HUNGRY HUNGRY HIPPOS

Project Report

“The Cthulhu Bot”

Heena Agrawal, Rishabh Berlia, Louis Dankovich, Omkar Reddy

University of Colorado Boulder

MCEN-5115

INDEX

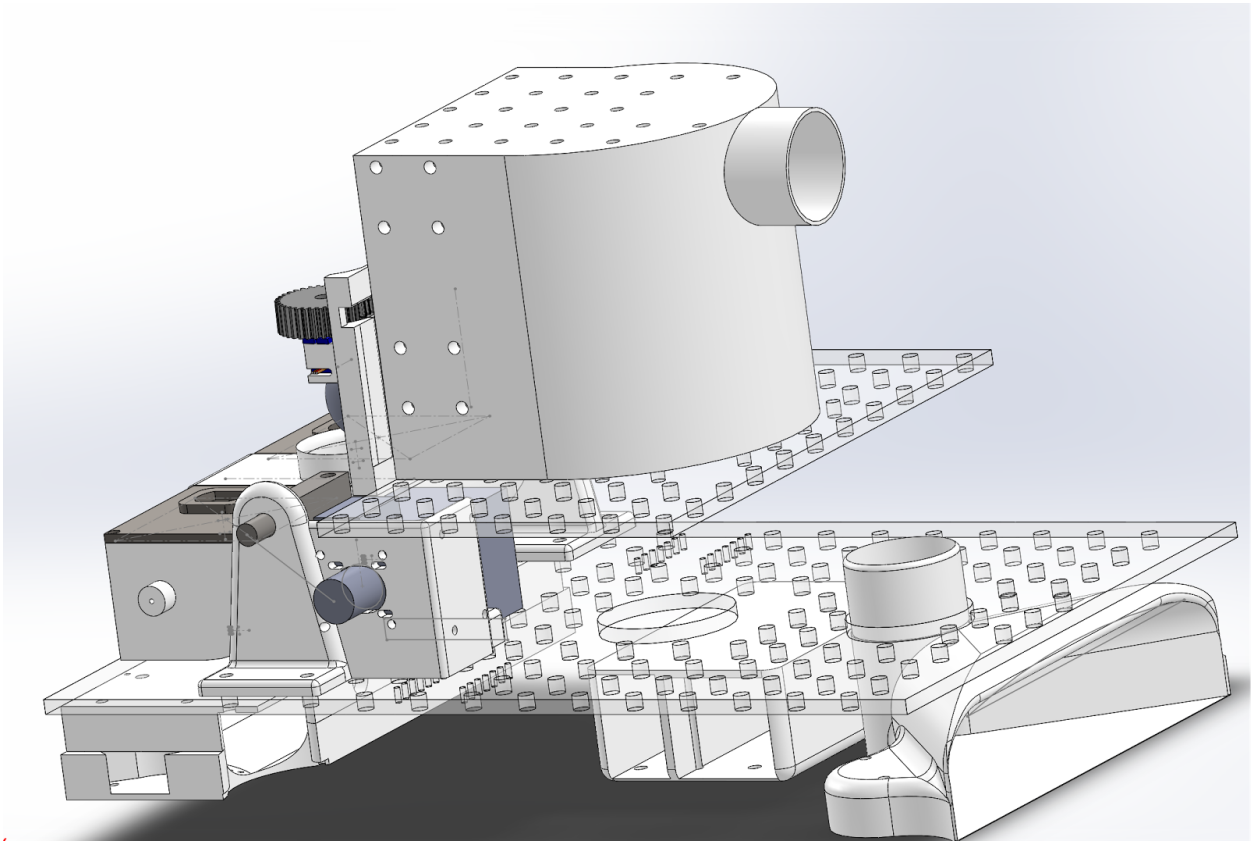
OBJECTIVE:

To design, build and manufacture a complete autonomous robot that can compete in the robot version of the game, Hungry Hungry Hippos.

MOTIVATION:

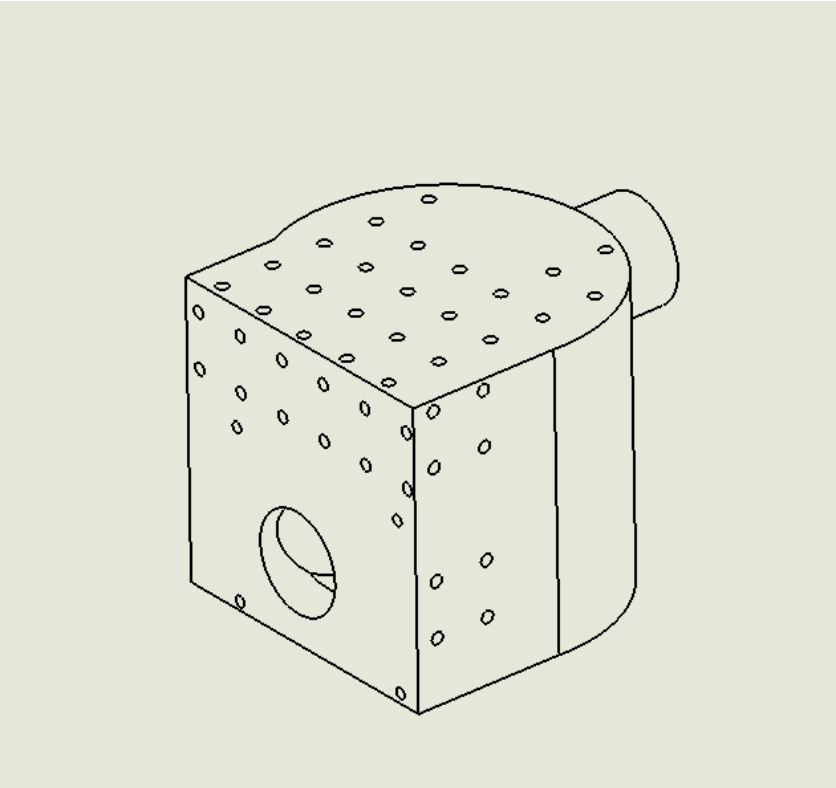
The name “Cthulhu Bot” has been inspired by a fictional cosmic entity created by writer and lunatic H.P. Lovecraft. Cthulhu was a half octopus, half man and half dragon from before time who inspired cultists. Late one night a team member was inspired to add USB tentacles to our robot by sleep deprivation. Or possibly by something that exists beyond time and sleeps in sunken *Ryleh*. *Aiai Ftaghan*.

SYSTEM OVERVIEW:



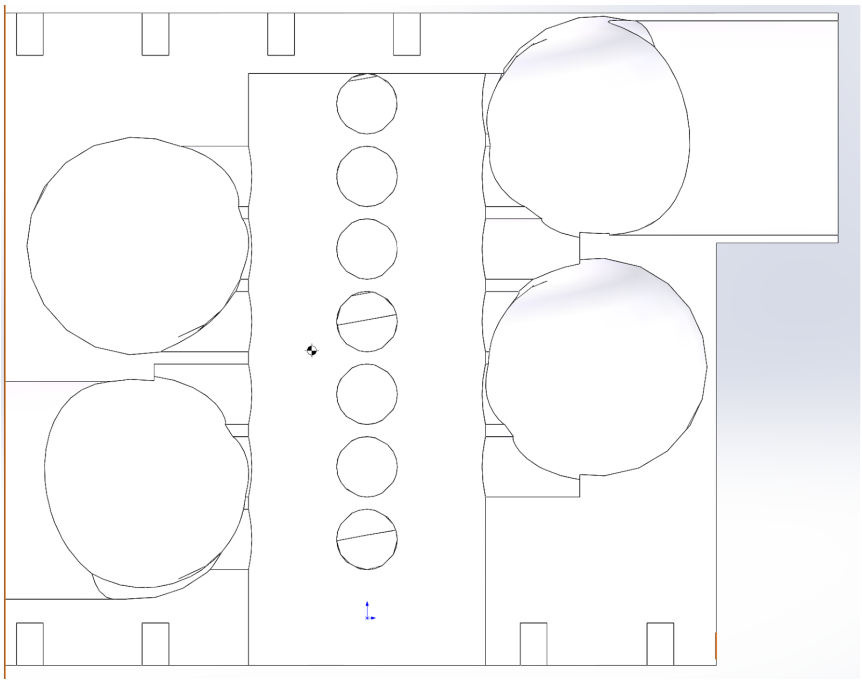
The entire project is divided into the following subsystems:

1.The Collector subsystem



Given the lack of machinists in the group, we focussed on utilizing 3d printers to create this device. Early in the project, a very inexpensive, battery powered vacuum was found at a thrift shop, pointing us towards a vacuum based collection system both due to price and easy availability.

The vacuum was a disassembled, and initially attached to pipes scavenged from the project depot. Unfortunately the pipes did not prove to be very efficient at maintaining vacuum or collecting balls. In the end tower unit with a helical manifold was designed and printed.

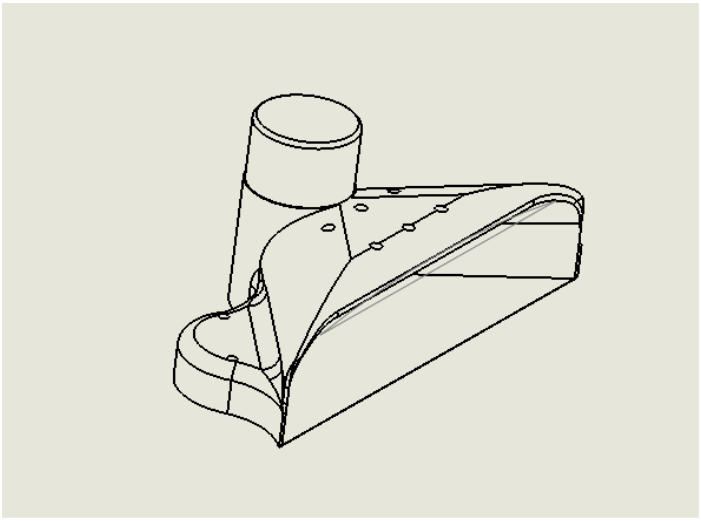


This manifold handled vacuum routing, ball collection/storage, and bolt on point for external boards and devices. The vacuum fan and motor was attached to the bottom of the tower using silicone caulk. It should be noted that the actual vacuum motor is not included in solidworks as it was an awkward shape to model, and so very rough dimensions of height and diameter were used in design planning rather than an actual model.

The front nozzle allows for attachment of an air hose to connect to a front collection unit. The hole in the rear allows for mounting of a vacuum gate. For clarity a cutaway view above shows the helix and central vacuum distribution network.

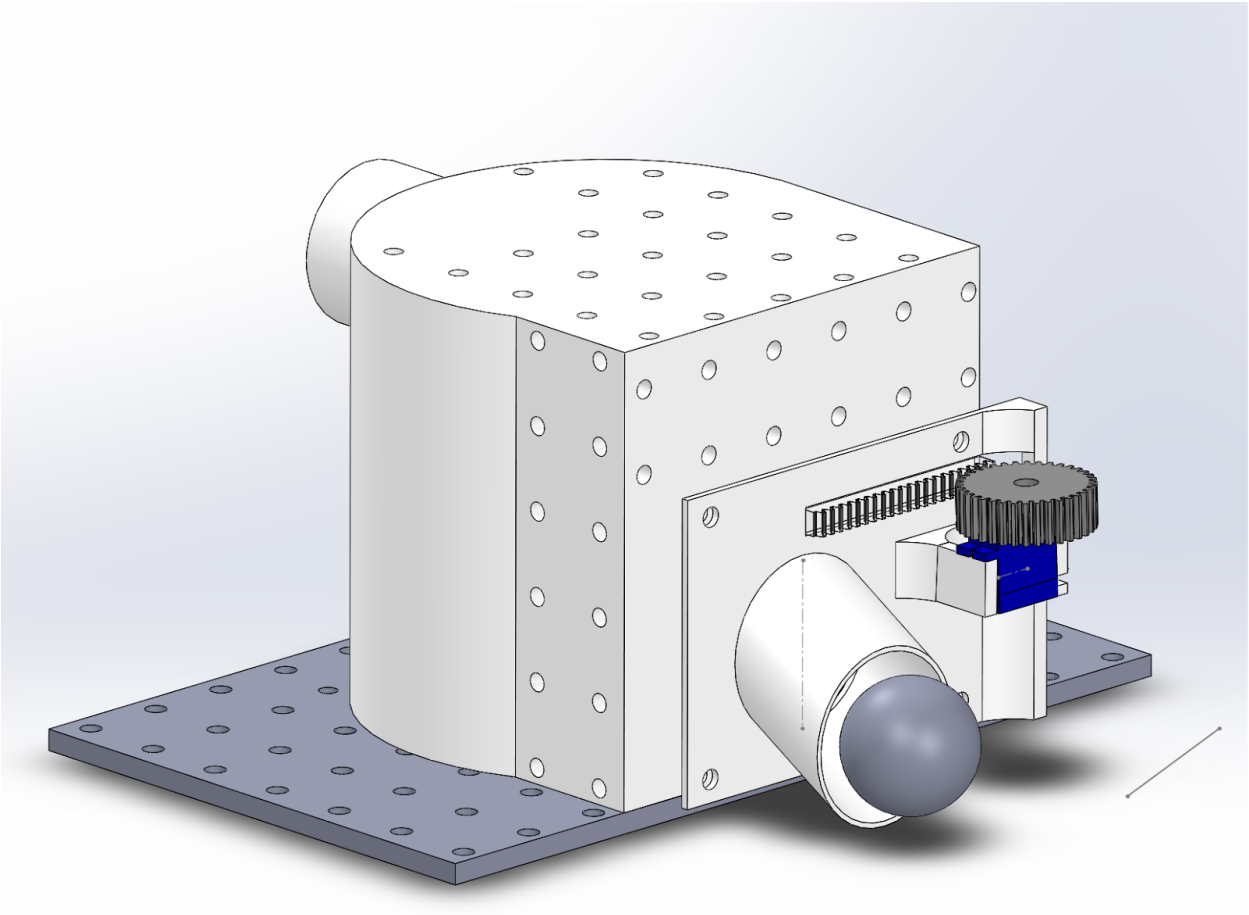
The large helical channel is for storage of collected balls. The central channel and smaller holes penetrating the sides of central channel are for vacuum distribution. Their size and number allow for vacuum to be distributed with minimal friction losses and insure that as balls are collected, many holes will remain uncovered to distribute vacuum.

In order to maximize efficiency of collections, a suction head modelled off that of a standard vacuum was placed in the front of robot. This suction head had a wide front angled back to a channel permitting a single ball to pass and a hose to be connected. This head was equipped with holes allowing a pair of casters to be placed so that it could roll freely on floor.



A significant use of the fillet feature in solidworks allowed for savings of material and print time in design of the vacuum head. It also lowered frictional losses in vacuum, and allowed for higher likelihood of successful prints by preventing sharp transitions.

The vacuum collector system was completed by a blast gate on rear of robot modelled off standard shop vacuum gates. It was bolted over rear opening on tower system. When gate was closed, smooth surfaces and strong vacuum allowed for efficient collection of balls. Actuated by a servomotor scavenged from the project depot and a rack and pinion track system, it could be opened when a payload of balls was collected. The angle of the downward spout allowed for balls to be gravity fed into the shooting mechanism subsystem with minimal intervention and no additional guidance tubes.



The entirety of vacuum system was mounted on a perforated acrylic platform with a 1” hole spacing pattern to allow for easy reconfiguration of system and mounting of additional part modules if needed.

Lucite cylinders (never modelled in solidworks) were cut, drilled and tapped on a late. These cylinders were used to mount the tower assembly over main platform of system. The systems main platform was initially made using perfboard, but proved both heavy and . For final iterations, acrylic sheet was cut on a laser cutter, while maintaining a similar hole pattern for convenience in mounting parts and routing wires.

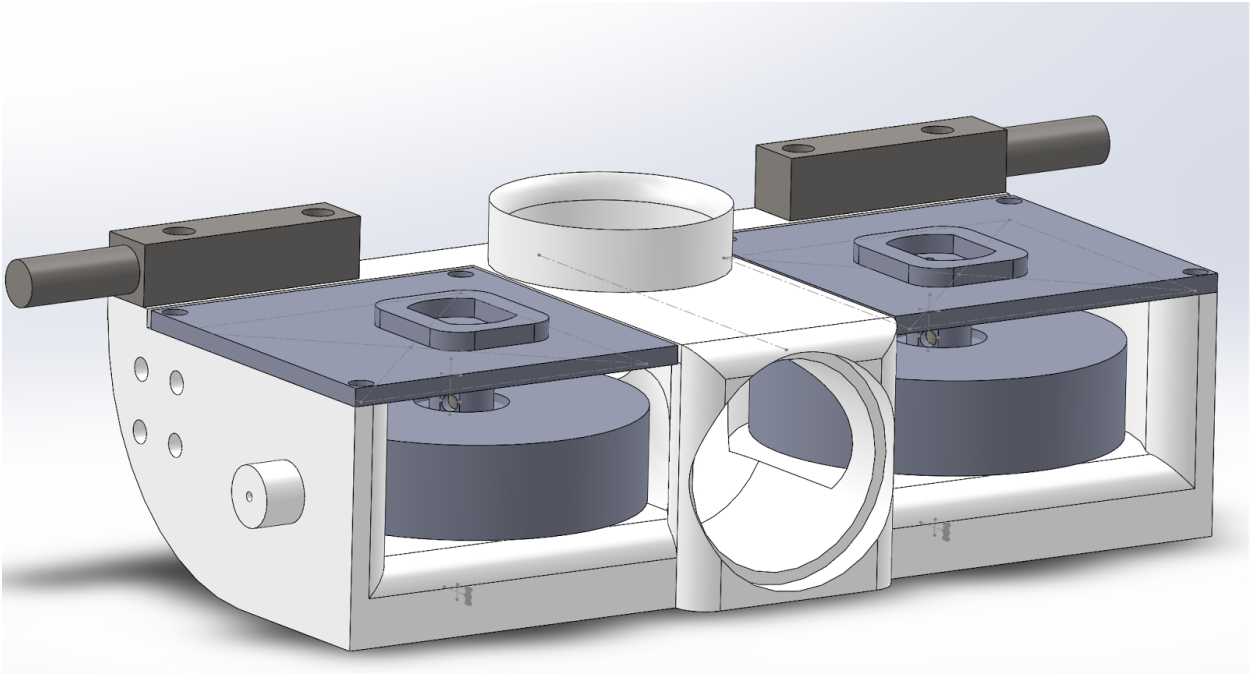
2. The Shooter Subsystem

One of the most common methods for firing ping pong balls in robotic contests and table tennis accessories is the flywheel mechanism. The operating principle is that a pair of spinning wheels will contact the ball. Assuming negligible deformation and good contact friction the flywheels accelerate the ball both linearly to the right and give it some counterclockwise rotation and linearly to the left, giving it clockwise rotation. When the interaction is over, the flywheel will lose some of its energy and angular momentum and

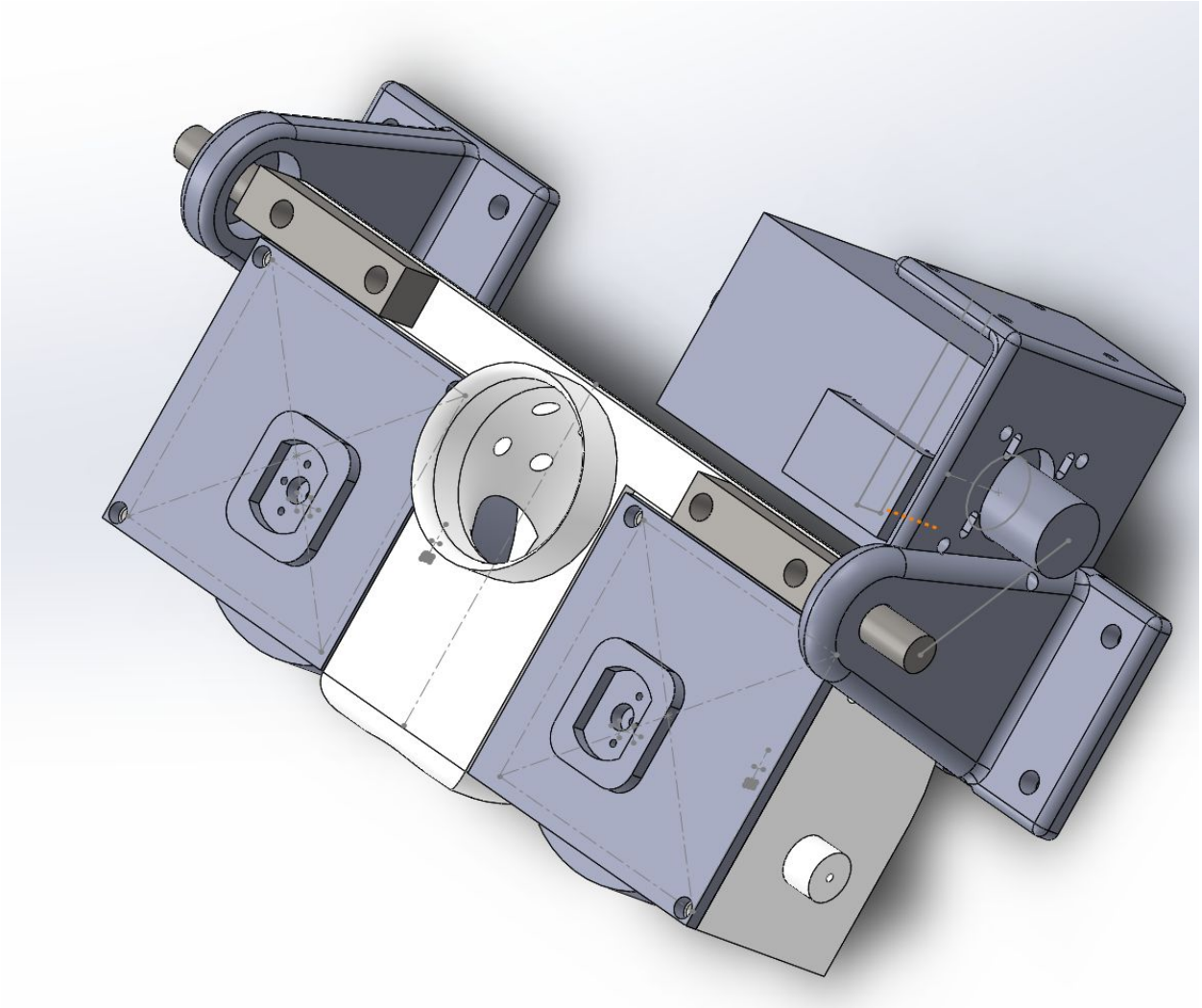
the ball will gain some linear and angular momentum. As a result the entire counter rotating flywheel mechanism imparts the ball enough force to follow a straight line trajectory.

The flywheels were round foam pieces scavenged from the ITLL Project Depot. They were mounted on a shaft, and bearings were used to support said shaft and minimize friction. A housing was printed using PLA and the design was completed with two metal plates for motor mounting.

After the initial print it was observed that the spaces for flywheels were not enough for the motors to sufficiently provide enough torque. Thus, a new design iteration was made with better spacing to provide enough room for the flywheels to rotate. The final design is shown below.



Earlier, it was thought that the shooter assembly could be made to align itself in straight line with the hoop, by providing it certain angles using a stepper motor with a belt drive mechanism. Though this design was incorporated, it could not be effectively used in the final demonstration. Therefore, at the very last moment, the shooter was super glued at an angle that directly points towards the centre of the hoop.



3. The Drive Subsystem

Two stepper motors (each equipped with a 4” diameter omniwheel) form the drive system. Initial designs had called for larger wheels, and more wheels to overcome the challenges of the proposed ⅜” foam barrier. However when it was eliminated from proposed arena our team was able to scale back drive requirements. A pair of casters under the suction head and another pair of casters under the circuitboard case in rear reduced friction and potential for robot to tip due to imbalances.

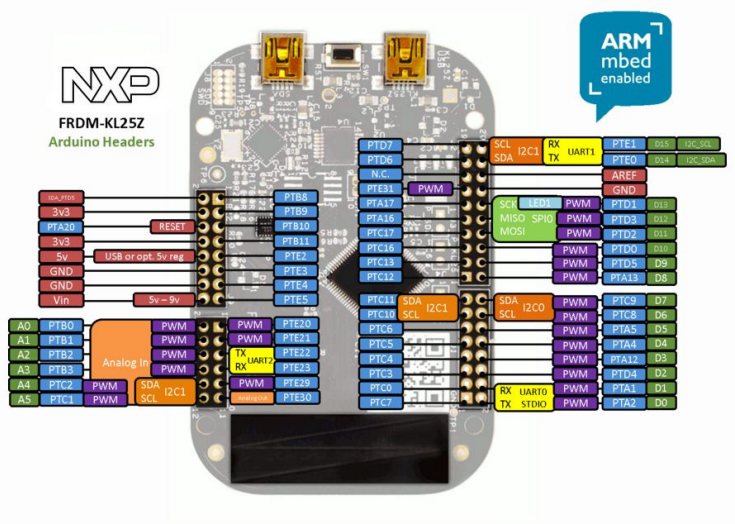
The structure of the robot consists of an acrylic sheet mounted with wheels and casters. The shooter is mounted in the front with vacuum sweeper at the back. The front of the bot also consists of a power board enclosed in a 3D printed casing with 2 casters at the bottom. The back of the robot has vacuum sweeper with bottom mounts for the casters. In the middle of the acrylic sheet, there are 4 stands holding up another acrylic plate,

where the collector unit is mounted. Above the collector unit stands the rotating vision system circuit. The bottom of the bot, also has a 3D printed case to hold the batteries.

4. The Electrical Subsystem

- Components Used
    1. Freedom KL64F development board
    2. 2 High Speed DC Brushless Motors (for shooter)
    3. Vacuum Motor
    4. Stepper Motor(BJ28Y) and driver (ULN2003) (For Vision System)
    5. 12V DC Motors with encoder (Drive system)
    6. 2 L293D motor driver (Shooter and Drive system)
    7. 3D Filament Rolls
    8. Circuit for the Vision System
    9. Polulu md07a motor driver (for the Vacuum Switch)
    10. USB Powered Tentacles.

The initial design called for use of a Raspberry Pi 2 with camera interface, but this resulted in very inconsistent image processing and had to be scrapped. In its place as simpler system using a single IR Receiver Transistor communicating with the Freescale Mbed was implemented.



Other electronic subsystems were designed individually during build of robot, and code was written/ validated on each individual system before being combined in system integration step. This saved a lot of time in final assembly of robot, and allowed for flaws in code to be found early in process.

A problem that did crop up in the final stages of design however was power supply overheating due to the current draw of the vacuum fan and drive wheels. In order to mitigate this an enclosure was designed and 12 Volt fan mounted on it to drive air across the heat sinks.

However the biggest challenge was the vision system. Unbeknownst to the team, the PCB we had etched to communicate with it was damaged in the final stages of system

integration. This gave us a significant amount of signal noise and false positives detected randomly even when there was no IR light in the vicinity. Significant time was lost in determining that root cause was not software but hardware based. System also suffered from motion artifacts when it was moved from stationary test bed to robot. The team was forced to design a new vision system in the final days of project, and did not have adequate time for proper calibration.

Another significant challenge was the shooting system. Initially it was meant to have a variable angle controlled by a stepper motor so that it could shoot from any range and aim at smaller or larger target depending on whether central tower was blocking smaller target. Unfortunately this stepper motor required significant current, more current in fact than the motor drivers available to team could provide.This resulted in the motor driver burning out the night before competition. The team was then forced to fix shooter angle and change code to accommodate fixed angle.

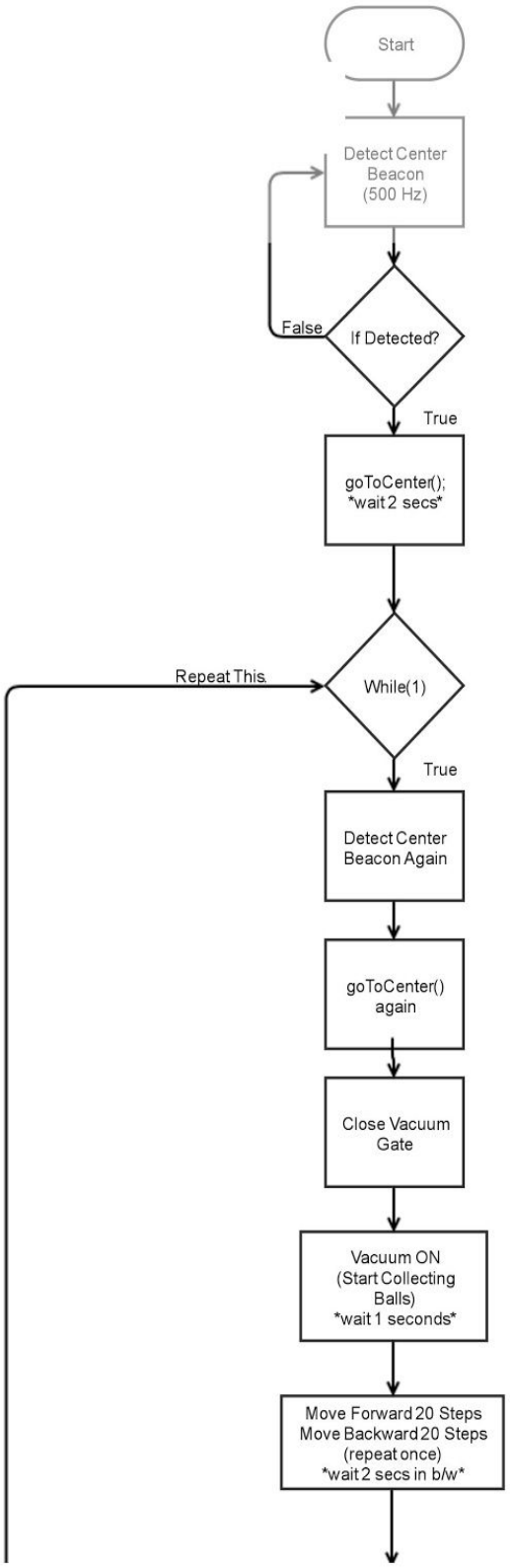
The third significant issue was the vacuum motor. A cheap (but powerful) handheld vacuum was sourced at a thrift shop. To satisfy our design needs we needed to disassemble it and integrate its motor into the robot. The motor however kept blowing the fuse we placed even at highest possible amperage of the Polulu Motor driver. It was later that Professor Reamon told us that the current spikes in such heavy motors could be a lot in the beginning. So as per his suggestion we moved to a higher amperage fuse and things worked out just fine.

5. The Software Subsystem

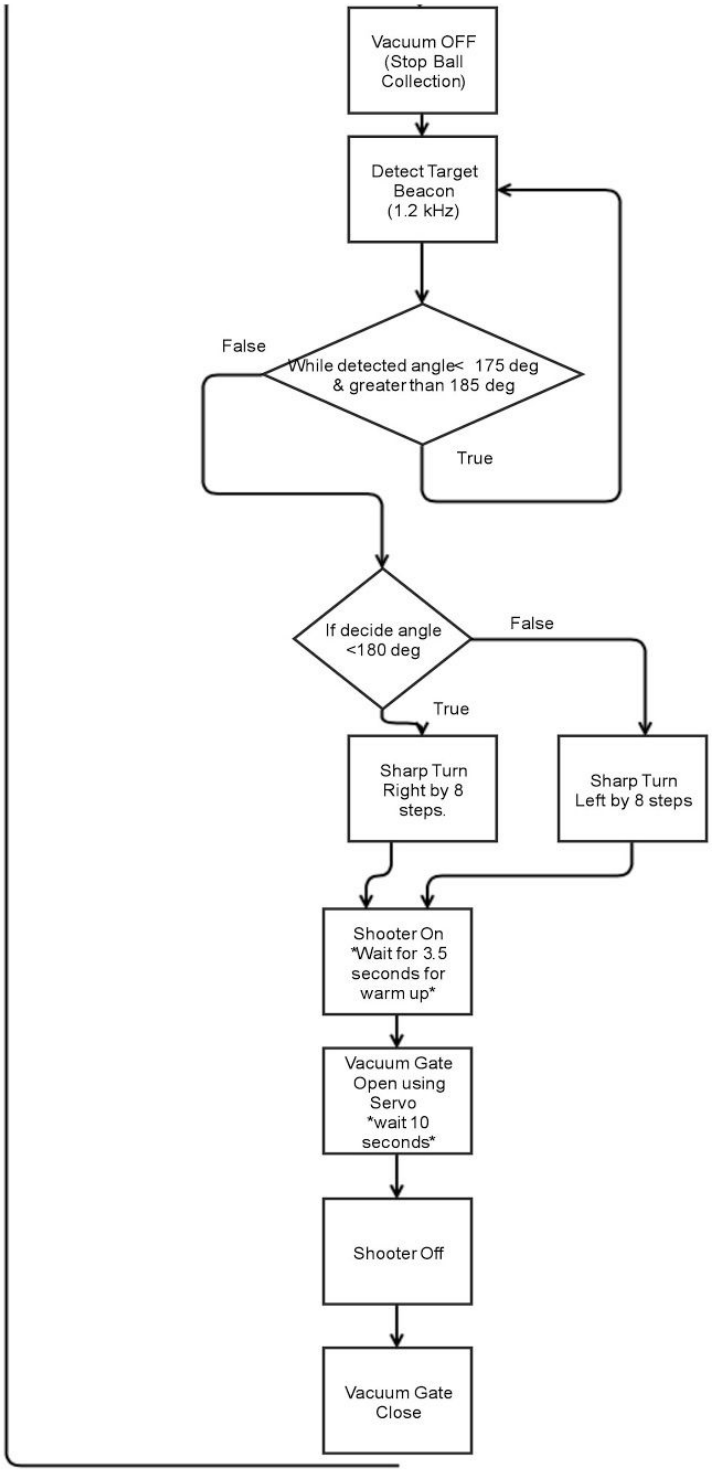
The software development was done using the mbed's online IDE and compiler. mbed has a great development IDE for a wide range of embedded microcontrollers and provides easy-to-use libraries and APIs.

The initial code flow has functions for each of the subsystems and test codes to test each of the individual subsystem. The various individual subsystems involved in the software flow are: Drive system, Shooter System, Vacuum suction system, Vacuum gate system, Vision/Beacon system and the Line follower system.

Flow Chart for system is shown on next page.


Start
Detect Center Beacon (500 Hz)
If Detected?
False
True
goToCenter(); *wait 2 secs*
While(1)
Repeat This
True
Detect Center Beacon Again
goToCenter() again
Close Vacuum Gate
Vacuum ON (Start Collecting Balls) *wait 1 seconds*
Move Forward 20 Steps
Move Backward 20 Steps
(repeat once)
wait 2 secs in b/w

at 20:51:51


Vacuum OFF (Stop Ball Collection)
Detect Target Beacon (1.2 kHz)
False
While detected angle< 175 deg & greater than 185 deg
True
If decide angle <180 deg
False
True
Sharp Turn Right by 8 steps.
Sharp Turn Left by 8 steps
Shooter On *Wait for 3.5 seconds for warm up*
Vacuum Gate Open using Servo *wait 10 seconds*
Shooter Off
Vacuum Gate Close

The most significant programming challenge was converting between integers and floats. It had been assumed that the x,y coordinates we provided would be implicitly converted to integers, but this did not prove to be the case. For example, we had a x y values of 260 and 360. Dividing should have given us an output of 0.72 in radians but the function was passing 0. After considerable frustration, we concluded that mbed compiler does not support implicit conversion. So we had to provide the float value directly.

Cost Report

Component Used	Quantity	Cost
PLA Filament	4	$ 100
Stepper Motors	2	$10
Omni Wheels	2	(project depot)
Casters	4	(project depot)
Acrylic Sheet	1	$20
DC Motors	2	(Old Projects)
Servo Motor	1	(project depot)
Jumper Wires Set	3	$15
Dust Devil Vacuum	1	$13
Raspberry Pi 3	1	$35
Thermal Paste	1	$5
Heat Sink Fan	1	$6
Heat Sinks	1 packet	$8
Tentacles	4	$20
Total		**$232**

Lessons Learned

This project has been a valuable learning experience and has taught a lot of short hand tricks and long term design challenges required in a real time project. Some of the lessons learned are:

- Creating a component or an entire robot is an iterative process, that needs work everyday. So, procrastinating is the worst practice.
- The better you prototype the design, the less you have to change in the actual product.
- 3D printing is a double edged sword. While it can save significant time with appropriate design (and luck) it can also cost a lot of time if the printer jams mid job or if there are flaws in the solidworks model.
- Proper planning and a lucrative thought process is very important.
- Just making the robot is not enough. The robot requires testing and iterative programing in the intended context to work out the bugs.
- It is inevitable that loose wires will be disconnected. Once initial design is concluded, fixing location of microcontroller and boards is critical so that wires can be routed through system.
- We learned that you should question every assumption of yours if stuck in an unexplainable situation. There is always an explanation, you just need to look at the right place.
- The most rookie and time costing mistake that we made was to connect the GND and VCC pins otherway. This blew up our mbed and a few motor drivers. We had to set up all this again and this cost a lot of time.
- We learnt that scavenging parts is a good idea but you need to identify the parts before hand and discard them if they do not fit your needs.
- It’s okay to make mistakes, that how we learn and hopefully the future semester students can learn from them.

Conclusion

Although the subsystems worked well, the “Cthulhu Bot” did not work as effectively as anticipated. The design process was challenging but led to and aesthetically pleasing and mostly functional end product. More time however was needed to refine the code used for sensing the beacon and orienting robot. Nonetheless it has given us the chance to learn from our mistakes for all the future projects we will be working on.

Acknowledgements

We would like to thank Dr. Derek Reamon, Our TA's Michael Ohradzansky and Dylan Highland.

We would extend our gratitude to the ITLL staff including Kai, Timothy Mayes and other student staff members for their valuable knowledge, great help, support and experiences that made the design and manufacture of this robot possible.

Appendix

```
// ECEN 5115 - Mechatronics and Robotics
// Omkar Reddy, Rishabh Berlia, Heena R Aggarwal, Louis Dankovich
// M1 - Right Motor; M2 - Left Motor; With Suction in the front

// 88 steps for complete revolution for each wheel
// For sharp_angle : 260 steps for 360 degrees

// Libraries
#include "mbed.h"
#include "InterruptIn.h"
#include "device.h"

// Global Variables
Serial pc(USBTX, USBRX);

// Macro Definitions
// Drive System; Motors
#define bot360_steps           260      // Full bot rotation; Sharp turns
#define wheel360_steps         88       // Steps for full wheel revolution
// Vaccuum Gate Servo
#define V_Gate_Period_ms        20      //PWM Period
#define V_Gate_Open_width_ms    1.0     //Pulse width for Opening gate
#define V_Gate_Close_width_ms   3.0     //Pulse width for Closing gate
// Beacon Detector
#define beaconDetectRotateAngle 10
#define BEACON_FREQ_TEST        1200
#define BEACON_BW_TEST          100
#define BEACON_FREQ_CENTER      500
#define BEACON_BW_CENTER        20
#define BEACON_FREQ_RED         1200
#define BEACON_BW_RED           100
#define BEACON_FREQ_BLUE        3000
#define BEACON_BW_BLUE          200
#define BEACON_SCANINTERVAL_MS  3
#define BEACON_SCANINTERVAL_S   0.003
// GPIO Pin Declarations
#define Motor1_A                PTB23           //Motor 1 Pin A
#define Motor1_B                PTA2            //Motor 1 Pin B
#define Motor2_A                PTC2            //Motor 2 Pin A
#define Motor2_B                PTC3            //Motor 2 Pin B
#define Motor_Encoder           PTB9

#define Vaccuum_Switch          PTD1
#define Vaccuum_Gate            PTA1

#define Shooter1_A              PTC16           // Shooter Motor 1 Pin A
#define Shooter1_B              PTC17           // Shooter Motor 1 Pin B
#define Shooter2_A              PTE24           // Shooter Motor 2 Pin A
#define Shooter2_B              PTE25           // Shooter Motor 2 Pin A

#define Beacon                  PTC4            // Beacon Pin Input

#define TapeFollower_left       PTD0            // Tape Follower Left Input
#define TapeFollower_right      PTD3            // Tape Follower Right Input

#define Stepper_A               PTB18           // I/P 1 Stepper
#define Stepper_B               PTB19           // I/P 2 Stepper
#define Stepper_C               PTC1            // I/P 3 Stepper
#define Stepper_D               PTC8            // I/P 4 Stepper


// Global Variables and Pin Setup
// Motor
DigitalOut M1_A(Motor1_A);
DigitalOut M1_B(Motor1_B);
```

```
DigitalOut M2_A(Motor2_A);
DigitalOut M2_B(Motor2_B);
InterruptIn Encoder(Motor_Encoder);
int count = 0;                          // Initializing the count to 0
int requiredSteps = 0;
bool count_done = false;

// Vaccuum Switch
PwmOut V_Switch(Vaccuum_Switch);

// Vaccuum Gate Servo
PwmOut V_Gate(Vaccuum_Gate);

// Shooter
DigitalOut S1_A(Shooter1_A);
DigitalOut S1_B(Shooter1_B);
DigitalOut S2_A(Shooter2_A);
DigitalOut S2_B(Shooter2_B);

// Beacon Detector
DigitalOut led_RED(LED_RED);
DigitalOut led_GREEN(LED_GREEN);
InterruptIn beaconDetector(Beacon);
int detectedFreq;

// Stepper for Beacon
DigitalOut stepperA(Stepper_A);
DigitalOut stepperB(Stepper_B);
DigitalOut stepperC(Stepper_C);
DigitalOut stepperD(Stepper_D);

// Line follower
InterruptIn linefollower_left(TapeFollower_left);
InterruptIn linefollower_right(TapeFollower_right);
bool BLACK_DETECTED = false;
bool BLACK_ON_LEFT = false;
bool BLACK_ON_RIGHT = false;

// Function to stop the Drive System (Stop all motors)
void stop(){
    M1_A = 0;
    M1_B = 0;
    M2_A = 0;
    M2_B = 0;
}
// Count for encoder
void triggerCount(){
    if (count <= requiredSteps)
        count++;        // increment the counter and count number of falling edges; because
        frequency is nothing but a count
    else{
        stop();
        count = 0;
        //Encoder.fall(NULL);
    }
}
//Turn On Encoder and start counting pulses
void motorEncoderOn(){
    Encoder.fall(&triggerCount);
}
//Move motor forward by given number of steps.
void forward(int steps){
    requiredSteps = steps;
    motorEncoderOn();
    // M1 clockwise
    M1_A = 1;
```

```
    M1_B = 0;
    // M2 anticlockwise
    M2_A = 0;
    M2_B = 1;
}
//Move motor backward by given number of steps.
void backward(int steps){
    requiredSteps = steps;
    motorEncoderOn();
    // M1 anticlockwise
    M1_A = 0;
    M1_B = 1;
    // M2 clockwise
    M2_A = 1;
    M2_B = 0;
}
// Sharp left - Move both left and right wheels to move left, the movement is about the center
of the bot.
void sharp_left(int angle){
    requiredSteps = angle * 0.72;
    motorEncoderOn();
    // M1 clockwise
    M1_A = 1;
    M1_B = 0;
    // M2 clockwise
    M2_A = 1;
    M2_B = 0;
}
// Sharp right - Move both left and right wheels to move right, the movement is about the
center of the bot.
void sharp_right(int angle){
    requiredSteps = angle * 0.72;
    motorEncoderOn();
    // M1 anticlockwise
    M1_A = 0;
    M1_B = 1;
    // M2 anticlockwise
    M2_A = 0;
    M2_B = 1;
}
//Test function to test the drive system.
void driveSystemTest(){
    forward(50);
    wait(1);
    backward(50);
    wait(1);
    sharp_left(50);
    wait(1);
    sharp_right(50);
    wait(1);
}

// Vaccuum Switch Routines
// Test Routine
void vaccuumSwitchTest(){
    wait(2);
    V_Switch = 0.5f;
    wait(7);
    V_Switch = 0.0f;
    /*V_Switch = 1;
     wait(3);
     V_Switch = 0;
     wait(3);*/
}
// Vaccuum On with 50% duty Cycle
void vaccuumOn(){
```

```
    wait(1);
    V_Switch = 0.5f;
}
// Vaccuum Off with 0% duty cycle
void vaccuumOff(){
    wait(1);
    V_Switch = 0.0f;
}

// Vaccuum Gate Routines
// Test Routine
void vaccuumGateTest(){
    static bool pwmSet = true;
    if (pwmSet)
        V_Gate.period_ms(V_Gate_Period_ms);
    V_Gate.pulsewidth_ms(V_Gate_Open_width_ms);
    wait(3);
    V_Gate.pulsewidth_ms(V_Gate_Close_width_ms);
    wait(3);
    pwmSet = false;
}
// Setup the PWM of the Vaccuum Gate Servo
// Open the Vaccuum Gate
void vaccuumGateOpen(){
    V_Gate.pulsewidth_ms(V_Gate_Open_width_ms);
}
// Close the Vaccuum Gate
void vaccuumGateClose(){
    V_Gate.pulsewidth_ms(V_Gate_Close_width_ms);
}
void vaccuumGateSetup(){
    V_Gate.period_ms(V_Gate_Period_ms);
    vaccuumGateClose();
}

// Shooter Routines
// Test Routine
void shooterTest(){
    S1_A = 1;
    S1_B = 0;
    S2_A = 1;
    S2_B = 0;
    wait(15);
    S1_A = 0;
    S1_B = 0;
    S2_A = 0;
    S2_B = 0;
    wait(5);
}
// Shooter On
void shooterOn(){
    S1_A = 1;
    S1_B = 0;
    S2_A = 1;
    S2_B = 0;
}
// Shooter Off
void shooterOff(){
    S1_A = 0;
    S1_B = 0;
    S2_A = 0;
    S2_B = 0;
}
//Stepper Rotate Back
void rotateBack(int steps){
    int seq_r[8][4] = {
```

```
        {1,0,0,1},
        {0,0,0,1},
        {0,0,1,1},
        {0,0,1,0},
        {0,1,1,0},
        {0,1,0,0},
        {1,1,0,0},
        {1,0,0,0}
    };
    int i,j,k;
    // Rotate back
    for (k = 0; k < steps; k++){
        for (i = 0; i < 8 ; i++){
            j = 0;
            stepperA = seq_r[i][j];
            stepperB = seq_r[i][j+1];
            stepperC = seq_r[i][j+2];
            stepperD = seq_r[i][j+3];
            wait_ms(1);
        }
    }
    // End
    stepperA = 0;
    stepperB = 0;
    stepperC = 0;
    stepperD = 0;
}
// Rotate stepper in given direction
void rotateStepper(int steps, bool direction){  // direction = true => clockwise(seq_f);
direction = false => Counter-clockwise(seq_r);
    int seq_f[8][4] = {
        {1,0,0,0},
        {1,1,0,0},
        {0,1,0,0},
        {0,1,1,0},
        {0,0,1,0},
        {0,0,1,1},
        {0,0,0,1},
        {1,0,0,1}
    };
    int seq_r[8][4] = {
        {1,0,0,1},
        {0,0,0,1},
        {0,0,1,1},
        {0,0,1,0},
        {0,1,1,0},
        {0,1,0,0},
        {1,1,0,0},
        {1,0,0,0}
    };
    int i,j,k;
    if (direction){
        // Rotate clockwise
        for (k = 0; k < steps; k++){
            for (i = 0; i < 8 ; i++){
                j = 0;
                stepperA = seq_f[i][j];
                stepperB = seq_f[i][j+1];
                stepperC = seq_f[i][j+2];
                stepperD = seq_f[i][j+3];
                wait_ms(1);
            }
        }
    }
    else{
        // Rotate counter-clockwise
```

```
        for (k = 0; k < steps; k++){
            for (i = 0; i < 8 ; i++){
                j = 0;
                stepperD = seq_f[i][j];
                stepperC = seq_f[i][j+1];
                stepperB = seq_f[i][j+2];
                stepperA = seq_f[i][j+3];
                wait_ms(1);
            }
        }
    }
}
//Increase count (detectedFreq) everytime a falling edge occurs to determine Beacon frequency
void beaconIRQ(){
    detectedFreq++;
}
//Beacon Routine
int detectBeacon(int frequency, int bandwidth){
    int angleFound = 0, countSteps = 0;
    bool direction;
    int upperCutoff = 0, lowerCutoff = 0;
    upperCutoff = BEACON_SCANINTERVAL_S * (frequency + (bandwidth/2));
    lowerCutoff = BEACON_SCANINTERVAL_S * (frequency - (bandwidth/2));

    // Rotate and Scan
    while(1){
        // Scan the beacon
        detectedFreq = 0;
        beaconDetector.fall(&beaconIRQ);
        // check if beacon detected
        wait(BEACON_SCANINTERVAL_S);/*wait_ms(BEACON_SCANINTERVAL_MS);*/
        beaconDetector.fall(NULL);
        if((detectedFreq >= lowerCutoff) && (detectedFreq <= upperCutoff)){
            // return the countSteps angle in degrees; For the Robot to rotate
            break;
        }
        // If Beacon not detected in current direction; Rotate the Beacon
        if (countSteps >= 512){
            direction = false;
            wait(1);
            rotateStepper(4, direction);
        }
        else if (countSteps <= 0){
            direction = true;
            wait(1);
            rotateStepper(4, direction);
        }
        else{
            rotateStepper(4, direction);
        }
        if (direction)
            countSteps += 4;
        else
            countSteps -= 4;
    }
    angleFound = 0.72 * countSteps;
    rotateBack(countSteps);
    if (angleFound >= 180){
        led_GREEN = 0;
        led_RED = 1;
    }
    else {
        led_GREEN = 1;
        led_RED = 0;
    }
    return angleFound;
```

```
}

//Tape Follower Routine
void blackDetected(){
    linefollower_left.fall(NULL);
    linefollower_right.fall(NULL);
    stop();
    BLACK_DETECTED = true;
}
// After center is detected, switch on the IR Sensors to detect the center tape and move
forward towards the detected center.
void goToCenter(){
    linefollower_right.fall(&blackDetected);
    linefollower_left.fall(&blackDetected);
    BLACK_DETECTED = false;
    while(!BLACK_DETECTED){
        forward(10);
    }
}
//Right at an angle of 90 deg
void alignWithCircle(){
    sharp_right(90);
}

//Stepper Test Routine
void stepperTest(int steps){
    int seq_f[8][4] = {
        {1,0,0,0},
        {1,1,0,0},
        {0,1,0,0},
        {0,1,1,0},
        {0,0,1,0},
        {0,0,1,1},
        {0,0,0,1},
        {1,0,0,1}
    };
    int seq_r[8][4] = {
        {1,0,0,1},
        {0,0,0,1},
        {0,0,1,1},
        {0,0,1,0},
        {0,1,1,0},
        {0,1,0,0},
        {1,1,0,0},
        {1,0,0,0}
    };
    int i,j,k;
    for (k = 0; k < steps; k++){
        for (i = 0; i < 8; i++){
            j = 0;
            stepperA = seq_f[i][j];
            stepperB = seq_f[i][j+1];
            stepperC = seq_f[i][j+2];
            stepperD = seq_f[i][j+3];
            wait_ms(1);
        }
    }
    // Rotate back
    for (k = 0; k < steps; k++){
        for (i = 0; i < 8; i++){
            j = 0;
            stepperA = seq_r[i][j];
            stepperB = seq_r[i][j+1];
            stepperC = seq_r[i][j+2];
            stepperD = seq_r[i][j+3];
            wait_ms(1);
```

```
        }
    }
    // End
    stepperA = 0;
    stepperB = 0;
    stepperC = 0;
    stepperD = 0;
}


void alignTowardsBeacon(int angle, bool direction){
    if (direction){
        if (angle <= 180){
            sharp_right(angle);
        }
        else if ((angle > 180) && (angle <= 360)){
            angle = 360 - angle;
            sharp_left(angle);
        }
    }
    else {
        if (angle <= 180) {
            //angle = angle + 5;
            angle = 180 - angle;
            sharp_left(angle);
        }
        else {
            //angle = angle - 5;
            angle = angle - 180;
            sharp_right(angle);
        }
    }
}

//Find the Beacon and Face it
void findAndFaceBeaconTest(){
    int beaconDirection_angle;
    beaconDirection_angle = detectBeacon(BEACON_FREQ_TEST, BEACON_BW_TEST);
    alignTowardsBeacon(beaconDirection_angle, true);
    //forward(100);
}

//Main Robot Routine
void hippobotamus(){
    int beaconDirection_angle = 0;
    // Read Current State; RED or BLUE
    beaconDirection_angle = detectBeacon(BEACON_FREQ_CENTER, BEACON_BW_CENTER);     // Find
    where the beacon is
    alignTowardsBeacon(beaconDirection_angle, true);
    beaconDirection_angle = detectBeacon(BEACON_FREQ_CENTER, BEACON_BW_CENTER);     // Find
    where the beacon is
    if (beaconDirection_angle > 10) {
        alignTowardsBeacon(beaconDirection_angle, true);
        beaconDirection_angle = detectBeacon(BEACON_FREQ_CENTER, BEACON_BW_CENTER);
    }
    goToCenter();
    // move forward
    wait(2);
    while (1){
        backward(50);
        beaconDirection_angle = detectBeacon(BEACON_FREQ_CENTER, BEACON_BW_CENTER);     // To
        make sure that the bot is facing correct direction
        while ((beaconDirection_angle > 5) && (beaconDirection_angle < 355)) {
            alignTowardsBeacon(beaconDirection_angle, true);
            beaconDirection_angle = detectBeacon(BEACON_FREQ_CENTER, BEACON_BW_CENTER);
        }
```

```
        goToCenter();
        vaccuumGateClose();                                                    // Make
        sure that the gate is closed
        vaccuumOn();
        wait(1);
        forward(20);                                                           // Turn
        vaccuum on; Suck the balls in;
        wait(2);
        backward(20);
        wait(1);
        forward(20);                                                           // Turn
        vaccuum on; Suck the balls in;
        wait(2);
        backward(20);
        vaccuumOff();                                                          // Turn
        off vaccuum
        beaconDirection_angle = detectBeacon(BEACON_FREQ_RED, BEACON_BW_RED);     // detect
        Target Beacon                                    // align with Target Beacon
        while ((beaconDirection_angle < 175) | (beaconDirection_angle > 185)) {
                                                 // It is for the target; So its on the other
        side of the front; So, 90-10
            alignTowardsBeacon(beaconDirection_angle, false);
            beaconDirection_angle = detectBeacon(BEACON_FREQ_RED, BEACON_BW_RED);
        }
        if(beaconDirection_angle <= 180)
            sharp_left(8);
        else
            sharp_right(8);
        // Set aim for shooter
        shooterOn();
        wait(3.5);
        vaccuumGateOpen();
        wait(10);
        shooterOff();
        vaccuumGateClose();
    }
    // Repeat
}

// Main routine
int main(){
    printf("Welcome\r\n");

    wait(3);

    hippobotamus();

    while (1){

    }
}
```