Team 17: CowShots

Oriana Peltzer, Qizhan Tam,
Sai Anurag, Shaun Harris

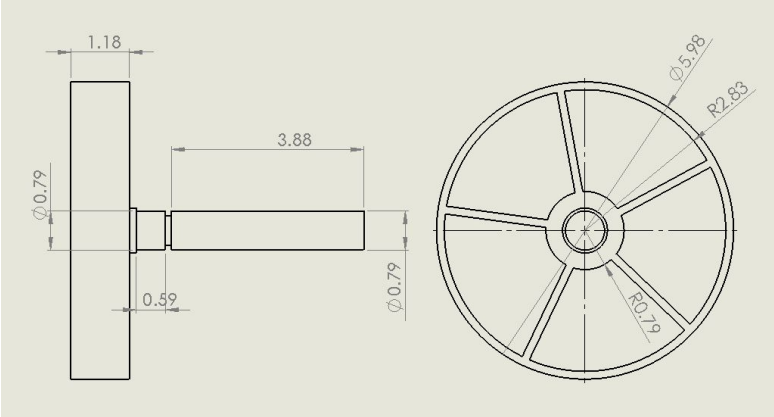
Contents

1. Mechanical System

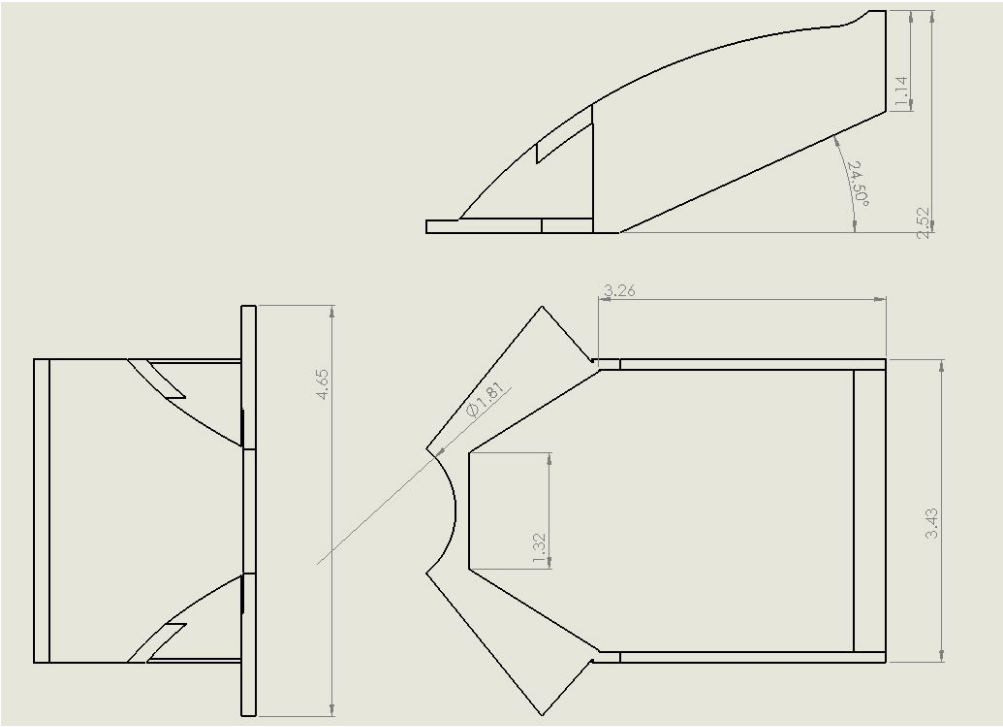
1.1 Part Drawings

Ball-Drop Mechanism

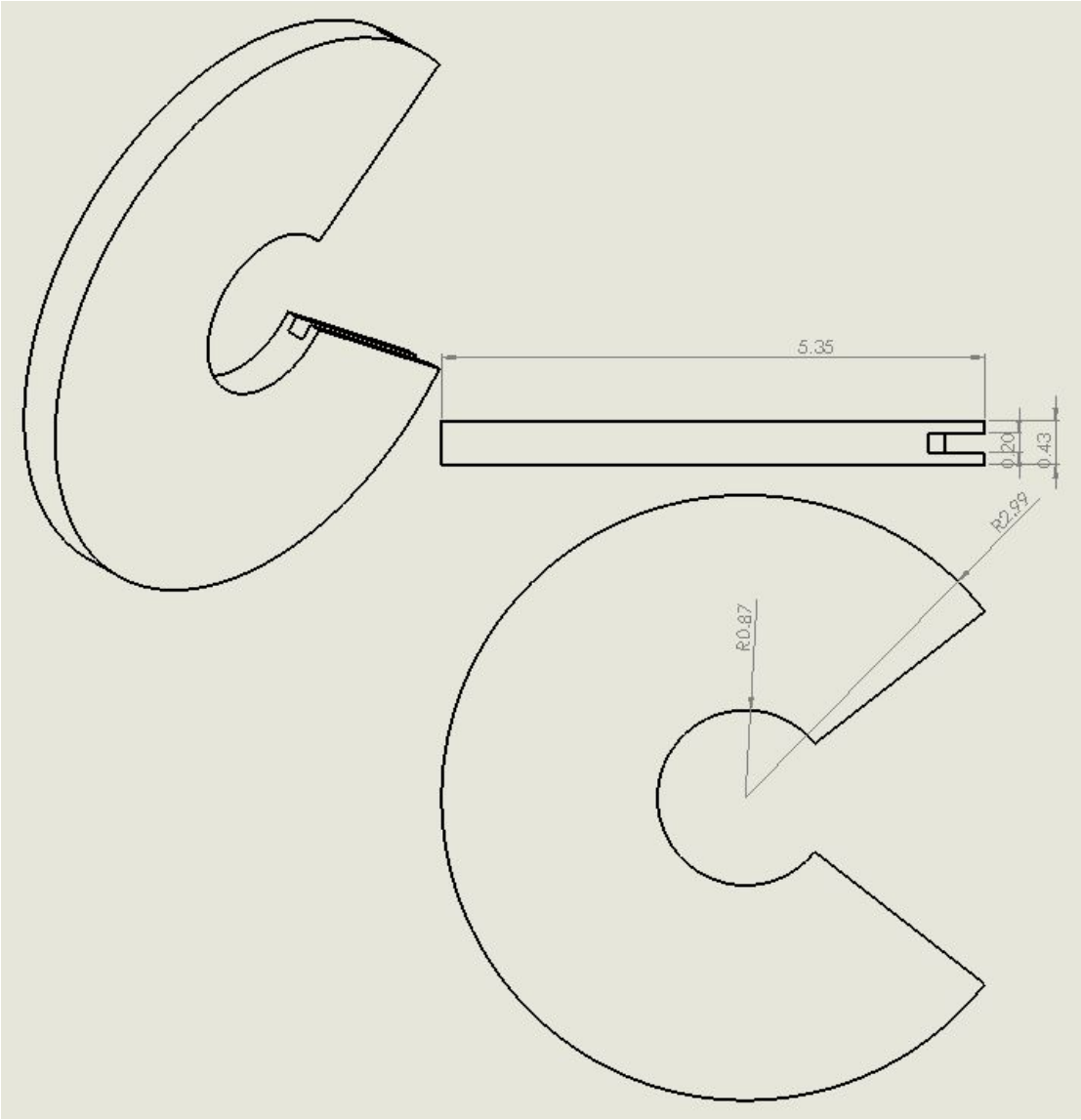
a) Rotating Wheel



b) Slide

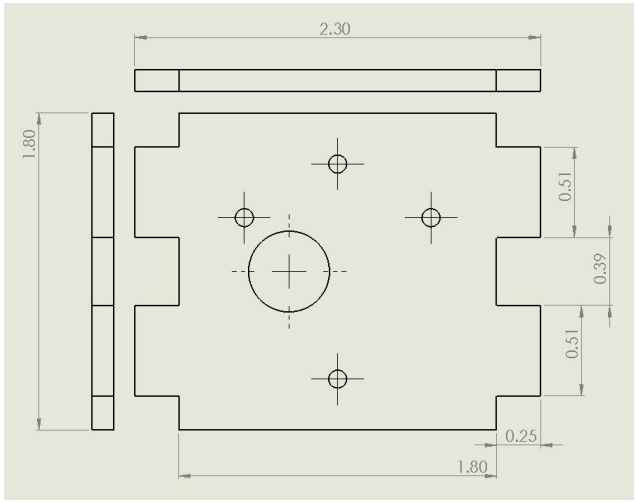


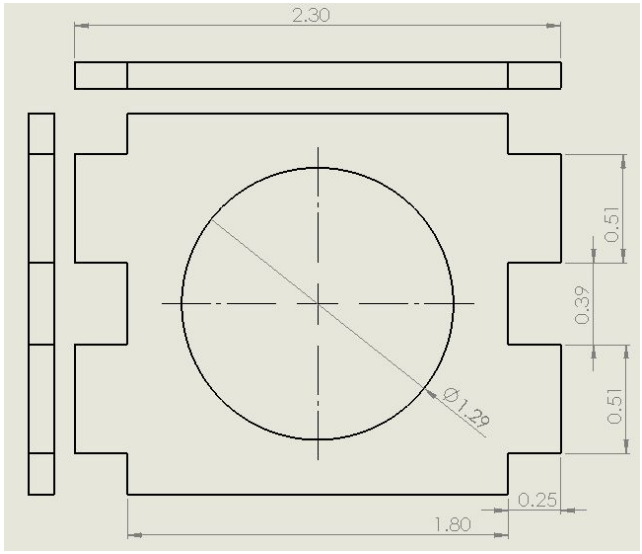
c) Rotating Wheel Holder



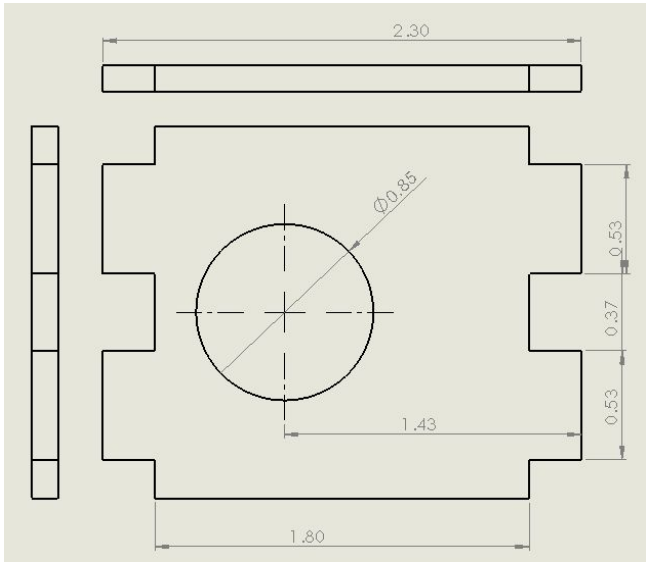
Drive Mechanism

a) Motor Fixtures

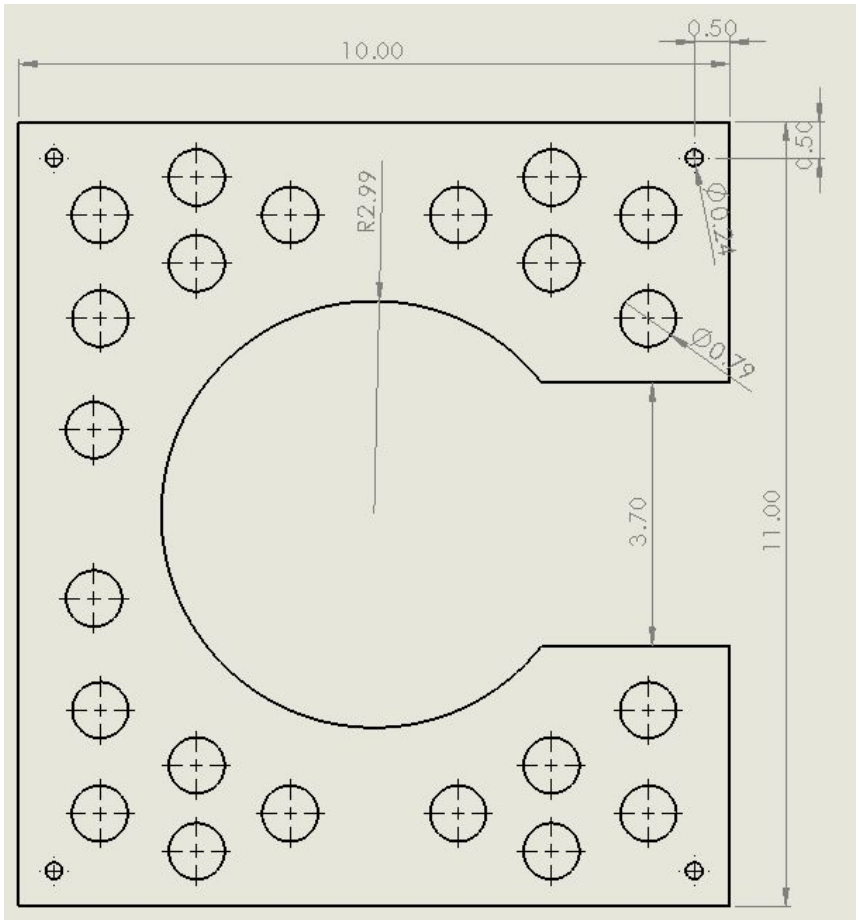


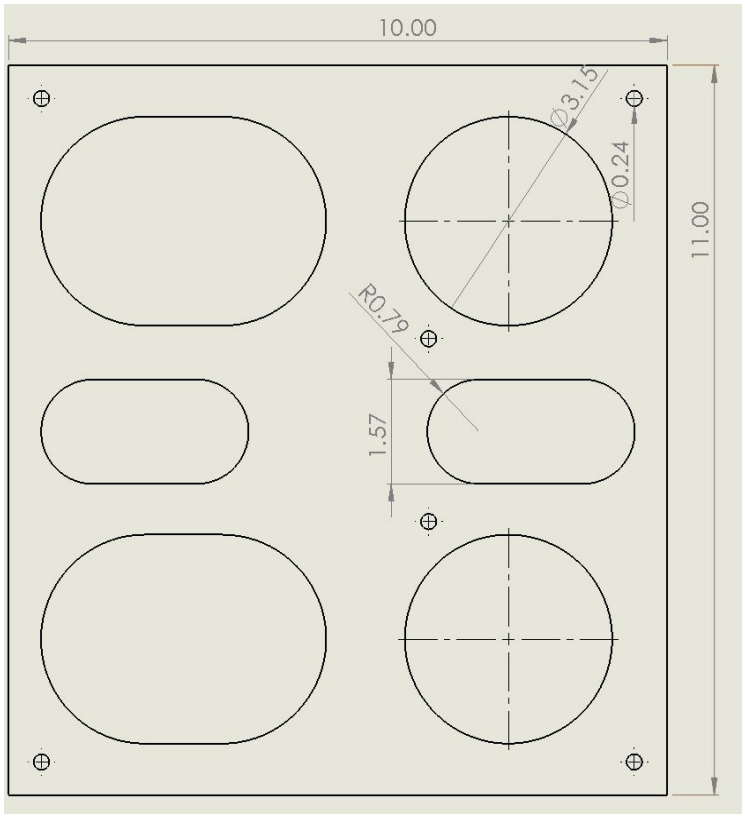


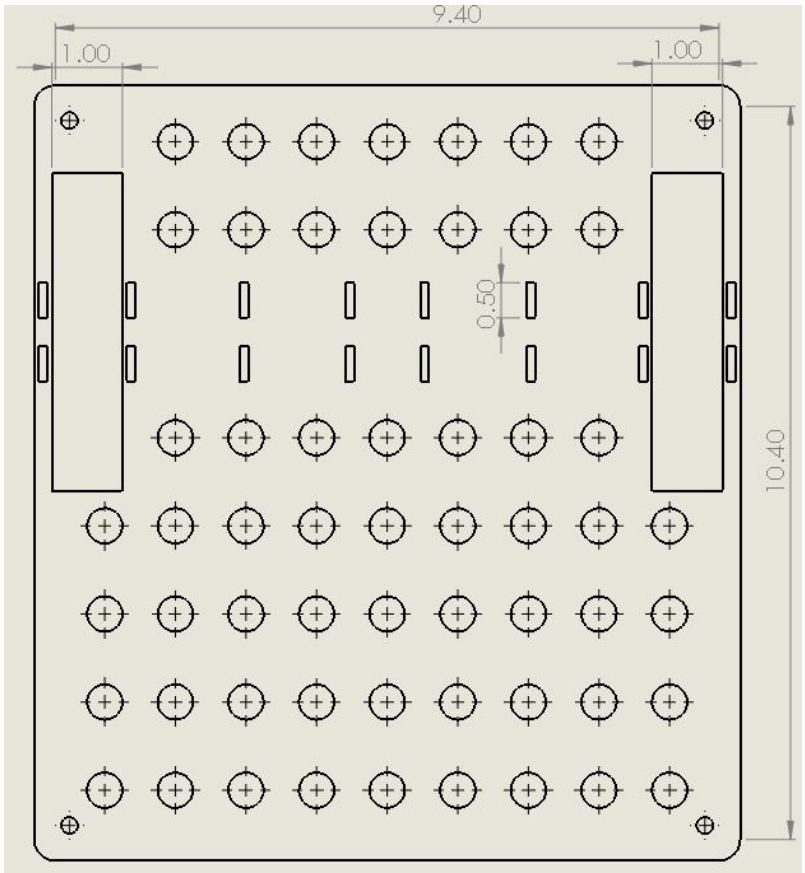
b) Drive Shaft Bearing Fixtures

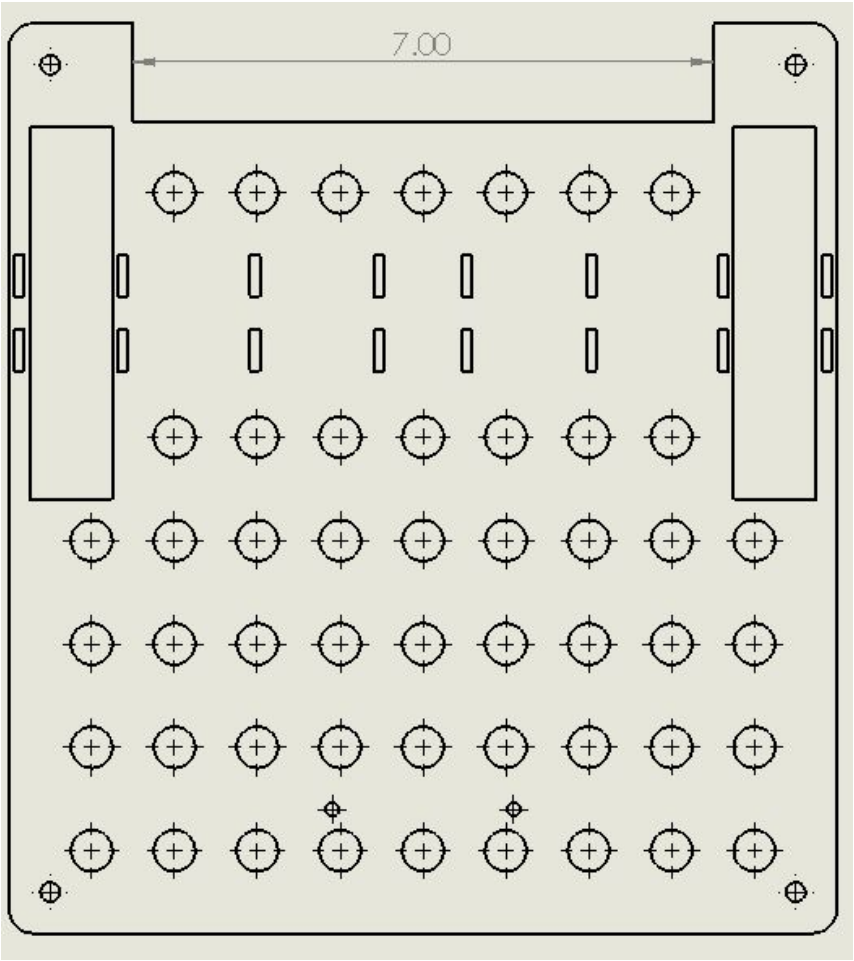


Platforms (Top → Bottom)

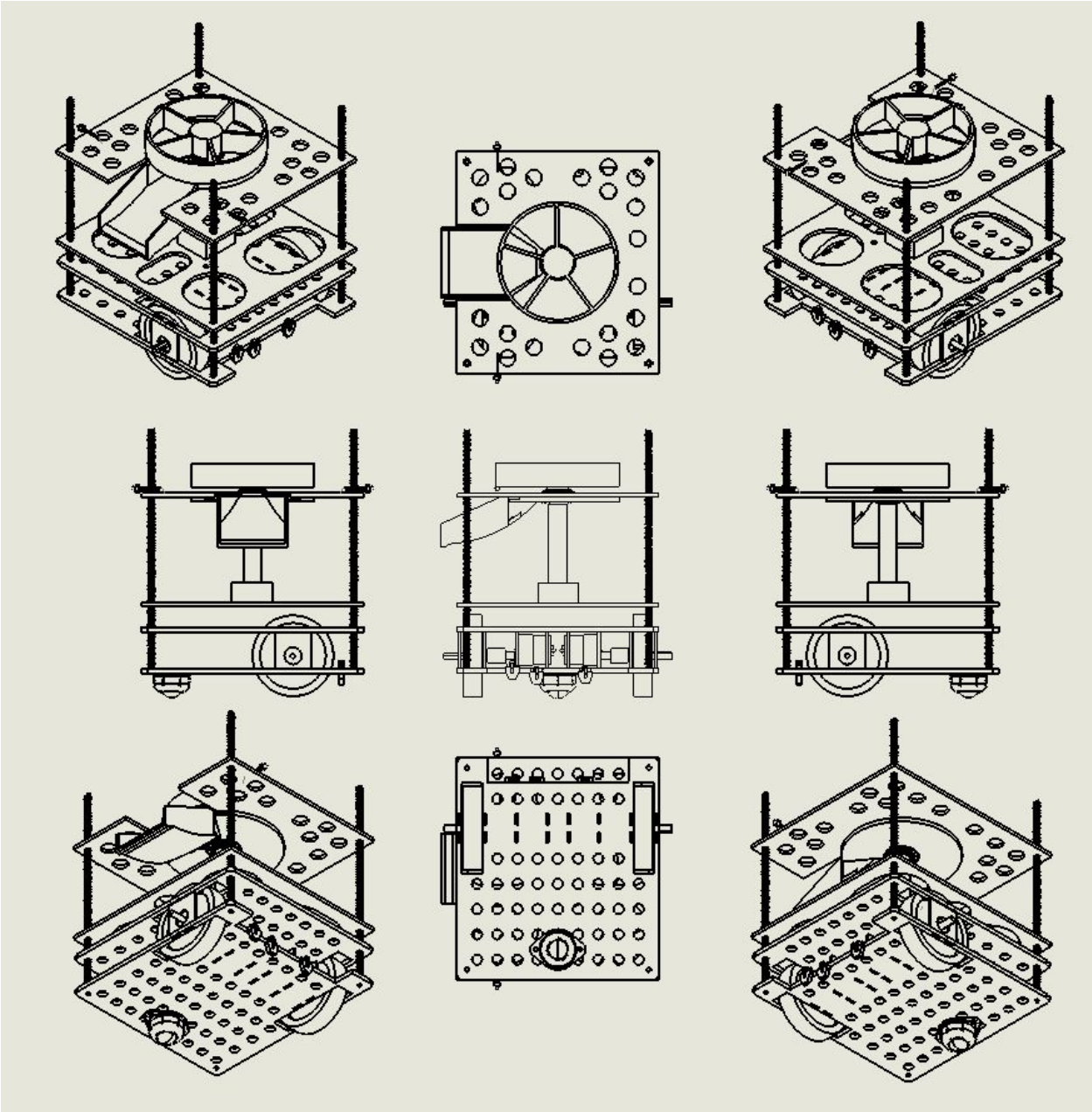


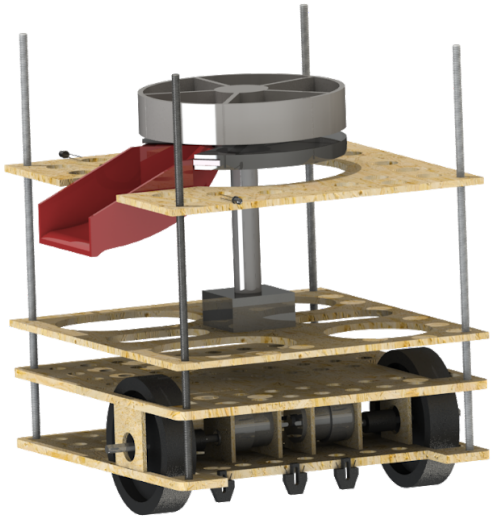

10.00
11.00
Ø3.15
Ø0.24
R0.79
1.57

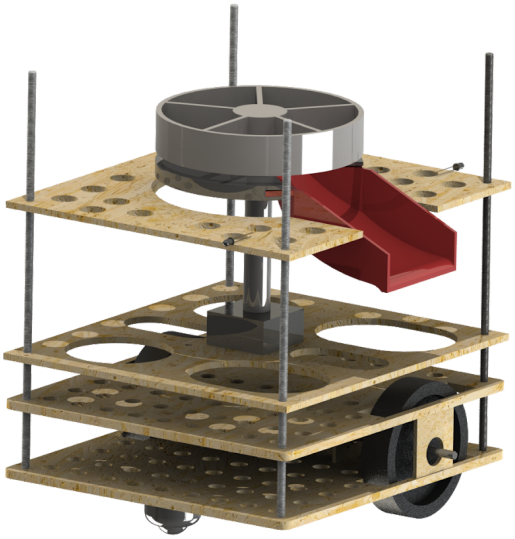

9.40
1.00
1.00
0.50
10.40


7.00

1.1.1 Assembly Drawings







1.2 Design Basis

Threaded rods were used to hold each platform in place as they provided sufficient strength for structural integrity and more importantly they allowed the flexibility for the platform heights to be adjusted quickly and also the ability for the robot to be designed in 2 large subsystems: the ball drop mechanism on the top and the drive mechanism on the bottom.

The ball drop mechanism was designed such that it was robust to potential shaking or variations in the balls. In contrast to using linear actuators to allow only a certain number of balls to drop using a quick timing linear actuator, using the stepper motor to rotate the ball wheel is easier to calibrate and is robust to uncertainties.

Press fits were used to hold the drivetrain, motors and wheels in place as they ensured positioning precision and provided sufficient structural strength to prevent the drivetrain from exerting radial load on the motor output shaft. Among the alternatives considered were just gluing the supports in place which would greatly simplify the design process but the trade-off would be the positioning will be imprecise which may lead to the axial load on the motor output shafts and motor torque not transmitted inefficiently to the wheels. Fasteners using bolts would provide the greatest strength and precision but they would add complexity to the design process as tap holes would be required. We determined that the additional strength and precision provided by fasteners were unnecessary for the load that the drive shafts would be bearing. Spider couplers were used to interface the motor output shafts with the wheel shafts to act as a buffer against potential misalignments in the shafts' concentricity.

The 2-wheel drive design was chosen for its simplicity, light-weight and low-cost design. It was compared against the 3/4-wheel (omniwheel) drive design which would make turning actions easier to implement but it would add additional failure modes to the robot by increasing circuitry complexity and it will also drastically increase (~+20%) the cost required to build the robot as the motors are the costliest components based on our design.

1.3 Parts List and Datasheets

Parts List on Google Drive

Part	ID	price	Final Design Number of items	Total Cost
In TA office				
vex 2 wire motor 393		$ 15.00	2	$ 30.00
vex shaft 3 in		$ 0.50	2	$ 1.00
skateboard wheels to work with		$ 3.65	2	$ 7.30

vex				
Vex bearing flats		$ 0.50	6	$ 3.00
d shaft ¼ by 6 in	634084	$ 2.89	2	$ 5.78
screw shaft ¼ by 12 in		$ 0.60	4	$ 2.40
spider coupler hub 0.197 in		$ 1.25	2	$ 2.50
insert for spider mcmaster		$ 1.50	1	$ 1.50
Stepper motor	42by48h07	$ 6.25	1	$ 6.25
Tamiya connector case		$ 0.50	2	$ 1.00
Tamiya metal crimp		$ 0.25	4	$ 1.00
Linear regulator	LM338	$ 1.50	2	$ 3.00
				$ -
Other				$ -
IR sensing	OPB703	$ 4.00	3	$ 12.00
Polulu Ball Caster		$ 2.99	1	$ 2.99
Miscellaneous Nuts and Bolts from Ace Hardware		$ 14.05	1	$ 14.05
				$ -
Lab Kit and provided				$ -
3000mAh Gens Ace 7.2V NiMH rechargaeble battery			2	$ -
Circuit Breaker			1	$ -
Teensy			1	$ -
IR phototransistor	LTR-3208E	$ 0.51	0	$ -
L293			0	$ -
DRV8825		$ 10.00	1	$ 10.00
				$ -
Miscellaneous items				$ -
Top wheel		$ 1.50	1	$ 1.50
MDF		$ 6.00	1	$ 6.00
MDF		$ 3.00	1	$ 3.00
4 x 47Ohm resistors		$ 0.20	1	$ 0.20
Spyder coupler for D_shaft		$ 1.25	1	$ 1.25
zip ties for batteries		$ 0.20	1	$ 0.20
Slide and Stepper support		$ 7.00	1	$ 7.00
Disfunctional Coupler		$ 1.00	1	$ 1.00

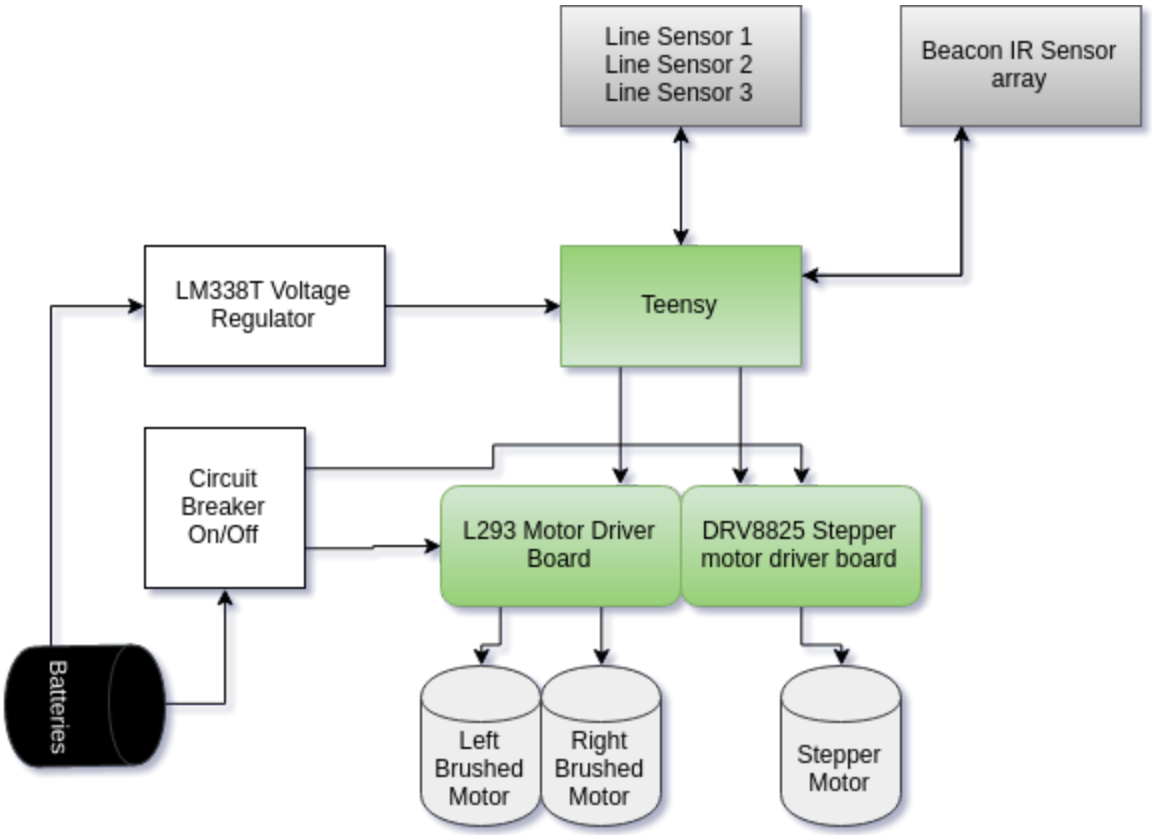
				$ -
			TOTAL	$ 123.92

2. Electrical

2.1 Schematic Drawings

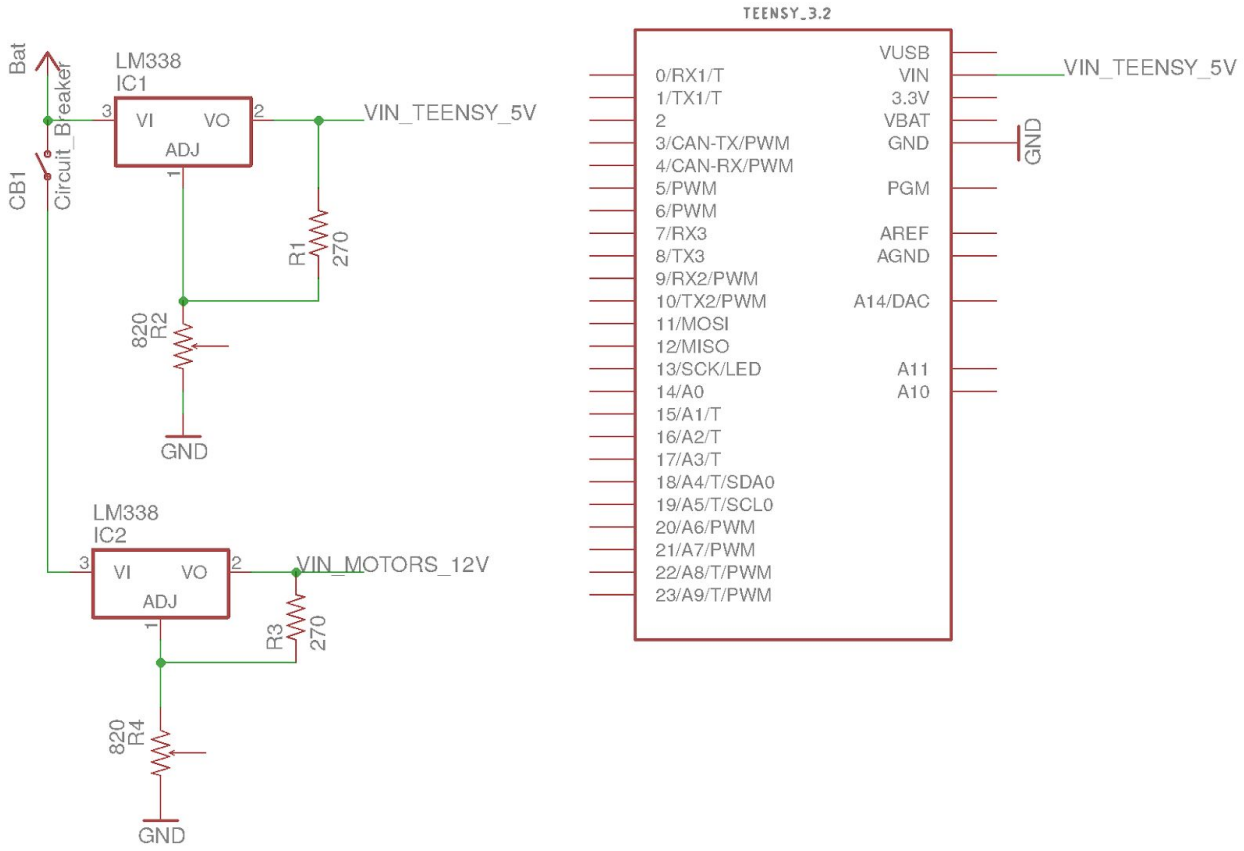
2.1.1. System block Diagram

The whole system block diagram is shown here.



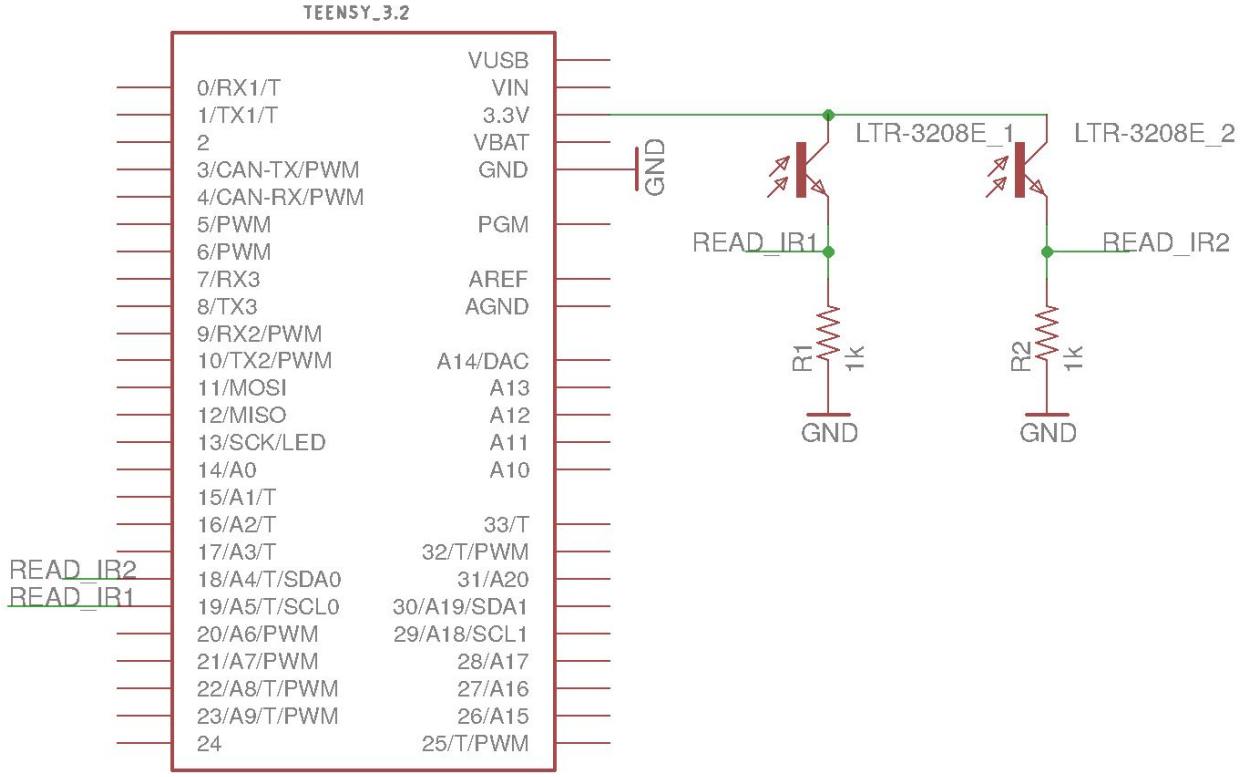
The battery circuit will use a linear voltage regulator to regulate the voltage on the teensy, however, the motor voltage does not need to be regulated given the voltage range of the batteries in use.

2.1.2 : Integration of battery, Teensy, Voltage Regulator



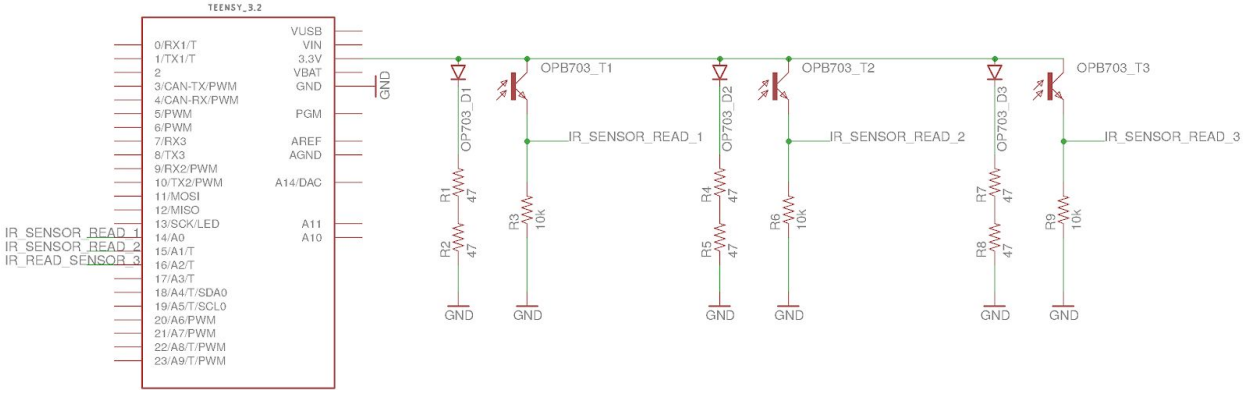
2.1.3 Schematic: Beacon Sensing

The IR beacon sensing diagram is shown here. This can have as many of these circuits in parallel as is required. This will be analyzed via trial and error.



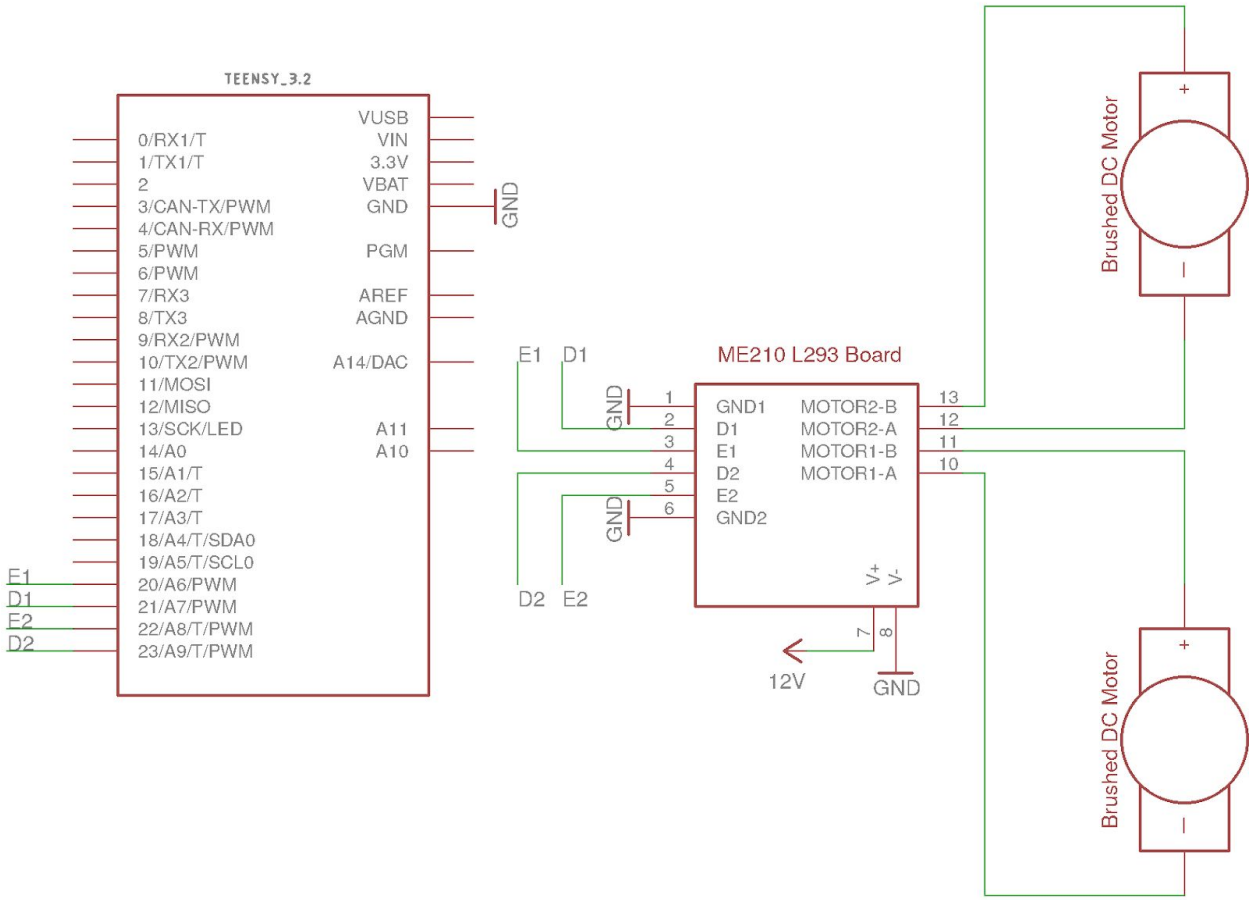
2.1.4 : Schematic : IR Line Sensing

The IR line following diagram is shown here. This will use at least 3 of these circuits in parallel, however, more can be used if found necessary. These configurations are similar to the circuits used in Lab 1.



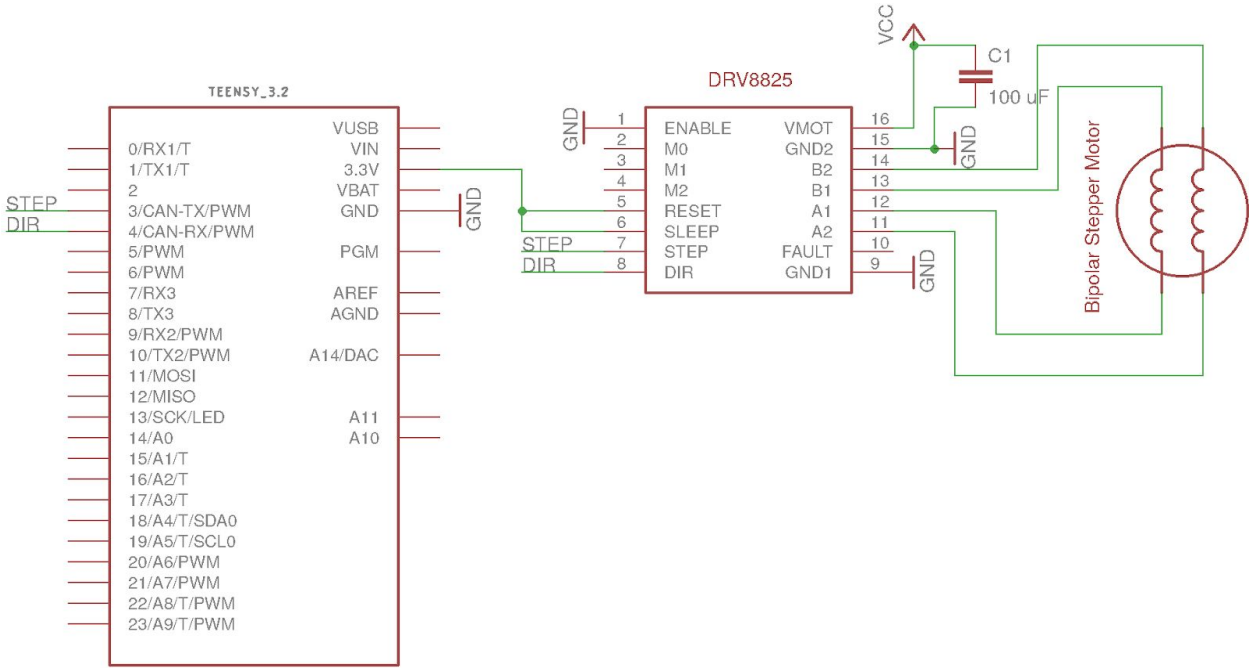
2.1.5 Schematic: DC Motors Circuit

The motor circuit is shown here for the 2 DC motor. This is similar to the circuitry used in Lab 2.



2.1.6 : Schematic Stepper Motor

The circuit for the stepper motor is shown here. This is also similar to the circuit found in Lab 2.



2.2 Design Basis

2.2.1 Digital Inputs - Digital Outputs Compatibility Study

a) Teensy - input, L 293 - Output
System Requirement: Control Motor Using Teensy
Calculations

Compatability - Teensy, L293				
Output		Input		Units
Parameter	Teensy	Parameter	L293	
VOH	2.8	VIH	2.3	V
VOL	0.5	VIL	1.5	V
VOH>VIH	YES	VOL<VIL	YES	SAFE

Since VOL< VIL, VOH> VIH, L293 can be used to control motor if Teensy microcontroller is used. It must be checked if L293 can sink the amount of desired current
b) Teensy - input, DRV 8825 Output
System Requirement: Control Stepper Motor Using Teensy
Calculations

Compatability - Teensy, DRV8825				
Output		Input		Units
Parameter	Teensy	Parameter	L293	
VOH	2.8	VIH	2.2	V
VOL	0.5	VIL	0.7	V
VOH>VIH	YES	VOL<VIL	YES	SAFE

Since VOL< VIL, VOH> VIH, DRV8825 can be used to control motor if Teensy microcontroller is used. It must be checked if DRV8825 can sink the amount of desired current

For line sensing and beacon sensing, analog inputs are used. The only user requirement is voltage being read must be readable by Teensy. Thus, any analog signal being read by Teensy must have a voltage signal less than 3.8 V

2.2.2 Circuit Design Considerations - Teensy, Battery, Circuit breaker integration

Teensy can have a maximum battery input voltage of 3.8 V. Voltage Regulator and potentiometer were used to ensure the same.

Further, voltage regulator selected can allow a maximum current of 5A. At any time instant, the net current being drawn by the circuit must be less than 5A. If the net current drawn by motors (which draw the max load) is less than 80% of 5A, the circuit should be safe since other circuits draw little current. This point was taken into consideration during design phase

2.2.3 Line Sensing Circuit

OPB 704 device is chosen for this application. It is a combination of a IR light emitting diode and npn photo transistor. The current that flows through the photo transistor is dependent on the surface upon which IR light is incident. Since it is an analog signal that is being tracked, the question of compatibility doesn't arise. The data sheet gave 40mA as an order of magnitude current that flows through the IR LED.

To ensure that this is the max current that flows through the LED, resistance was chosen. (In worst case scenario, voltage drop across diode is 0, i max = V teensy/ R). Standard Resistors 47 and 47 ohms were connected in series based on this calculation.

The signal corresponding current through the photo transistor was measured by measuring the voltage drop that will occur over a resistor if the current flows through it. Resistor value was chosen accordingly so that the signal is of the order of magnitude of V.

2.2.4 DC Motor Circuit

User Requirements:
a) Motors provide enough torque so that the robot overcomes static friction
b) Motors provide enough torque at the desired speed
c) Estimate the voltage required
d) Estimate the max current/normal current for sink considerations
e) Design the sink accordingly

Motor Specifications

Motor Specs	Jameco 161382	Units	Source
Gear Ratio	30:1	Ratio	Datasheet
NL Speed	200	RPM	Datasheet
Ke	0.5729577951	V/(rad/s)	Calculated
No Load Current	66	mA	Datasheet
Rated Torque	850	gm-cm	Datasheet
Rated Current	306	mA	Datasheet
Kt	0.2722222222	Nm/A	
R	6.451612903	Ohms	Calculated

Motor evaluation to check if it meets user demands

Weight	3	Kg
Normal Force	29.4	N
Static Friction Factor	0.6	(Worst Case)
Rolling Friction Factor	0.047	
Static F Force	17.64	N
Wheel Size	4	inch Dia
Drive Stall Torque Calculation		

No of motors	2	
Stall T per motor req	0.45	Nm (Required)
Stall T Actual Per Motor	0.49	Nm
Rolling Friction Force	1.23	N
Drive torque during motion	0.03	N m
Drive current	0.12	Amps
Speed	0.20	m/s
Wheel Speed	4.00	rad/s
Back EMF	3.82	V
NL EMF	0.43	V
Torque EMF	0.74	V
Total V	4.99	V
Is Total < 12V?	Yes	
Max Starting Current	1.794	Amps

Note:
Rolling friction factor has been estimated using the information provided in this link - (http://www.mhi.org/media/members/14220/130101690137732025.pdf)

Inferences:

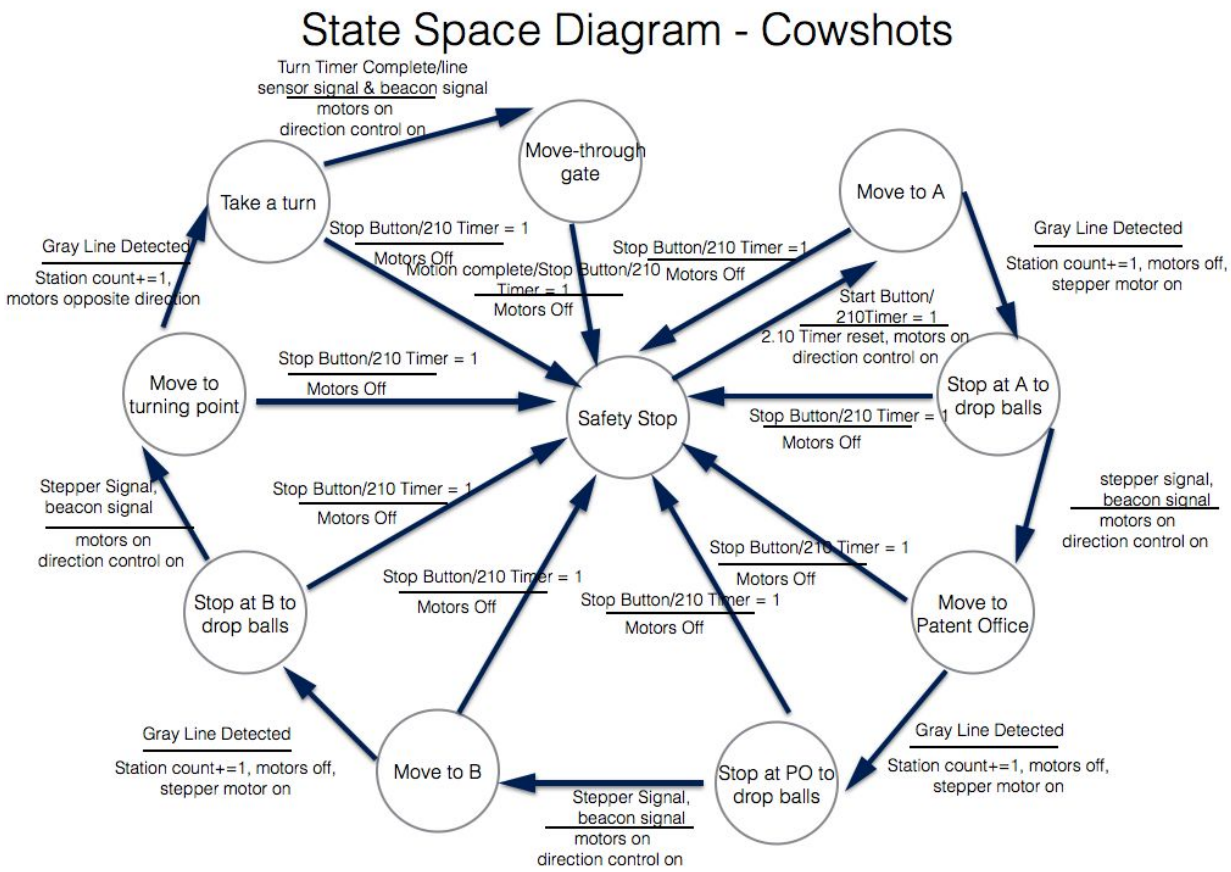
1. The Motor can safely drive the system that has to be driven
2. The max starting current is 1.794A, the normal operating current is much less than 1A
3. L293 can sink an instantaneous current of 2A and a normal current of 1A. L293 can be used for this application.
4. The desired speed can be achieved using the 12V battery and PWM model.
5. Since the normal operating current is much less than 1 A, the voltage regulator is safe for use since it can take a max of 5 A.

2.2.5 Stepper Motor Circuit

The amount of torque required is extremely low. All it needs to push is a small ball. When the motor available in lab was run, it was observed that it can safely push the ball. The designers in the team believed that the application does not require modelling based thinking when heuristic experience is giving a clear signal and since the current/torque associated with the application is so small.

3. Software

3.1 State Space Diagram



3.2 Implementation

The current software presented here allows for a function to follow a line. This incorporates features of line sensing, and DC motor control.

The main code is listed here:

```
/*--------------Includes-----------------------------------*/
#include <IntervalTimer.h>
#include <AccelStepper.h>
#include <Metro.h>
#include "GlobalVariables.h"
```

```
// CowShots created header files
#include "LED_Blink.h"

#include "Motor.h"          // functions to control the motor
#include "Stepper.h"

// global variables needed for both Line sensing and Line following
#include "Line_Sensing.h"   // function to sense the line
#include "Line_Following.h" // function to follow the line
#include "State_Machine.h"

/*---------------Module Defines-----------------------------*/
// #define LIGHT_THRESHOLD 172  // trial and error thresholds
// #define LINE_THRESHOLD 10    // trial and error thresholds



/*---------------Module Function Prototypes-----------------*/
void blink_LED();
void Resp_to_key_motor(char);

/*---------------State Definitions--------------------------*/

/*---------------Module Variables---------------------------*/



/*---------------CowShots Main Functions----------------*/
void setup() {
  // put your setup code here, to run once:
  pinMode(ledPin,OUTPUT);
  digitalWrite(ledPin,ledState);
  Serial.begin(9600);

  Setup_LED_Blink();
  Setup_Motor_Pins();
  Setup_Line_Sensors();

  //Initialize actions for the first state
  //Setup_Line_Following();
  Setup_Line_Following_PID();
  //GoForward();
  //state=STATE_TEST;
```

```
  state = STATE_MOVE_TO_A;
  //state = STATE_STOP_AT_B;
  //state = STATE_MOVE_TO_TURN;

  //Setup_Line_Sampling_Print();
  Setup_Stepper();
}

void loop() {
  // put your main code here, to run repeatedly:
  if (Serial.available()){
    char key_serial_monitor = Serial.read();
    Serial.print("Read in key: ");
    Serial.println(key_serial_monitor);
    Resp_to_key_motor(key_serial_monitor);
  }
  //Follow_Line();
  //stepmotor.setSpeed(140);
  //stepmotor.runSpeed();
  switch (state) {
    case STATE_MOVE_TO_A:
      handleMoveToA();
      break;
    case STATE_STOP_AT_A:
      handleStopAtA();
      break;
    case STATE_MOVE_TO_PATENT_OFFICE:
      handleMoveToPatentOffice();
      break;
    case STATE_STOP_AT_PATENT_OFFICE:
      handleStopAtPatentOffice();
      break;
    case STATE_MOVE_TO_B:
      handleMoveToB();
      break;
    case STATE_STOP_AT_B:
      handleStopAtB();
      break;
    case STATE_MOVE_TO_TURN:
      handleMoveToTurn();
      break;
    case STATE_WAIT_FOR_TURN:
      handleWaitForTurn();
```

```
      break;
    case STATE_TAKE_A_TURN:
      handleTakeATurn();
      break;
    case STATE_MOVE_TO_GATE:
      handleMoveToGate();
      break;
    case STATE_STOP_AT_GATE:
      handleStopAtGate();
      break;
    case STATE_TEST:
      break;
    default:    // Should never get into an unhandled state
      Serial.print("What is this I do not even...");
      Serial.println(state);
  }
  stepmotor.run();
}

/*----------------Module Functions-------------------------*/
void Resp_to_key_motor(char a){
  if (a=='0'){
    //E1_state=LOW;
    //D1_state=LOW;

    state=STATE_TEST;
    Line_Sampling_Timer.end();
    Motor_Stop();
  }
  else if (a=='1'){//direction 1
    state=STATE_TEST;
    GoBackwards();

  }
  else if (a=='2'){//direction 2
    GoForward();
    state=STATE_TEST;

  }
  else if (a=='s'){// stepper motor go
    //runsteppermotor();
    //stepmotor.setSpeed(1400);
    stepmotor.move(10);
```

```
  Serial.println("Stepper Motor moved one");
}
else if (a=='c'){// check if stepper motor is done
  Serial.print("DistanceToGo = ");
  Serial.println(stepmotor.distanceToGo());
}
else if (a=='l'){// what is my line sensor 1 reading?
  UpdateLineSensorValues();
  Sensor_1_Color = Get_Color1(Sensor_1);
  Sensor_2_Color = Get_Color(Sensor_2);
  Sensor_3_Color = Get_Color(Sensor_3);

  Serial.println("Your line sensor 1 is reading ");
  Serial.print(Sensor_1_Color);
  Serial.print(" ");
  Serial.println(Sensor_1);
  Serial.print(Sensor_2_Color);
  Serial.print(" ");
  Serial.println(Sensor_2);
  Serial.print(Sensor_3_Color);
  Serial.print(" ");
  Serial.println(Sensor_3);
}
else if (a=='t'){// what state am I in?
  Serial.print("You are in state = ");
  Serial.println(state);
}
else if (a=='r'){
  noInterrupts();
  Serial.print("Correction = ");
  Serial.println(correction);
  Serial.print("error = ");
  Serial.println(error);
  interrupts();
}
else{
  Serial.println("Key not recognized");
}
analogWrite(E1,E1_state);
digitalWrite(D1,D1_state);
analogWrite(E2,E2_state);
digitalWrite(D2,D2_state);
//Serial.print("digitalWrite ");
```

```
  //Serial.print(E1_state);
  //Serial.print(" ");
  //Serial.println(D1_state);
}
```

3.3 The Motor.h header is shown here:

```
/*
 * Functions to be used to output motor control
*/

// ---------------------- PINS ------------------------- //
int E1 = 20;
int D1 = 21;
int D1_state = LOW;
int E1_state = LOW;
int E2 = 23;
int D2 = 22;
int D2_state = LOW;
int E2_state = LOW;
// ----------------------------------------------------- //

int DutyCycle = 40; //Our reference speed
int Max_Speed = 200; //We use this to apply saturation
int Right_Speed = DutyCycle;
int Left_Speed = DutyCycle;
int Right_Direction = LOW; //These could eventually replace D1_state if it works out
int Left_Direction = LOW;

// ------------------- SETUP MOTOR --------------------- //
void Setup_Motor_Pins(void) {
  pinMode(E1,OUTPUT);
  pinMode(D1,OUTPUT);
  pinMode(E2,OUTPUT);
  pinMode(D2,OUTPUT);
}
// ----------------------------------------------------- //


// ----------------- Custom Moving --------------------- //
```

```
void Advance(void) {
  //Serial.println(Left_Speed);
  //Serial.println(Right_Speed);
  E1_state = Left_Speed;
  D1_state=Left_Direction;
  E2_state = Right_Speed;
  D2_state=Right_Direction;
  analogWrite(E1,E1_state);
  digitalWrite(D1,D1_state);
  analogWrite(E2,E2_state);
  digitalWrite(D2,D2_state);
}

// ------------------------------------------------------ //

// ----------------- Elementary Moving ----------------- //

void TurnRight(void) {
    E1_state=38;
    D1_state=LOW;
    E2_state=38;
    D2_state=HIGH;
    analogWrite(E1,E1_state);
    digitalWrite(D1,D1_state);
    analogWrite(E2,E2_state);
    digitalWrite(D2,D2_state);
}

void Start_Pulse(void) {
    E1_state=200;
    D1_state=HIGH;
    E2_state=200;
    D2_state=HIGH;
    analogWrite(E1,E1_state);
    digitalWrite(D1,D1_state);
    analogWrite(E2,E2_state);
    digitalWrite(D2,D2_state);
}

void GoBackwards(void) {
    E1_state=DutyCycle;
    D1_state=LOW;
    E2_state=DutyCycle;
```

```
    D2_state=LOW;
    analogWrite(E1,E1_state);
    digitalWrite(D1,D1_state);
    analogWrite(E2,E2_state);
    digitalWrite(D2,D2_state);
}

void GoForward(void) {
    E1_state=DutyCycle;
    D1_state=HIGH;
    E2_state=DutyCycle-4;
    D2_state=HIGH;
    analogWrite(E1,E1_state);
    digitalWrite(D1,D1_state);
    analogWrite(E2,E2_state);
    digitalWrite(D2,D2_state);
}

void Motor_Stop(void) {
    E1_state=LOW;
    D1_state=LOW;
    E2_state=LOW;
    D2_state=LOW;
    analogWrite(E1,E1_state);
    digitalWrite(D1,D1_state);
    analogWrite(E2,E2_state);
    digitalWrite(D2,D2_state);
}
// ------------------------------------------------------- //
```

3.4 The Line_Sensing.h header is shown here:

```
/*
 *Functions to be used for reading the IR sensors to follow the line.
 *This includes reading the analog input to the Teensy from the IR sensors
*/


//Sensor pins
int IR_SENSOR_READ_1 = 14;
```

```
int IR_SENSOR_READ_2 = 15;
int IR_SENSOR_READ_3 = 16;

// Sensor values, between 0 and 1024
unsigned int Sensor_1;
unsigned int Sensor_2;
unsigned int Sensor_3;



//Sensor Timer
IntervalTimer Line_Sampling_Timer_Print;
int sampling_rate_print = 500; //This is in ms so that we store it in smaller variable


// Some module function declarations here --------- //
void UpdateLineSensorValues(void);
void PrintLineSensorValues(void);
int Get_Color(int Sensor_Value);
int Get_Color1(int Sensor_Value);
// ------------------------------------------------ //


// -------------------- SETUP FUNCTIONS ----------------------- //
void Setup_Line_Sensors(void) {
  pinMode(IR_SENSOR_READ_1, INPUT);
  pinMode(IR_SENSOR_READ_2, INPUT);
  pinMode(IR_SENSOR_READ_3, INPUT);
}

void Setup_Line_Sampling_Print(void) {
   Line_Sampling_Timer_Print.begin(PrintLineSensorValues, 10000*sampling_rate_print);
}
void Setup_Line_Sampling(void) {
   Line_Sampling_Timer_Print.begin(UpdateLineSensorValues, 10*sampling_rate_print);
}

void Stop_Line_Sampling(void) {
   Line_Sampling_Timer_Print.end();
}
// ------------------------------------------------------------ //
```

```
// ------------------ Read sensor values ------------------------ //
void Read_IR_Sensor_1(void) {
  Sensor_1 = analogRead(IR_SENSOR_READ_1);
  Sensor_1_Color = Get_Color1(Sensor_1);

}
void Read_IR_Sensor_2(void) {
  Sensor_2 = analogRead(IR_SENSOR_READ_2);
  Sensor_2_Color = Get_Color(Sensor_2);
}
void Read_IR_Sensor_3(void) {
  Sensor_3 = analogRead(IR_SENSOR_READ_3);
  Sensor_3_Color = Get_Color(Sensor_3);
}
// ------------------------------------------------------------ //


void PrintLineSensorValues(void) {
  Read_IR_Sensor_1();
  Read_IR_Sensor_2();
  Read_IR_Sensor_3();
  Serial.println( "------------ LINE SENSING ----------------");
  Serial.print("Sensor 1 Value: ");
  Serial.println(Sensor_1);
  Serial.print("Sensor 2 Value: ");
  Serial.println(Sensor_2);
  Serial.print("Sensor 3 Value: ");
  Serial.println(Sensor_3);
  Serial.println("-----------------------------------------");
  Serial.println();
  Serial.print("Motor speeds ");
  Serial.print(Right_Speed);
  Serial.print(" - ");
  Serial.print(Left_Speed);

}

void UpdateLineSensorValues(void) {
  Read_IR_Sensor_1();
  Read_IR_Sensor_2();
  Read_IR_Sensor_3();
  //Serial.println(Sensor_2);
}
```

```
// ------------------- Identify Color ---------------------- //
int Get_Color(int Sensor_value) {
  //return map(Sensor_value, 0, 500, 0, 100);
  if(Sensor_value < 150) {
    return 0; //Black
  }
  else if (Sensor_value < 400) {
    return 1; //Gray
  }
  else {
    return 2; //White
  }
}
int Get_Color1(int Sensor_value) {
  //return map(Sensor_value, 0, 500, 0, 100);
  if(Sensor_value < 100) {
    return 0; //Black
  }
  else if (Sensor_value < 250) {
    return 1; //Gray
  }
  else {
    return 2; //White
  }
}
// ------------------------------------------------------- //
```

3.5 The Line_Following.h header is shown here.

```
/*
 * Functions to be used for following the Line, this needs functions from Line_Sensing.h
 * But we don't need an extra include here.
*/

//Sensor 1 is on the left, sensor 2 is on the right

// ------------------ VARIABLES ---------------------- //
double previous_error = 0; //For the D of PID
```

```
double cumulated_error=0; //For the I of PID
double error = 0;
double correction = 0;
double correction_saturation=40.;

double Kp = 0.07; //Gains
double Kd = 1.7;
double Ki = 0.00;

typedef enum {
    LEFT, RIGHT
} turn;
turn turn_Remember;


//In case we hit a gray or black turning tape we want to stop following line with an interrupt, using this for example
bool Breaking_Event = 0;
// ------------------------------------------------- //

//Module Functions
void Follow_Line(void);
void Follow_Line_PID(void);
void Reset_PID_vars(void);

// ------------------- TIMER -------------------------- //
// Timer we will use to read IR values and update motor speeds with PID
IntervalTimer Line_Sampling_Timer;
IntervalTimer Reset_PID_vars_Timer;
int sampling_rate = 1; //in ms
// ------------------------------------------------- //


// ------------------------- SETUP FUNCTIONS ---------------------------- //
void Setup_Line_Following(void) {
  Line_Sampling_Timer.begin(Follow_Line, 10*sampling_rate); //10 micro
}
void Setup_Line_Following_PID(void) {
  Line_Sampling_Timer.begin(Follow_Line_PID, 5000*sampling_rate);
  //Reset_PID_vars_Timer.begin(Reset_PID_vars, 300000);
  previous_error = save_previous_error; //For the D of PID
  cumulated_error = 0; //For the I of PID
}
```

```
// ------------------------------------------------------------------- //

// ------------------------- END FUNCTIONS -------------------------- //
void Stop_Line_Following(void) {
  Line_Sampling_Timer.end();
}
void Stop_Line_Following_PID(void) {
  Line_Sampling_Timer.end();
  Reset_PID_vars_Timer.end();
  previous_error = 0; //For the D of PID
  cumulated_error = 0; //For the I of PID
}
// ------------------------------------------------------------------- //


void Follow_Line_PID(void) {

  //Read the IR LED measurements
  UpdateLineSensorValues();

  //Sensor_1_Color = Get_Color1(Sensor_1);
  //Sensor_2_Color = Get_Color(Sensor_2);
  //Sensor_3_Color = Get_Color(Sensor_3);

  //Error calculation
  previous_error = error;
  //error = Sensor_2_Color - Sensor_3_Color; //error is big if we are too far right, so we measure
black on left (3) and white on right (2)
  error=(float(Sensor_2)-float(Sensor_3));
  if (error > 0) {
    error *=1.5;
  }
  cumulated_error += error;

  //Saturation on cumulated error
  if (cumulated_error*Ki > 10) {
    cumulated_error = 10./Ki;
  }
  else if (cumulated_error*Ki < -10) {
    cumulated_error = -10./Ki;
  }

  //Correction calculation
```

```
  correction = Kp*error + Ki*cumulated_error + Kd*(error-previous_error);

  //Saturation on correction
  if (correction > correction_saturation) {
    correction = correction_saturation;
  }
  else if (correction < -correction_saturation) {
    correction = -correction_saturation;
  }

  //Instead of keeping constant speed we will rather set the speeds slower:

  if (correction > 0) { //We want to turn left
    Right_Speed = DutyCycle + correction/2.;
    Left_Speed = DutyCycle - correction/2.;
    Right_Direction = LOW;
    Left_Direction = LOW;

    //Set the directions in case left
    if (Left_Speed < 0) {
      Left_Speed = DutyCycle-correction;
      Left_Direction = HIGH;
    }

  }
  else {
    Left_Speed = DutyCycle - correction/2.;
    Right_Speed = DutyCycle + correction/2.;
    Right_Direction = LOW;
    Left_Direction = LOW;

    if (Right_Speed < 0) {
      Right_Speed = -Right_Speed;
      Right_Direction = HIGH;
    }
  }
  //Serial.println(Right_Speed);
  //Serial.println(correction);
  Advance();

}
```

```
void Follow_Line(void) {
  int change_dutyCycle=10;
  //Read the IR LED measurements
  UpdateLineSensorValues();

  Sensor_1_Color = Get_Color1(Sensor_1);
  Sensor_2_Color = Get_Color(Sensor_2);
  Sensor_3_Color = Get_Color(Sensor_3);


  //Apply saturation
  if (Sensor_2_Color == 2 && Sensor_3_Color == 2) {
    if (turn_Remember == RIGHT){
       Right_Speed = DutyCycle-change_dutyCycle;
       Left_Speed = DutyCycle;
    }
    else if (turn_Remember == LEFT){
       Right_Speed = DutyCycle;
       Left_Speed = DutyCycle - change_dutyCycle;
    }
    //Right_Speed = DutyCycle;
    //Left_Speed = DutyCycle;
    //Serial.println(0);
  } else if (Sensor_2_Color == 0){
    Right_Speed = DutyCycle - change_dutyCycle;
    Left_Speed = DutyCycle;
    turn_Remember=RIGHT;
    //Serial.println(1);
  } else if (Sensor_3_Color == 0){
    Right_Speed = DutyCycle;
    Left_Speed = DutyCycle - change_dutyCycle;
    turn_Remember=LEFT;
    //Serial.println(2);
  }
  else{
      //Serial.println("Something is wrong in applying saturation");
  }
  //Run the motor

  save_previous_error = previous_error;

  Advance();
```

```
}

void Reset_PID_vars(void) {
  cumulated_error = 0;
  previous_error = 0;
}
```

3.6 The State_Machine.h header is shown here.

```
// variables and flags
bool resp_To_Gray_Happened=false;
bool resp_Move_Stepper_Motor=false;
bool TakeATurnHappened=false;
// State machine
typedef enum {
    STATE_MOVE_TO_A,
    STATE_STOP_AT_A,
    STATE_MOVE_TO_PATENT_OFFICE,
    STATE_STOP_AT_PATENT_OFFICE,
    STATE_MOVE_TO_B,
    STATE_STOP_AT_B,
    STATE_MOVE_TO_TURN,
    STATE_WAIT_FOR_TURN,
    STATE_TAKE_A_TURN,
    STATE_MOVE_TO_GATE,
    STATE_STOP_AT_GATE,
    STATE_TEST,
} States_t;

// State variable
States_t state;

// Function declarations
void handleMoveToA();
void handleStopAtA();
void handleMoveToPatentOffice();
void handleStopAtPatentOffice();
void handleMoveToB();
void handleStopAtB();
```

```
void handleMoveToTurn();
void handleWaitForTurn();
void handleTakeATurn();
void handleMoveToGate();
void handleStopAtGate();

// Resp function declarations
void Resp_to_Gray();


// Function Definitions
void handleMoveToA(){
    noInterrupts();
    int Sensor_1_Color_Copy = Sensor_1_Color;
    interrupts();
    if (Sensor_1_Color_Copy == 1){// if gray detected on far right sensor
        Resp_to_Gray();
        resp_To_Gray_Happened=true;
    }
    if (resp_To_Gray_Happened && metroTimer.check()){
        //Line_Sampling_Timer.end(); // stop line following
        Stop_Line_Following_PID();
        Motor_Stop(); // stop motor
        state = STATE_STOP_AT_A;
        Serial.println("state set to STATE_STOP_AT_A");
        resp_To_Gray_Happened=false;
    }
}
void handleStopAtA(){
    if (resp_Move_Stepper_Motor==false){
        Move_Stepper_Motor();
        resp_Move_Stepper_Motor=true;
    }
    Run_Stepper_Motor();
    if (stepmotor.distanceToGo() == 0){
        Setup_Line_Following_PID();
        state = STATE_MOVE_TO_PATENT_OFFICE;
        Serial.println("state set to STATE_MOVE_TO_PATENT_OFFICE");
        resp_Move_Stepper_Motor=false;
    }
}
void handleMoveToPatentOffice(){
    noInterrupts();
```

```
    int Sensor_1_Color_Copy = Sensor_1_Color;
    interrupts();
    if (Sensor_1_Color_Copy == 1){// if gray detected on far right sensor
        Resp_to_Gray();
        resp_To_Gray_Happened=true;
    }
    if (resp_To_Gray_Happened && metroTimer.check()){
        //Line_Sampling_Timer.end(); // stop line following
        Stop_Line_Following_PID();
        Motor_Stop(); // stop motor
        state = STATE_STOP_AT_PATENT_OFFICE;
        Serial.println("state set to STATE_STOP_AT_PATENT_OFFICE");
        resp_To_Gray_Happened=false;
    }
}
void handleStopAtPatentOffice(){
    if (resp_Move_Stepper_Motor==false){
        Move_Stepper_Motor();
        resp_Move_Stepper_Motor=true;
    }
    Run_Stepper_Motor();
    if (stepmotor.distanceToGo() == 0){
        Setup_Line_Following_PID();
        state = STATE_MOVE_TO_B;
        Serial.println("state set to STATE_MOVE_TO_B");
        resp_Move_Stepper_Motor=false;

    }
}
void handleMoveToB(){
    noInterrupts();
    int Sensor_1_Color_Copy = Sensor_1_Color;
    interrupts();
    if (Sensor_1_Color_Copy == 1){// if gray detected on far right sensor
        Resp_to_Gray();
        resp_To_Gray_Happened=true;
    }
    if (resp_To_Gray_Happened && metroTimer.check()){
        //Line_Sampling_Timer.end(); // stop line following
        Stop_Line_Following_PID();
        Motor_Stop(); // stop motor
        state = STATE_STOP_AT_B;
        Serial.println("state set to STOP_AT_B");
```

```
        resp_To_Gray_Happened=false;
    }
}
void handleStopAtB(){
    if (resp_Move_Stepper_Motor==false){
        Move_Stepper_Motor();
        resp_Move_Stepper_Motor=true;
    }
    Run_Stepper_Motor();
    if (stepmotor.distanceToGo() == 0){
        Setup_Line_Following_PID();
        state = STATE_MOVE_TO_TURN;
        Serial.println("state set to MOVE_TO_TURN");
        resp_Move_Stepper_Motor=false;
    }
}
void handleMoveToTurn(){
  noInterrupts();
  Sensor_1_Color = Get_Color1(Sensor_1);
  interrupts();

  if (Sensor_1_Color == 0) {
    //Line_Sampling_Timer.end();
    Stop_Line_Following_PID();
    //Setup_Line_Sampling;
    //TurnRight();
    metroTimer.interval(timer_WaitForTurn);
    metroTimer.reset();
    state = STATE_WAIT_FOR_TURN;
    Serial.println("state set to WAIT_FOR_TURN");
  }
}
void handleWaitForTurn() {
  if (metroTimer.check() == 1) {

    TurnRight();
    save_previous_error = 0;
    state = STATE_TAKE_A_TURN;
    Serial.println("state set to TAKE_A_TURN");
  }
}
void handleTakeATurn(){
```

```
  UpdateLineSensorValues();
  noInterrupts();
  Sensor_2_Color = Get_Color(Sensor_2);
  interrupts();

  if (Sensor_2_Color == 0) {
    //Stop_Line_Sampling();
    //DutyCycle=43;
    DutyCycle=25;
    Setup_Line_Following_PID();
    state = STATE_MOVE_TO_GATE;
    Serial.println("state set to MOVE_TO_GATE");
    //Motor_Stop();
    metroTimer.interval(timer_gray);
    metroTimer.reset();
    metroPostTurn.interval(timer_PostTurn);
    metroPostTurn.reset();
  }

}
void handleMoveToGate(){
    //UpdateLineSensorValues();
    noInterrupts();
    int Sensor_1_Color_Copy = Sensor_1_Color;
    interrupts();
    if (DutyCycle<40 && metroPostTurn.check()){
      DutyCycle++;
    }
    if (TakeATurnHappened==false && metroTimer.check()){
       TakeATurnHappened=true;
       Serial.println("TakeATurnHappened");
    }
    if (resp_To_Gray_Happened==false && TakeATurnHappened && Sensor_1_Color_Copy ==
0){// if black detected on far right sensor
        metroTimer.interval(timer_PastBlackGate);//Resp_to_Gray();
        metroTimer.reset();
        resp_To_Gray_Happened=true;
        Serial.println("RespToGrayHappened");
    }
    if (TakeATurnHappened && resp_To_Gray_Happened && metroTimer.check()){
        //Line_Sampling_Timer.end(); // stop line following
        Stop_Line_Following_PID();
        Motor_Stop(); // stop motor
```

```
        state = STATE_STOP_AT_GATE;
        Serial.println("state set to STOP_AT_GATE");
        resp_To_Gray_Happened=false;
    }
}
void handleStopAtGate(){
}


// Resp Function Definitions
void Resp_to_Gray(){// end LineFollow
    metroTimer.interval(timer_gray);
    metroTimer.reset();
}
```

3.7 GlobalVariables.h header is shown here.

```
/*
 * Global Variables to be used for every header .h file.
 *
*/
// Global Includes
int timer_gray = 300; // in milliseconds
//int timer_stepper = 2000; // in milliseconds
//int timer_pulse = 10; //in milliseconds
int timer_WaitForTurn=200;
int timer_PostTurn = 200;
int timer_PastBlackGate = 1000;
double save_previous_error=0;
//IntervalTimer little_Timer;
Metro metroTimer = Metro(timer_gray);
Metro metroPostTurn = Metro(timer_PostTurn);
//Metro pulseTimer = Metro(timer_pulse);

int Sensor_1_Color;
int Sensor_2_Color;
int Sensor_3_Color;
```

3.8 LED_Blink.h header is shown here.

```
// onboard LED
int ledState=LOW;
int ledPin=13;

IntervalTimer myTimer_LED;


void blink_LED(){
  if (ledState==LOW) ledState=HIGH;
  else ledState=LOW;
  digitalWrite(ledPin,ledState);
  //Serial.println("blink_LED called");
}
void Setup_LED_Blink(){
  myTimer_LED.begin(blink_LED,1000000);
}
```

3.9 Stepper.h header is shown here.

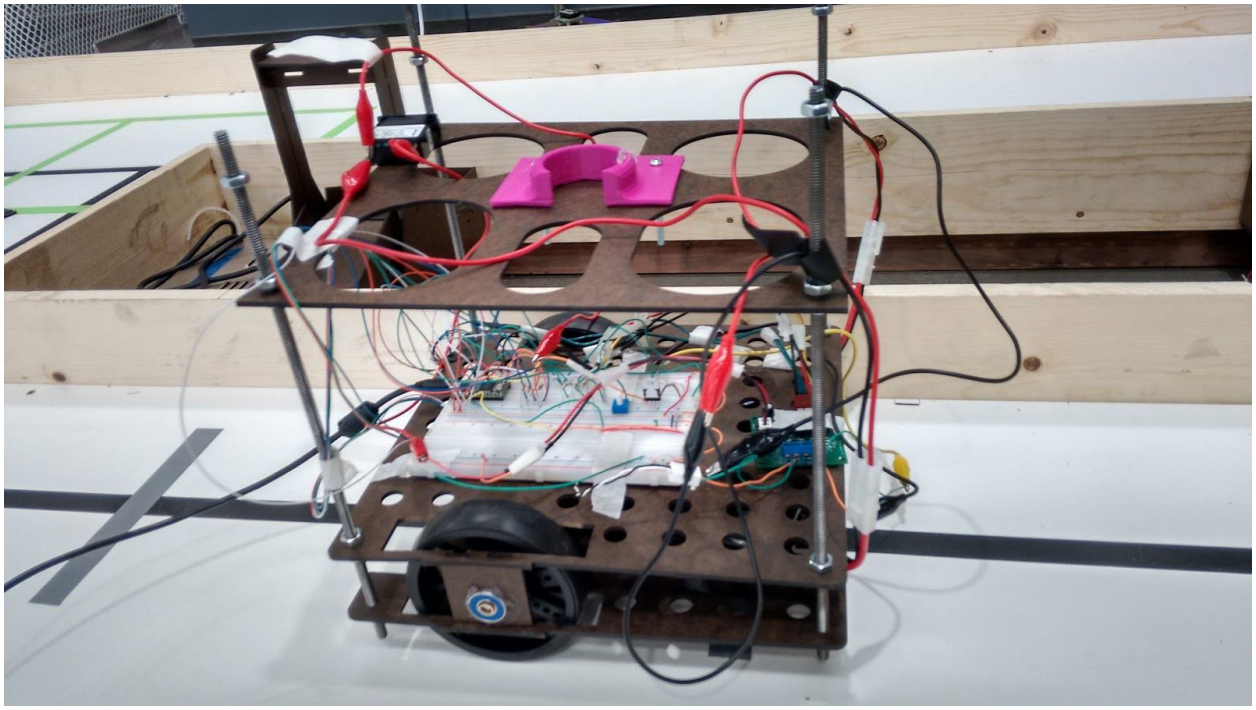
```
/*----------Module Variables----------*/
AccelStepper stepmotor(1, 3, 4); // pin 3 = step, pin 4 = direction
int distance = 10;

void Setup_Stepper(void) {
    stepmotor.setMaxSpeed(40);
}

void Move_Stepper_Motor(void) {
    stepmotor.move(distance);
}
void Run_Stepper_Motor(){
    stepmotor.run();
}
```

This link is to the GitHub repository for the software for this project.
https://github.com/srharris91/ME210_CowShots.git

4. Preliminary Test Results



The line sensors and motors work as intended. Initial testing was conducted on the line following capabilities of the robot. The robot oscillates and runs off the line soon after starting.

The beacon detectors have not been tested yet as the main focus currently is to achieve the check-off requirements.

5. Design modifications

5.1 Completing the design

We have implemented the ball dropping mechanism on our robot. Our first prototype was made with a plateau consisting of layers of pink foam surrounding the ball bearing, that we placed around a shaft. Spider couplers now make the link in between the shaft and rotating wheel.
Our final plateau design consisted of layers of laser-cut duron, pressfit around the bearing and enabling the slide to be inserted inside in a stable way. Nuts and bolts enable stability and placement of the upper plateau layer at an adjustable height.

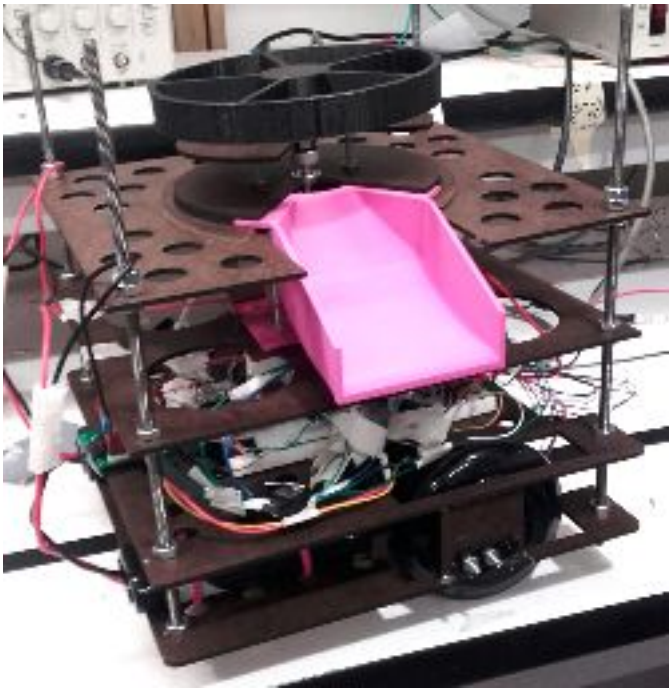
5.2 Issues encountered and solutions

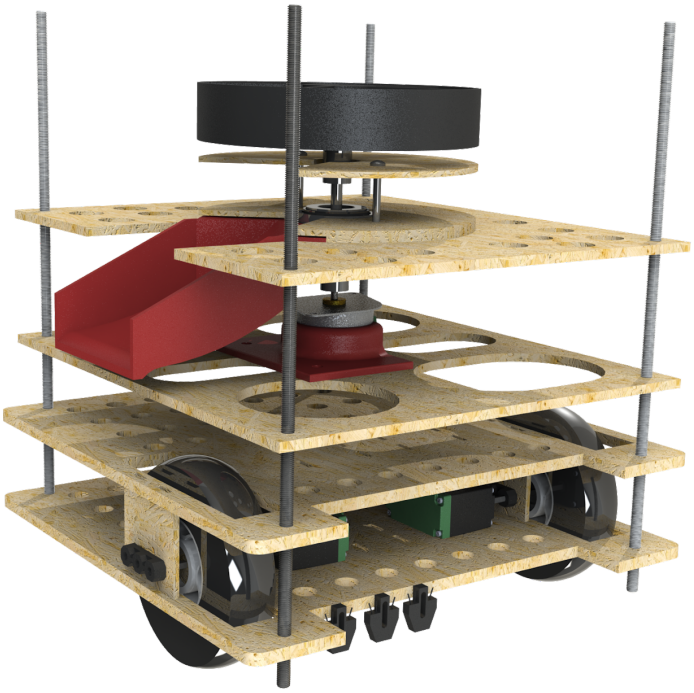
While constructing our robot, we have met the following issues which led to changes in our initial design:

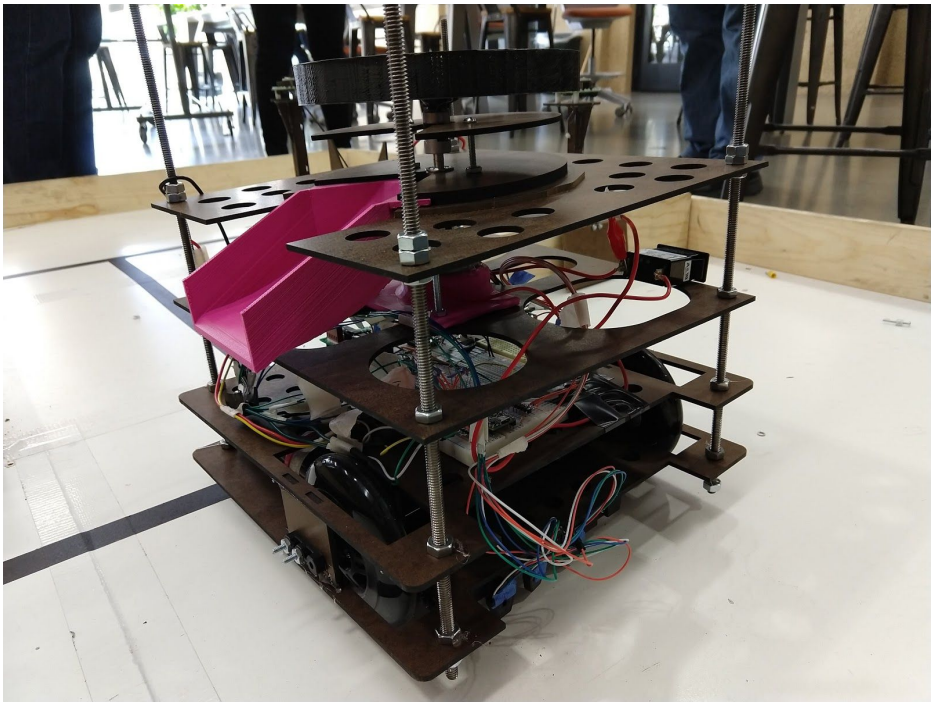
- Unstable electric wires: these would easily come off. We soldered most of our wires to headers to create a stronger link to the breadboard.
- Unable to tune the line following controller properly: the slow time response of the motors lead to amplifying oscillations. Moreover, slowing down the robot would have been tricky to implement since the initial speed required to counter friction forces is larger than the speed generating oscillations. Our main solution for this problem was to change the motors, and remove the spider couplers.
- Robot often banging into the wall (due to the oscillations). To take less risk, we moved the line sensors from centered to the right of the robot. We now move at a safe distance from the wall.
- Non repeatability of our robot’s behavior. In fact, our motor’s input voltage came straight from the batteries, and our PID tuning had to change with time. To solve this we added a 12V voltage regulator, which was immediately effective

5.3 Checkoff results

Solidworks CAD Renderings and Actual Robot







Our new robot design led to a repeatable and convenient behavior, which brought us to checkoff successfully.

Here is the link to watch our robot in action!

https://youtu.be/0EHYUA8m4A4