functions to move the robot in various directions by writing to output pins that controlled the motor. The movement part contained functions that would move the motor in specific directions, and was mostly contained for abstraction purposes. The OLED display portion contained mostly functions for use to display values and debugging output on the OLED display, so that the robot could be tested without dependency on a USB wire to a computer.

# 8. Conclusion

Even though the design and strategy of Bumblebee was changed a week before the competition, he still won the competition. Bumblebee could consistently hug the right wall and avoid obstacles. It would hug the wall until it detects the opposing base, fire a shot, and continue to hug the wall until it returned to the home dock. In one five-minute round of the competition, Bumblebee made three runs around the arena and fired three shots at the opposing base.

During the first round, Bumblebee completed three out of the four achievements required during the competition. He successfully achieved “returned home”, “no collision”, and “opponent base hit”. In fact, Bumblebee did not collide with any obstacles throughout the entire duration of the competition. Unfortunately, Bumblebee did not complete the final achievement of “opponent robot hit”. This was due to IR sensors mounted on him not being in ideal positions to detect the location of the beacon on opposing robots and forgetting to arm him when he did detect an opposing robot.

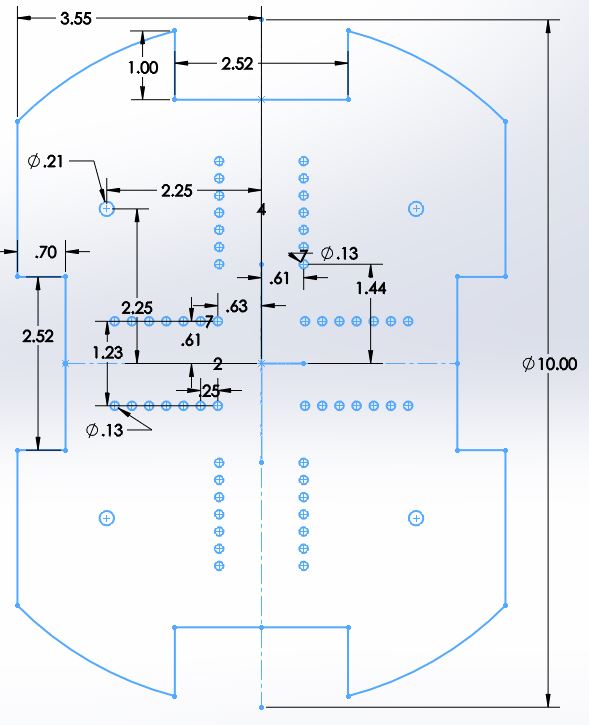
Overall Bumblebee completed all 4 rounds of competition first with a record of 2 wins, 1 tie, and 1 loss. This record was tied with another team, but Bumblebee won the tiebreaker by having more total points throughout the competition.

# Appendix A: Cost Report

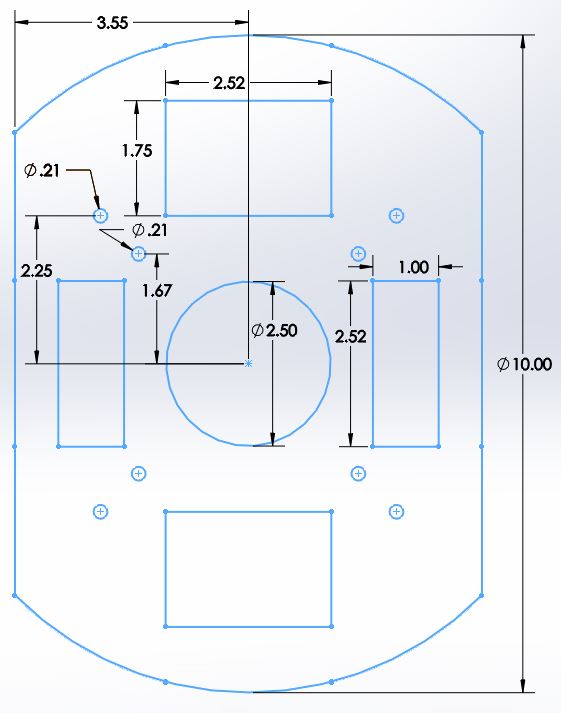
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Quantity | Part Name | Common Name | Unit Price | Total Price |
| 1 | Adafruit HMC5883L | Compass | 10 | 10 |
| 4 | iTead HC-SR04 | Range Finder | 3 | 12 |
| 5 | Lite-On LTR-516AD | IR photodiode | 1 | 5 |
| 4 | GHM-13 | Motors | 30 | 120 |
| 4 | RB-Nex-75 | Omni Wheels | 16 | 64 |
| 4 | RB-Nex-80 | Mounting Hub for Omni Wheel | 7 | 28 |
| 2 | RB-Pol-03 | Motor Mounting Bracket (pair) | 8 | 16 |
| 1 | ATMega2560 | Arduino Mega | 46 | 46 |
| 4 | L298N | H-Bridge Motor Driver | 3 | 12 |
| 1 | 1512 | Lock-Style Solenoid | 15 | 15 |
| 1 | N/A | Rubber Band | 11 | 11 |
| 1 | BAT-01 | Power Supply | 30 | 30 |
| 1 | P112FGP-WH-5 | 1.5" PVC Pipe | 9 | 9 |
| 1 | N/A | Clear Acrylic Sheet 18"x 32" | 27 | 27 |
| 12 | 91115A823 | McMaster Threaded Hex Standoff | 2.72 | 32.64 |
| 2 | 12BH348/CS-GR | 4 AA Battery Enclosure | 2.08 | 4.16 |
| 1 | GHM-03 | Motors | 40 | 40 |
| 1 | MSP430 | OLED Display | 10 | 10 |
|  |  | TOTAL PRICE |  | 491.80 |

# Appendix B: CAD Files

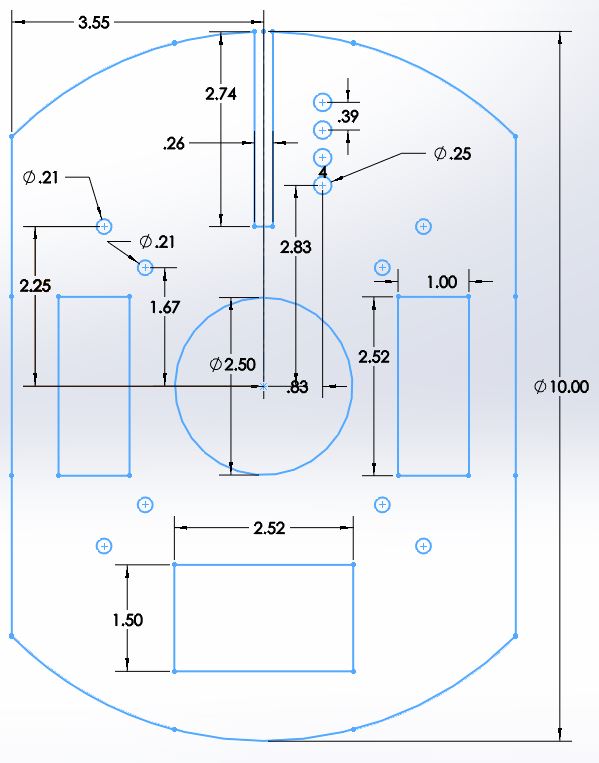
**Note: all CAD Files show front of robot downwards.**



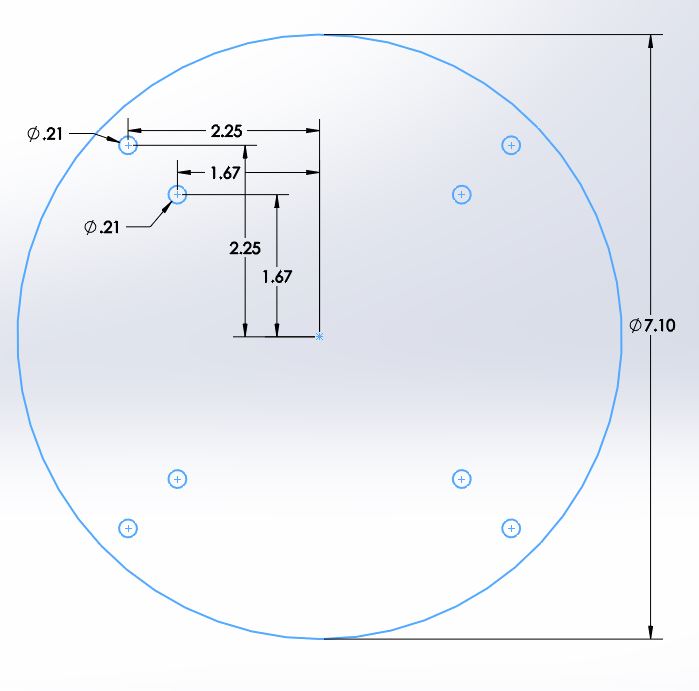
Appendix Figure 1. Schematic of Layer 1



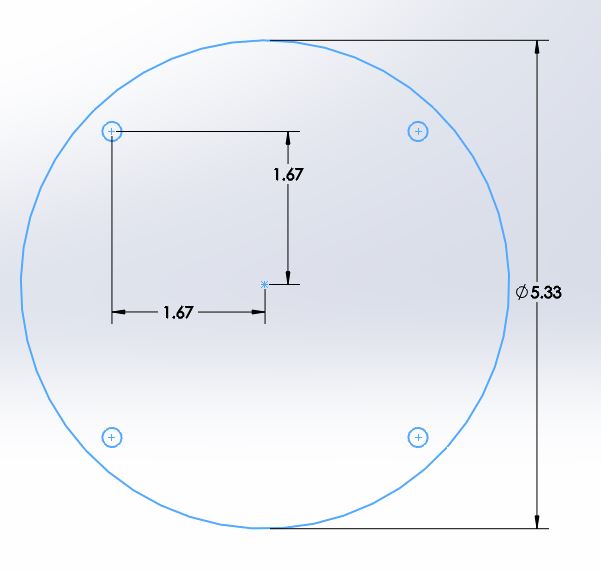
Appendix Figure 2. Schematic of Layer 2



Appendix Figure 3. Schematic of Layer 3



Appendix Figure 4. Schematic of Layer 4



Appendix Figure 5. Schematic of Layer 5

# Appendix C: Code

/\*

\* NOTE: Libraries can be found at:

\* Adafruit\_GFX.h       https://github.com/adafruit/Adafruit-GFX-Library

\* Adafruit\_SSD1306.h   https://github.com/adafruit/Adafruit\_SSD1306

\* Ultrasonic.h         https://www.itead.cc/wiki/Ultrasonic\_Ranging\_Module\_HC-SR04

\*/

#include "Ultrasonic.h"

//libraries for OLED display

#include <SPI.h>

#include <Wire.h>

#include <limits.h>

#include "Adafruit\_GFX.h"

#include "Adafruit\_SSD1306.h"

//initializations for OLED display

#define OLED\_RESET 4

Adafruit\_SSD1306 display(OLED\_RESET);

//launcher pin

#define LAUNCHERPIN          39  // pin with launcher

#define IRUPPIN1                      7

#define IRUPPIN2                      8

//initializations for ultrasonics

//TODO: MAKE SURE THESE PINS ARE CORRECT

Ultrasonic ultrasonic4(26,27);  // right front

Ultrasonic ultrasonic3(28,29);  // front left

Ultrasonic ultrasonic2(30,31);  // front mid

Ultrasonic ultrasonic1(32,33);  // front right

Ultrasonic ultrasonic5(34,35);  // right back (was left)

//direction indicators

#define MOTOR\_FORWARD            0

#define MOTOR\_TURN\_RIGHT      1

#define MOTOR\_TURN\_LEFT          2

#define MOTOR\_BRAKE                   3

// USER INPUT

//Thresholds to tweak

#define ULTRASONIC\_RIGHT\_LOW\_THRESH   9

#define ULTRASONIC\_RIGHT\_HIGH\_THRESH  11

#define ULTRASONIC\_FRONT\_LOW\_THRESH   13

#define IR\_LOW\_THRESHOLD              38

#define IR\_LOW\_DELTA\_THRESHOLD\_UP     20

#define MOTOR\_LEFT\_ADJUST\_TIME      200

#define MOTOR\_RIGHT\_ADJUST\_TIME     100

#define MOTOR\_LEFT\_90\_TIME          500  // What delay time will generate a ~90 degree turn?

#define LAUNCHERSPINTIME            1500   // experimentally determined to provide ~45 degrees spin

//Thresholds added after flowchart was created

#define MOTOR\_INCH\_FWD\_TIME         50

#define MOTOR\_TRACK\_TURN\_TIME       200

//Global Variables

int currDir = 0;      // 0 indicates forward, 1 indicates turning right, 2 indicates turning left, 3 indicates stopped (ex. launching ball)

int IRPins[7];        // 7 IRpins, leftmost is 0, rightmost is 6

//startup values

bool hasBall = true;  // true indicates going away from home, false indicates returning home

int init\_up\_1\_val;

int init\_up\_2\_val;

void setup() {

 Serial.begin(19200);          // Arduino Mega valid baud rates: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, 115200

 setup\_display();

 pinMode(LAUNCHERPIN, OUTPUT);

 // 5 - 13 are motor pins

 for(int i = 5; i < 13; i++) {

   pinMode(i, OUTPUT);

 }

 init\_up\_IR();

}

// ACTIVE CODE

void loop() {

 if(hasBall){

   if(foundIRTarget()){

     trackIRAndShoot();

   }

   if(foundUpIRTarget()){

     brake();

     shoot();

   }

 }

 //check Ultrasonics

 //check Right

 if(rightTooClose()){  //too close

   smallTurnLeft();

//    moveBackwards();

//    delay(200);

 }

 else if(rightTooFar()){ //too far

   smallTurnRight();

 }

 //check Front

 if(frontTooClose()){  //too close

   turn90Left();

   inchForward();

 }

 else{ //ok

   inchForward();

 }

 refresh\_display();

}

/\*\*\*\*\*\*\*\*\*\*

ULTRASONIC SENSORS

\*\*\*\*\*\*\*\*\*\*/

bool USBetween(Ultrasonic us, int lowThresh, int highThresh){

//  refresh\_display();

 int read = us.Ranging(CM);

 if(read < highThresh && read > lowThresh){

   return true;

 }

 return false;

}

bool frontTooClose(){

 int lowThresh = INT\_MIN;

 int highThresh = ULTRASONIC\_FRONT\_LOW\_THRESH;

 return USBetween(ultrasonic1, lowThresh, highThresh) || USBetween(ultrasonic2, lowThresh, highThresh); // TODO: fix ultrasonic 3 and put the check for it back

}

bool rightTooFar(){

 int lowThresh = ULTRASONIC\_RIGHT\_HIGH\_THRESH;

 int highThresh = INT\_MAX;

 return USBetween(ultrasonic3, lowThresh, highThresh) || USBetween(ultrasonic4, lowThresh, highThresh)|| USBetween(ultrasonic5, lowThresh, highThresh);

}

bool rightTooClose(){

 int lowThresh = INT\_MIN;

 int highThresh = ULTRASONIC\_RIGHT\_LOW\_THRESH;

 return USBetween(ultrasonic3, lowThresh, highThresh) || USBetween(ultrasonic4, lowThresh, highThresh);

}

/\*\*\*\*\*\*\*\*\*\*

IR SENSORS & LAUNCHER

\*\*\*\*\*\*\*\*\*\*/

// function to run through analog pins; returns IR with max value if past threshold

int getMaxIR(){

//  refresh\_display();

 int maxPin = 0;

 int maxIndex = 0;

 for (int j = 0; j < 7; j++) {

   int dummy = 0;

   for (int i = 0; i < 3; i++) {

     dummy = dummy + analogRead(j);

   }

   dummy = dummy / 3;

   IRPins[j] = dummy;

   if(IRPins[j] > maxPin) {

     maxPin = IRPins[j];

     maxIndex = j;

   }

 }

 if(maxPin > IR\_LOW\_THRESHOLD) {

   return maxIndex;

 }

 return -1;

}

int init\_up\_IR(){

 int r1 = 0;

 int r2 = 0;

 int trials = 10;

 for(int i = 0; i < trials; i++){

     r1 += analogRead(IRUPPIN1);

     r2 += analogRead(IRUPPIN2);

     delay(10);

 }

 init\_up\_1\_val = r1/trials;

 init\_up\_2\_val = r2/trials;

}

bool foundUpIRTarget(){

 int read1 = analogRead(IRUPPIN1);

 int read2 = analogRead(IRUPPIN2);

 return ((abs(read1 - init\_up\_1\_val) > IR\_LOW\_DELTA\_THRESHOLD\_UP) || (abs(read2 - init\_up\_2\_val) > IR\_LOW\_DELTA\_THRESHOLD\_UP));

}

//whether the target has been seen

bool foundIRTarget(){

 if(getMaxIR() != -1){

   return 1;

 }

 return 0;

}

// function tracks IR signal; stops when it is centered

int trackIRAndShoot(){

 brake();

 int stime, etime, toRight, maxIR, timeMovingRight; //to undo time

 while(1){

//    brake();

//    delay(100);

//    refresh\_display();

   maxIR = getMaxIR();

   if(maxIR < 0){    //fuck it

     brake();

     break;

   }

   else if(maxIR == 4) {//brake and shoot

     brake();

     shoot();

     break;

   }

   stime = millis();

   if(maxIR < 4){

     toRight = 1;

     turnRight();

     delay(MOTOR\_TRACK\_TURN\_TIME);

   }

   else if(maxIR > 4) {

     toRight = -1;

     turnLeft();

     delay(MOTOR\_TRACK\_TURN\_TIME);

   }

   brake();

   etime = millis();

   timeMovingRight += toRight \* (etime - stime);

 }

 brake();

//  undoTrackingRotate(timeMovingRight);

 return 0;

}

//resets angle of attack to that ideally before trackIRAndShoot.

//error acceptable due to movement strategy tolerating it.

void undoTrackingRotate(int milliseconds){

 if(milliseconds > 0){

   turnLeft();

   delay(milliseconds);

 }

 else{

   turnRight();

   delay(milliseconds);

 }

brake();

}

// after IR is detected, shoot a gun, turn, and go home

void shoot() {

 refresh\_display();

 digitalWrite(LAUNCHERPIN, HIGH);

 delay(LAUNCHERSPINTIME);

 digitalWrite(LAUNCHERPIN, LOW);

 hasBall = false;

}

/\*\*\*\*\*\*\*\*\*\*

MOTORS

\*\*\*\*\*\*\*\*\*\*/

void brake()  {

 digitalWrite(7, LOW);

 digitalWrite(8, LOW);

 digitalWrite(9, LOW);

 digitalWrite(10, LOW);

 digitalWrite(11, LOW);

 digitalWrite(12, LOW);

//  currDir = MOTOR\_BRAKE;

//  refresh\_display();

}

void moveForward() {

 digitalWrite(7, LOW);

 digitalWrite(8, HIGH);

 digitalWrite(9, LOW);

 digitalWrite(10, HIGH);

 digitalWrite(11, LOW);

 digitalWrite(12, LOW);

 currDir = MOTOR\_FORWARD;

//  refresh\_display();

}

//moveBackwards is not used, so this can safely be deleted

//however, this function is kept for bookkeeping purposes.

//in case of using moveBackwards, please add a new #define MOTOR\_BACKWARD

void moveBackwards() {

 digitalWrite(7, HIGH);

 digitalWrite(8, LOW);

 digitalWrite(9, HIGH);

 digitalWrite(10, LOW);

 digitalWrite(11, LOW);

 digitalWrite(12, LOW);

 //currDir = MOTOR\_BACKWARD;

//  refresh\_display();

}

void turnRight(){

 digitalWrite(7, LOW);

 digitalWrite(8, HIGH);

 digitalWrite(9, HIGH);

 digitalWrite(10, LOW);

 digitalWrite(11, LOW);

 digitalWrite(12, HIGH);

 currDir = MOTOR\_TURN\_RIGHT;

//  refresh\_display();

}

void turnRight2(int x){

 digitalWrite(7, LOW);

 digitalWrite(8, HIGH);

 digitalWrite(9, HIGH);

 digitalWrite(10, LOW);

 digitalWrite(11, LOW);

 digitalWrite(12, HIGH);

 delayMicroseconds(x);

//  currDir = MOTOR\_TURN\_RIGHT;

//  refresh\_display();

}

void turnLeft(){

 digitalWrite(7, HIGH);

 digitalWrite(8, LOW);

 digitalWrite(9, LOW);

 digitalWrite(10, HIGH);

 digitalWrite(11, HIGH);

 digitalWrite(12, LOW);

 currDir = MOTOR\_TURN\_LEFT;

//  refresh\_display();

}

/\*\*\*\*\*\*\*\*\*\*

MOVEMENT (using motors)

\*\*\*\*\*\*\*\*\*\*/

void smallTurnLeft(){

 turnLeft();

 delay(MOTOR\_LEFT\_ADJUST\_TIME);

}

void smallTurnRight(){

 turnRight();

 delay(MOTOR\_RIGHT\_ADJUST\_TIME);

}

void turn90Left(){

 turnLeft();

 delay(MOTOR\_LEFT\_90\_TIME);

}

void inchForward(){

 moveForward();

 delay(MOTOR\_INCH\_FWD\_TIME);

}

/\*\*\*\*\*\*\*\*\*\*

OLED DISPLAY

\*\*\*\*\*\*\*\*\*\*/

// setup for the display

void setup\_display(){

 display.begin(SSD1306\_SWITCHCAPVCC, 0x3C); // initialize oled with the I2C addr 0x3D (for the 128x64)

 // Show image buffer on the display hardware.

 // Since the buffer is intialized with an Adafruit splashscreen

 // internally, this will display the splashscreen.

 display.display();

 delay(250);

 // Clear the buffer.

 display.clearDisplay();

}

// refreshes the display

void refresh\_display(){

 //text setup

 display.clearDisplay();

 display.setTextSize(1);

 display.setTextColor(WHITE);

 display.setCursor(0,0);

 //text

 displayAllRawUltrasonicValues();

//  displayAllHIGHRawIRvalues();

 displayAllRawIRvalues();

 display.setTextSize(2);

 display.println("BUMBLEBEE");

 display.print("Dir:  ");

 if(currDir == MOTOR\_FORWARD){display.println("FWD");}

 else if(currDir == MOTOR\_TURN\_RIGHT){display.println("RGT");}

 else if(currDir == MOTOR\_TURN\_LEFT){display.println("LFT");}

 else if(currDir == MOTOR\_BRAKE){display.println("STP");}

 else{display.println("ERR");}

 display.print("Mode: ");

 if(hasBall){

   display.println("ATK");

 }

 else{

   display.println("DEF");

 }

 display.display();

}

void displayAllRawUltrasonicValues(){

 display.print("US:");

 display.print(int\_format(ultrasonic1.Ranging(CM)));

 display.print("|");

 display.print(int\_format(ultrasonic2.Ranging(CM)));

 display.print("|");

 display.print(int\_format(ultrasonic3.Ranging(CM)));

 display.print("|");

 display.print(int\_format(ultrasonic4.Ranging(CM)));

 display.print("|");

 display.print(int\_format(ultrasonic5.Ranging(CM)));

 //display.print("|IR:");  //NOTE: IR printed with Ultrasonic values due to lack of display space

 display.print("\n");

}

void displayAllRawIRvalues(){

//  display.print("IR:"); //NOTE: IR printed with Ultrasonic values due to lack of display space

 for (int j = 0; j < 7; j++) {

   IRPins[j] = analogRead(j);

   display.print(IRPins[j]);

   if(j != 7){

     display.print("|");

   }

 }

 display.println();

}

void displayAllHIGHRawIRvalues(){

 display.print("IR:");

 display.print(analogRead(7));

 display.print("|");

 display.print(analogRead(8));

 display.print("|");

 display.print(init\_up\_1\_val);

 display.print("|");

 display.print(init\_up\_2\_val);

 display.println();

}

String int\_format(int i){

 if(i > 99){

   return "99";

 }

 else if(i < 10){

   return "0" + String(i);

 }

 else{

   return String(i);

 }

}