# Game of Drones: House of the Slug

**December 9, 2022**

Yueyi Mao
Connor Guzikowski
Stephanie Lu

## Table of Contents

## Introduction

The goal of the final project was to apply the concepts of event driven programming, sensors, and motors learned in previous labs to design and implement a robot that could complete the specified challenge. We were to build an autonomous robot that could navigate the given field, launch ping-pong balls into the basket, and return to the reload zone.

# Part 1: Robot Design

## Basic Design

The full robot needed several different parts to be able to satisfy the minimum requirements of the objective. These parts included a ping-pong ball launcher, wheels for movement on the field, and sensors for navigating the field while resolving collisions from walls or obstacles.

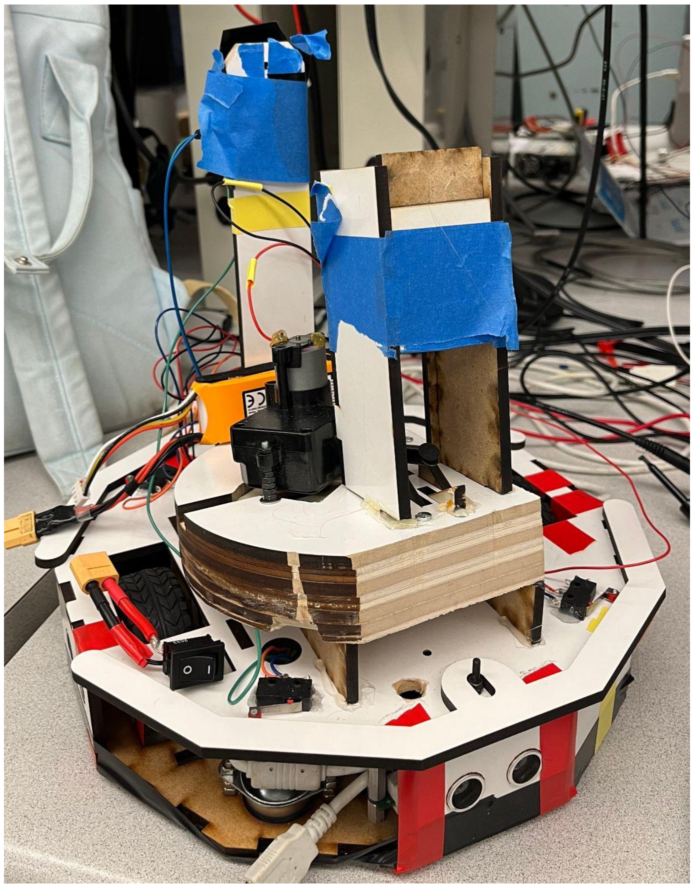

*Figure 1.1.1 Fully constructed autonomous robot*

## Prototype

The design used four ultrasonic sensors and bumper switches for field navigation and a beacon detector mounted at maximum height to locate the basket. We decided to use stepper motors for accurate movement on the field. The chassis was designed to be octagonal and the full robot would be built low to the ground with four casters to provide better movement.

## Changes

The first idea of our robot was to find the beacon and map out the field using the ultrasonic sensors to measure distance, then orient the robot parallel to the field by finding the minimum distance to the walls. The beacon detector would locate the 2KHz beacon above the basket, and then the robot would orient, move forward, and turn to the beacon. Using stepper motors, the steps could be recorded for easy reorienting after launching the ping-pong balls into the basket. These ideas were refined during implementation due to the limitations of our parts.

Over the course of implementation, we discovered that some portions of the robot needed to be changed to meet minimum requirements. These changes included:

- Increased launch angle for increased launch distance
- Inclusion of front and rear bumpers for collision detection
- Increased beacon detector gain
- Reduced usage of the ultrasonic sensors

# Part 2: Launcher

## Launcher

After three revisions, we finally settled on a design that used a single motor and a spring.

Our shooting mechanism has three different states.

The first stage is the preparation stage.

During the preparation stage, balls are stacked in the vertical chute and the firing pin is set to the initial position. This is shown in *fig. 2.1.1* below.

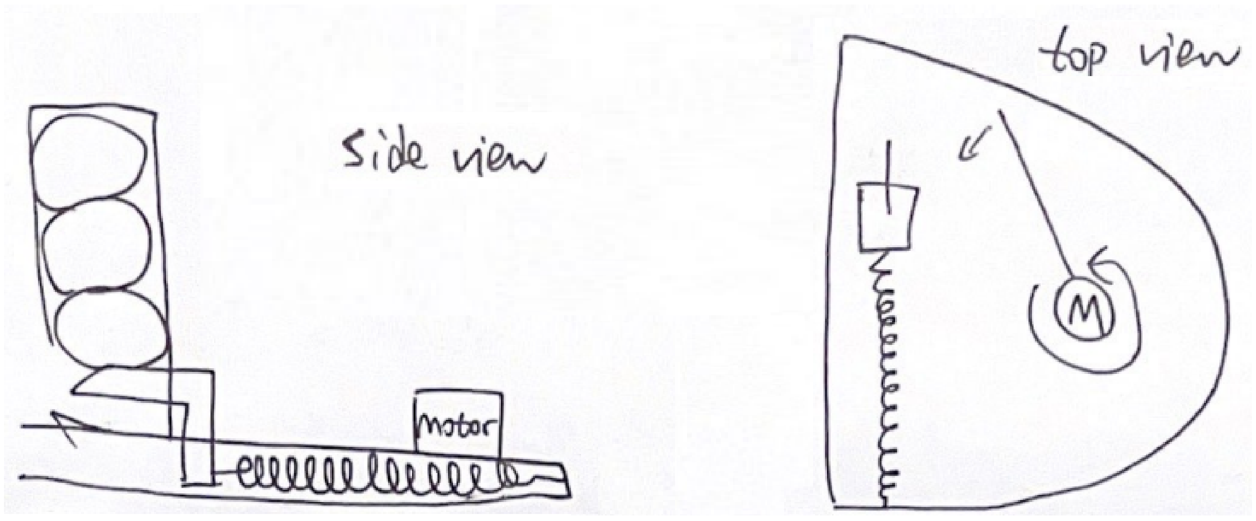

*Figure 2.1.1 Side view and top view of preparation stage*

The second stage is the loading stage.

During the loading stage, one ball moves into the loading zone as the arm on the motor pulls the firing pin back. This is shown in *fig. 2.1.2* below.

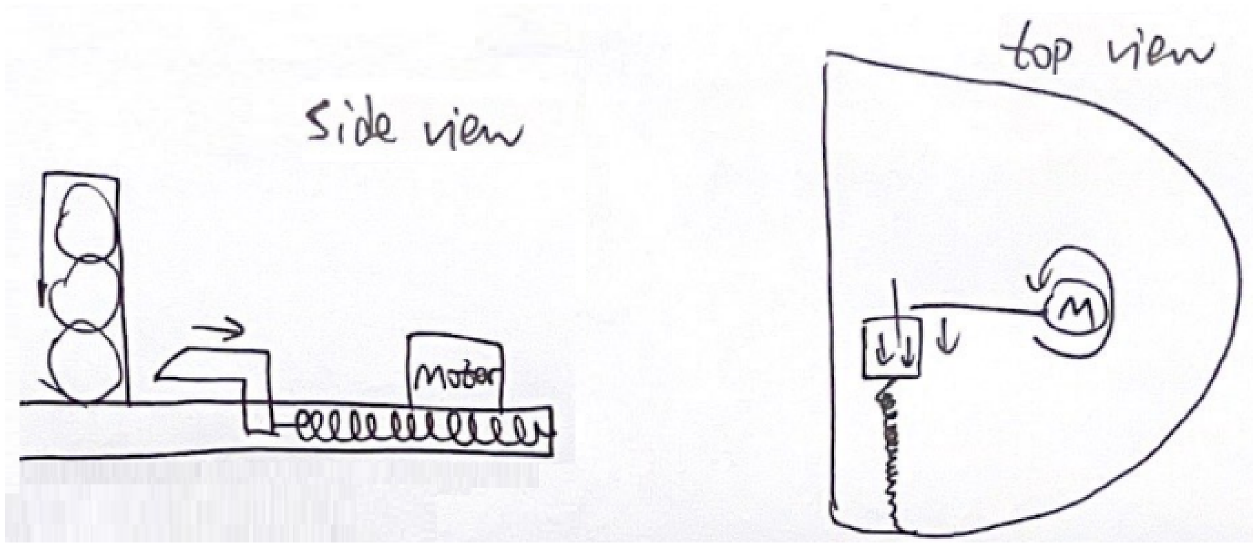

*Figure 2.1.2 Side view and top view of loading stage*

The third stage is the launching stage.

In this stage, the arm releases the firing pin and the firing pin is pushed forward by the spring. This is shown in *fig. 2.1.3* below.

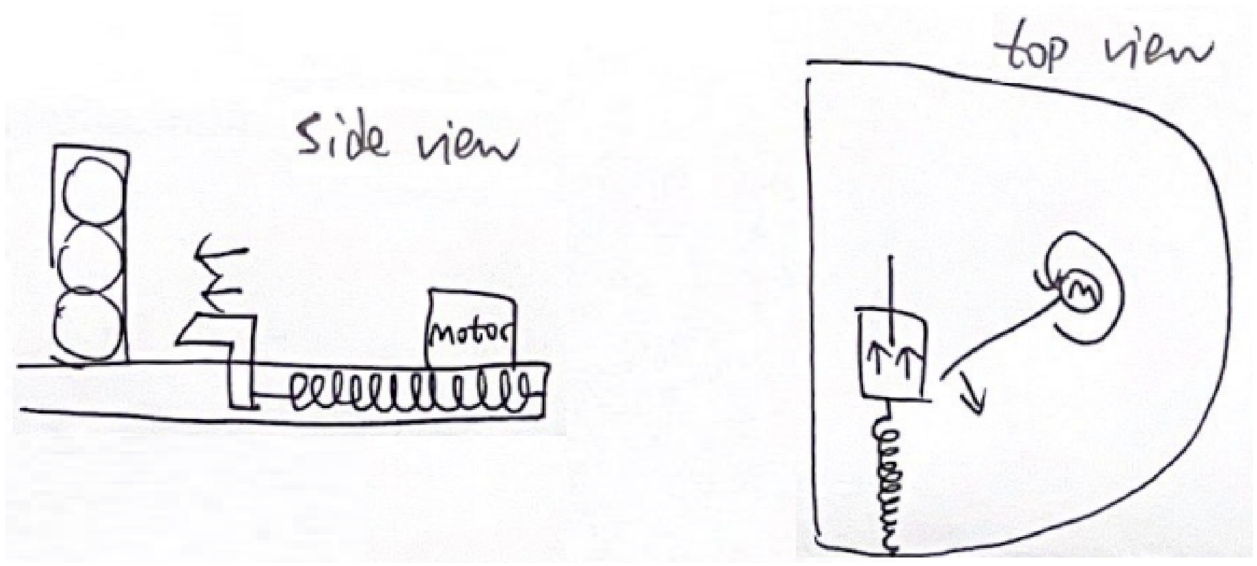

*Figure 2.1.3 Side view and top view of launching stage*

After the successful launch, the motor continues to spin and we are back in the ready state. This is shown in fig. 2.1.4 below.

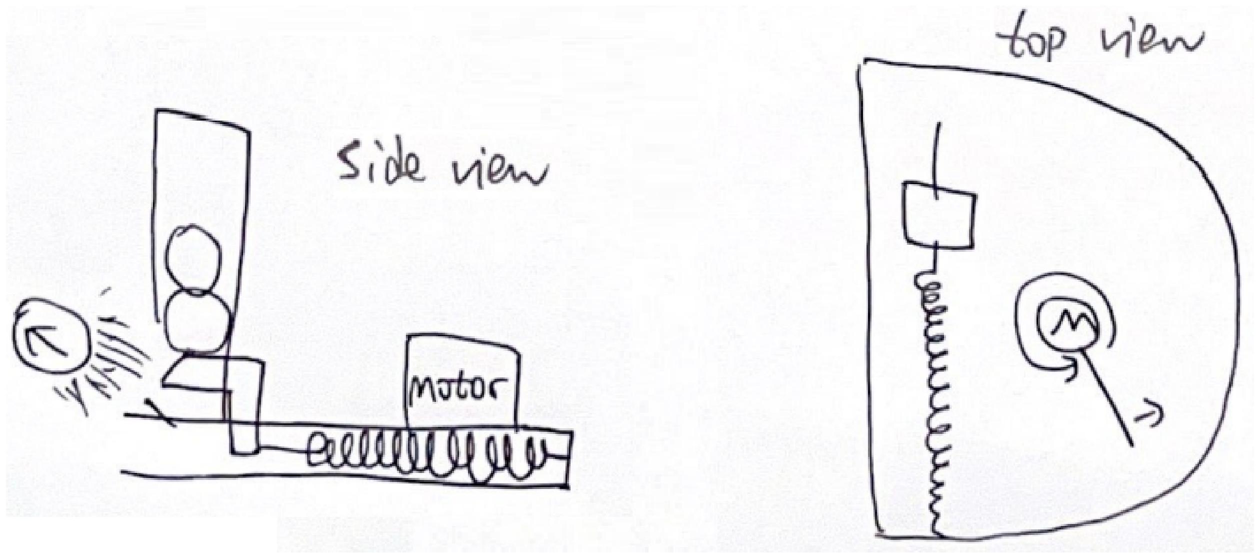

*Figure 2.1.4 Side view and top view of a new  preparation stage*

## What worked

- DC motor with gearbox

## What didn't work

- DC motor with pvc pipe
  - This was because the DC motor was too weak for launching the ping-pong ball the necessary distance
  - Launching from a DC motor has low accuracy and poor repeatability

# Part 3: Navigation

## Ultrasonic Sensor

The HC-SR04 ultrasonic sensors send out an ultrasonic sound pulse after the trigger pin is set to high for at least 10 microseconds. After sending out the pulse, the echo pin goes high and does not drop until the pulse that was sent out is received again, or after 38ms if there is nothing detected.

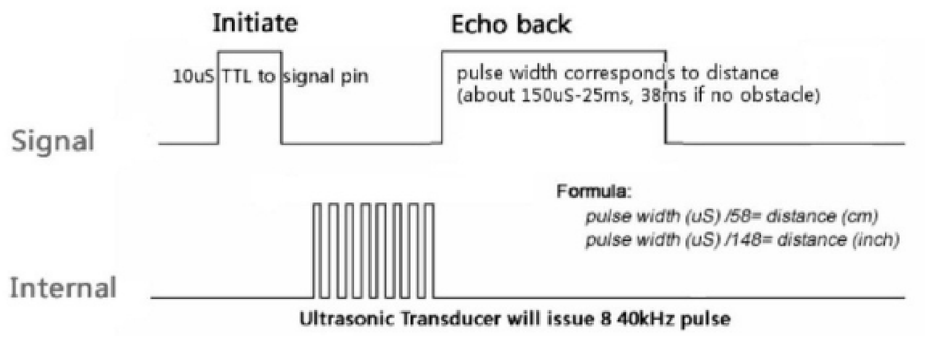
Initiate
Echo back
10uS TTL to signal pin
pulse width corresponds to distance
(about 150uS-25ms, 38ms if no obstacle)
Signal
Formula:
pulse width (uS) /58= distance (cm)
pulse width (uS) /148= distance (inch)
Internal
Ultrasonic Transducer will issue 8 40kHz pulse

*Figure 3.1.1 Timing diagram for the UC-SR04 Sensor*

The precision required for measuring the echo pulse width made this a challenging task. On top of needing the measured time to be just right, the sensors had to have a delay in between each measurement to avoid interference from the wrong ultrasonic pulse.

The way we went about implementing this was through the use of a state machine to drive each of the sensors and a change notify register to measure the time that the echo pulse is high.

```
static void RunPingFSM(void){
    switch(ping_state){
        // Immediately go from the trigger state to the wait state as
            // by the time this function has been ran, the time to be on
has passed
        case ON1:
            TRIGGER1 = HIGH;
            ping_state = WAIT1;
            trigger_time = ES_Timer_GetTime() * 1000;
            break;
        case WAIT1:
            TRIGGER1 = LOW;
            if(ES_Timer_GetTime()*1000 - trigger_time > CYCLE_DELAY){
                ping_state = ON2;
            }
            break;
        case ON2:
            TRIGGER2 = HIGH;
            ping_state = WAIT2;
            trigger_time = ES_Timer_GetTime() * 1000;
            break;
        case WAIT2:
            TRIGGER2 = LOW;
            if(ES_Timer_GetTime()*1000 - trigger_time > CYCLE_DELAY){
                ping_state = ON3;
            }
            break;
        case ON3:
            TRIGGER3 = HIGH;
            ping_state = WAIT3;
            trigger_time = ES_Timer_GetTime() * 1000;
            break;
        case WAIT3:
            TRIGGER3 = LOW;
            if(ES_Timer_GetTime()*1000 - trigger_time > CYCLE_DELAY){
                ping_state = ON4;
            }
            break;
        case ON4:
            TRIGGER4 = HIGH;
            ping_state = WAIT4;
            trigger_time = ES_Timer_GetTime() * 1000;
            break;
        case WAIT4:
            TRIGGER4 = LOW;
            if(ES_Timer_GetTime()*1000 - trigger_time > CYCLE_DELAY){
                ping_state = ON1;
            }
            break;
    }
```

```
void __ISR(_CHANGE_NOTICE_VECTOR) ChangeNotice_Handler(void) {
    static char readPort = 0;
    readPort = ECHO_BIT;
    IFS1bits.CNIF = 0;
    if(readPort){
        start_time = (ES_Timer_GetTime() * 1000) + (TMR1/TICKS_PER_MICRO);
    }
    else{
        end_time = (ES_Timer_GetTime() * 1000) + (TMR1/TICKS_PER_MICRO);
    }
}
```

*Figure 3.1.2 State machine and change notify register used*

Once we were able to get the time of flight for the ping sensors, we were able to get the distance measured by using the equation: $Distance\ in\ inches\ =\ Time\ /\ 148$. From this we were able to get the measurements that we were expecting.

Going into this project, we were planning on using the ultrasonic sensors to do pretty much all of the navigation for us. We first planned on having four ping sensors on the robot, set up in 90 degree intervals. We would then use the ping sensors to try to localize the robot with respect to the field. This would in theory be done by having the robot rotate to find the beacon, rotate a set amount to the left, and then turn 90 degrees to the right, taking measurements from the ping sensors as it goes along.

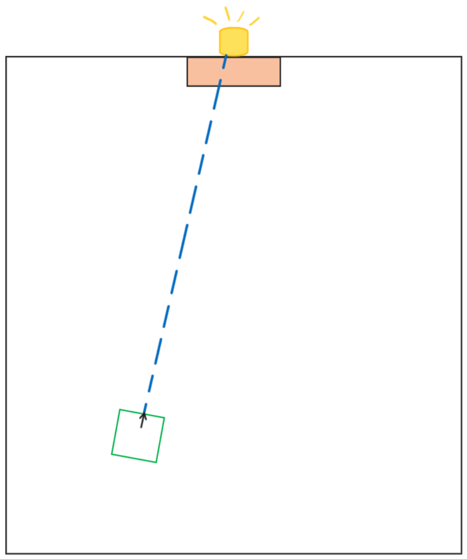
*Figure 3.1.3 Orientation of the robot when it finds the beacon*

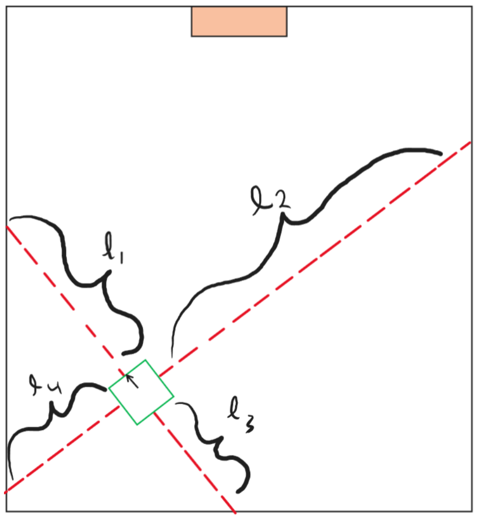

*Figure 3.1.4 Robot turns left*

The exact amount of rotation the robot does in this step was not a set amount. We never got far enough into making this work to determine the exact angle we needed. The reason for this step will be explained later.

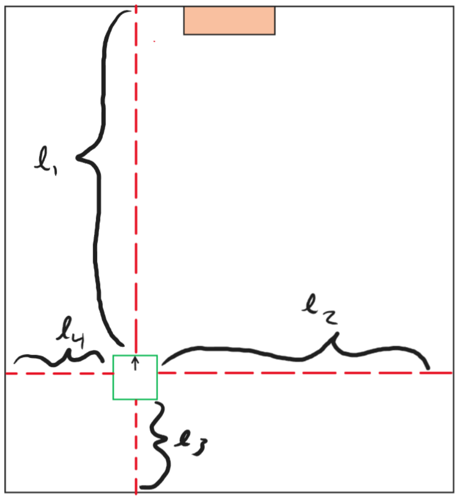

*Figure 3.1.5 Robot reaches alignment with the field*

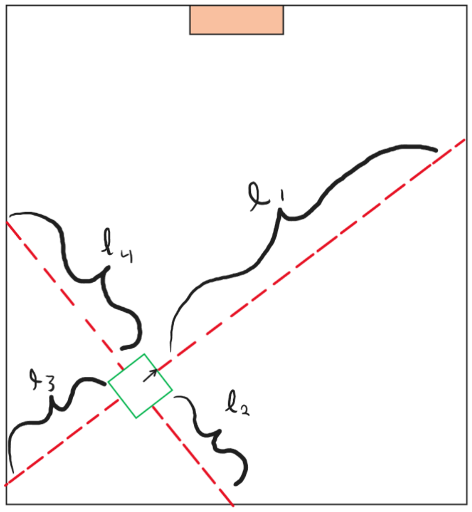

*Figure 3.1.6 Robot rotates 90 degrees right*

The idea behind this plan is that as the robot rotates to the right and takes in the measurements L1 - L4 using the ping sensors, there will be a point where L2 - L4 are at a minimum. This point of the rotation is where we would determine the robot to be aligned with the orientation of the field, and we would use that reference to determine the next steps for the robot.

There are several problems with this approach. First, the ping sensors are not precise or accurate enough to find a proper minimum, which would result in the robot being off of the proper orientation by a couple of degrees. If the ping sensors are sending the ultrasonic signal at an angled face, this throws off the sensors massively, even if the angle is relatively small (~15 degrees). Second, the idea in theory would work, but as soon as an obstacle is introduced, it makes localization significantly harder.

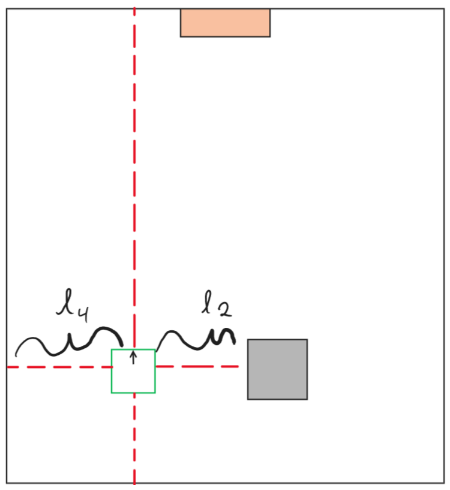

*Figure 3.1.7 Difficult field positioning*

When we have a field placement similar to the orientation above, this makes it a lot harder for the robot to determine if it is on the right or left side of the field.

The final issue with this plan is that the ping sensors are severely affected by noise when the stepper motors are running. Whenever we want to read the ping sensors, we would have to stop, take a measurement from all of the sensors, and then turn the motors on and repeat. This could result in a very long process.

After testing the ping sensors and realizing all of this, we realized that we needed to come up with a plan that would only rely on rough measurements from the ultrasonic sensors. Our new and final plan was the following:

1. Look for the beacon
2. Measure the distance to the back wall
    a. Calculate how far we need to move to be in a position that our launcher can make a shot from
3. Move forward the calculated distance
    a. If there is a bump, stop and read the left and right sensors.
        i. Go to the side that has the greatest distance.
        ii. Stop and backup a couple of inches when a front bump is detected.
        iii. Go back to part 1

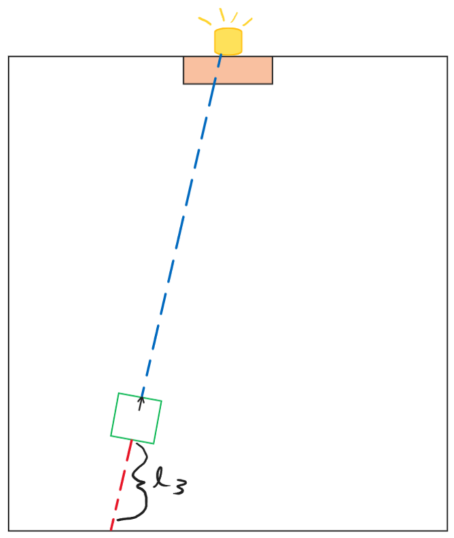

*Figure 3.1.8 First Step*

The first step of this plan is pretty simple. All we are doing is getting an estimate of our vertical position on the field. This will help us determine how far forward we need to move in order to make the shot. As stated before, the ping sensors are not the most accurate, especially on angled surfaces, but through testing, we found that taking an average of the back sensor and eliminating readings above a certain value made for good enough measurements.

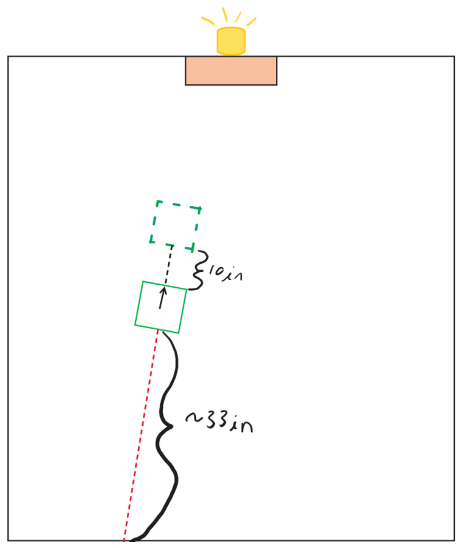

*Figure 3.1.9 Moving to Position*

The second step of this plan is to get in the proper positioning for the robot to be able to make its shots. Through testing, we figured out that the robot needs to be 33 inches away for the balls to go in, and that we need at least 10 inches of free space in front of the robot for the ball to go over an obstacle. Since we are using stepper motors, we were able to determine how many steps are in 10 inches, allowing us to scale that up or down to move any specific amount of inches.

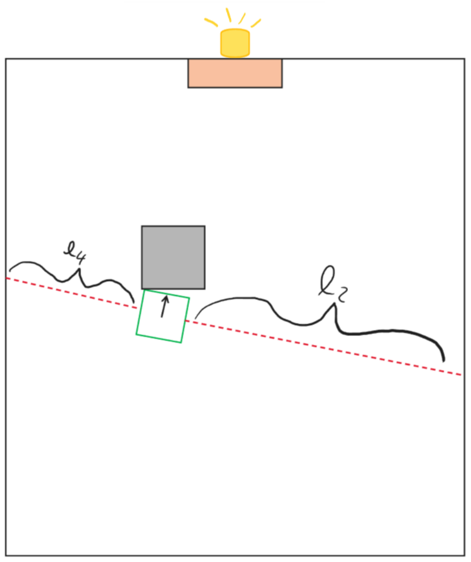

*Figure 3.1.10 Bump Detected*

In the case of an obstacle being detected, the robot backs up and takes an average of the measurements of L2 and L4 and looks to see which value is greater. After determining the greater distance, the robot then turns 80 degrees and then drives until it reaches the side wall. After doing so, it backs up and starts the process over again and looks for the beacon.

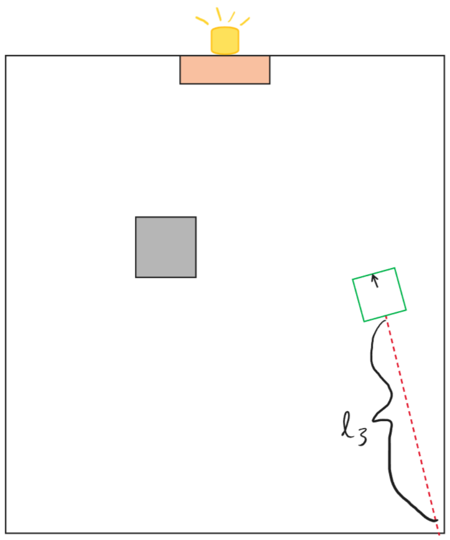

*Figure 3.1.11 Collision Handled*

## Beacon Detector

The beacon detector (schematic shown in *fig* $3.2.1$) was used to locate the 2KHz beacon above the basket and attenuate signals above and below this frequency. A large gain was necessary to detect the beacon from >6ft. A comparator was installed to provide a digital signal of HIGH or LOW to the Uno32.

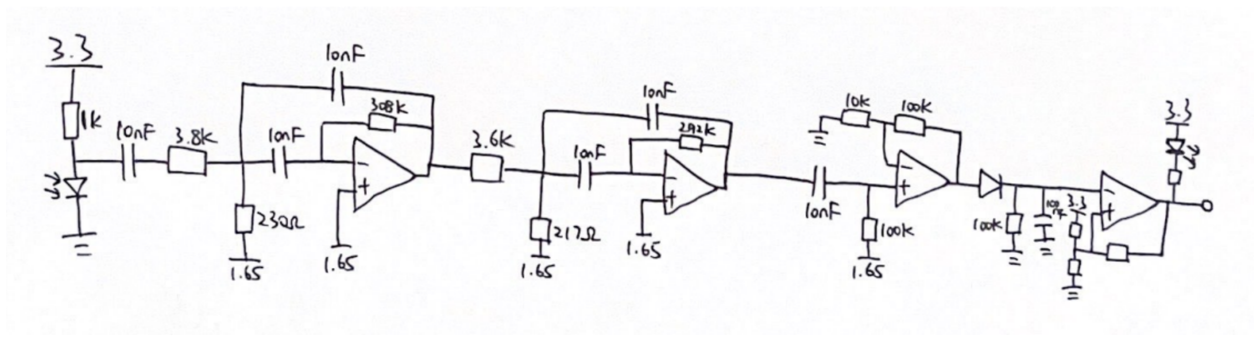

*Figure 3.2.1 Beacon Detector schematic*

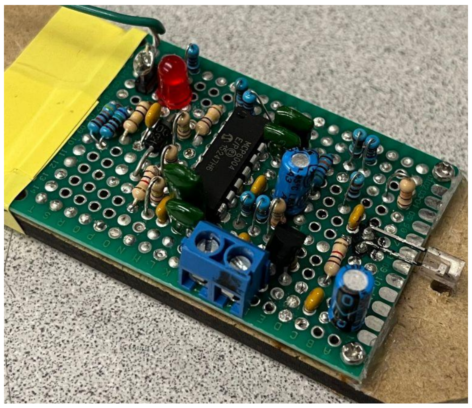

*Figure 3.2.2 Completed beacon detector soldered onto perfboard*

## Bumper

The following figure is the circuit schematic diagram of the bumper.

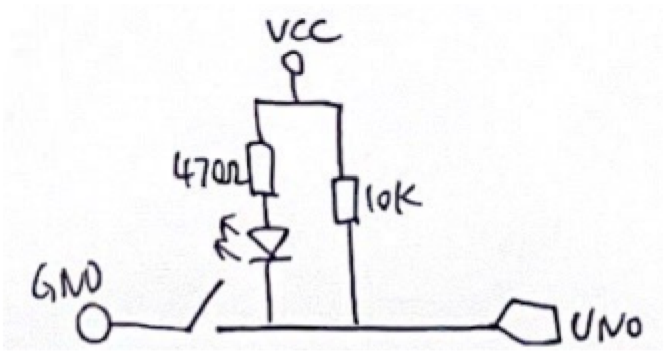

*Figure 3.3.1 Bumper switch schematic*

## Stepper Motor

We drove the stepper motors with a step signal using interrupts. To minimize step loss, we decided on a 750Hz step rate and used ⅛ stepping; every step of our motor was equivalent to 1.8/8° = 0.225° steps.

The step signal was produced using a simple state machine with an interrupt to decide the frequency of the state change, as shown in the snippet below (*fig. 3.4.1*).

```
void LeftMotorDriver(void)
{
    LEFT_DIRECTION = stepDir;
    switch (coilState) {
    case step_one:
        LEFT_STEP_SIGNAL = 1;
        coilState = step_two;
        break;

    case step_two:
        LEFT_STEP_SIGNAL = 0;
        coilState = step_one;
        break;
    }
}
```

*Figure 3.4.1 Step Signal State Machine*

Simple movement functions were created for the HSM in part 4. We required a function to turn a specified angle right or left, and a function to move forward or backwards a specified distance, as shown in *fig. 3.4.2*. The steps necessary to make a 360° spin were defined in STEPS_PER_SPIN, while the steps measured to reach 10 inches were defined in STEPS_PER_10.

The equation to find the amount of steps needed to make a specific turn angle was:
$steps = STEPS\ PER\ SPIN * DESIRED\ ANGLE / 360°$

The turn direction was decided by a LOW or HIGH signal to the driver.

We converted the steps to inches via the equation:
$steps_{inches} = DESIRED\ INCHES * STEPS\ PER\ 10\ INCHES / 10$

```
void TurnLeft(int angle){
    int steps = STEPS_PER_SPIN * angle / 360;
    Stepper_InitSteps(REVERSE, steps);
    Stepper_InitRightSteps(FORWARD, steps);
}
```

```
void driveInches(float inches){
    uint32_t steps = inches * STEPS_PER_10 / 10;
    printf("steps = %d\n\r", steps);
    DriveForward(steps);
}
```

*Figure 3.4.2 Helper functions for movement*

## What worked

***Bumpers:***
For the final design we reduced our usage of the ultrasonic sensors to the rear and side sensors. Front bumpers were installed to detect collisions, as the front sensor could not provide accurate measurements when not oriented parallel to objects. The installation of front and rear bumper switches allowed collision detection with walls and objects when moving into launch and reload positions.

## What didn't work

***Full use of ultrasonic sensors:***
The original design utilized four ultrasonic sensors to measure and record the distance to each wall or obstacle while the robot rotated. The recorded distances would be placed in four arrays and the elements compared to find the minimum distance to each wall.

In practice, the ultrasonic sensors were shown to be unreliable when not directed parallel to the obstacle of which the distance was being measured. To reduce the rate of error, we shut off the motors to reduce noise before every measurement, and measured only when facing nearly parallel to the walls and not the obstacle.

***High current draw:***
To increase the torque of the stepper motors we increased the current limit of the stepper motor drivers. While this would increase the speed and reduce step loss, the draw on the LiFe battery meant the robot could not run long enough to function properly (i.e. the bumpers would no longer detect collisions). One thing that we found out after the competition was that we could remove the low battery warning given in the ES_Framework, which would let us get a little more juice out of the battery.

# Part 4: Hierarchical State Machine

## Summary

We chose to focus on scoring from the back of the field so that our movement was simplified to navigating and scoring in half of the full area. Our robot would orient to the beacon and move forward a specified distance (*fig. 4.1.2*), navigating around obstacles as necessary (*fig. 4.1.3*). Once in position, our robot would send a signal to the launcher to launch all three ping-pong balls on a timer (*fig. 4.1.4*), before backing up the same distance to the reload zone (*fig 4.1.5*).

We chose to use an HSM in our event driven program. The state machine utilizes the stepper motors to move accurately by counting the number of steps, while using the ultrasonic sensors to record the distance needed for movement. There are only four upper level states (*fig. 4.1.1*), and four sub-level HSMs for movement, obstacle avoidance, launching the ping-pong balls, and reloading.

## Visuals

Here is a video of the autonomous robot completing the requirements of the project.

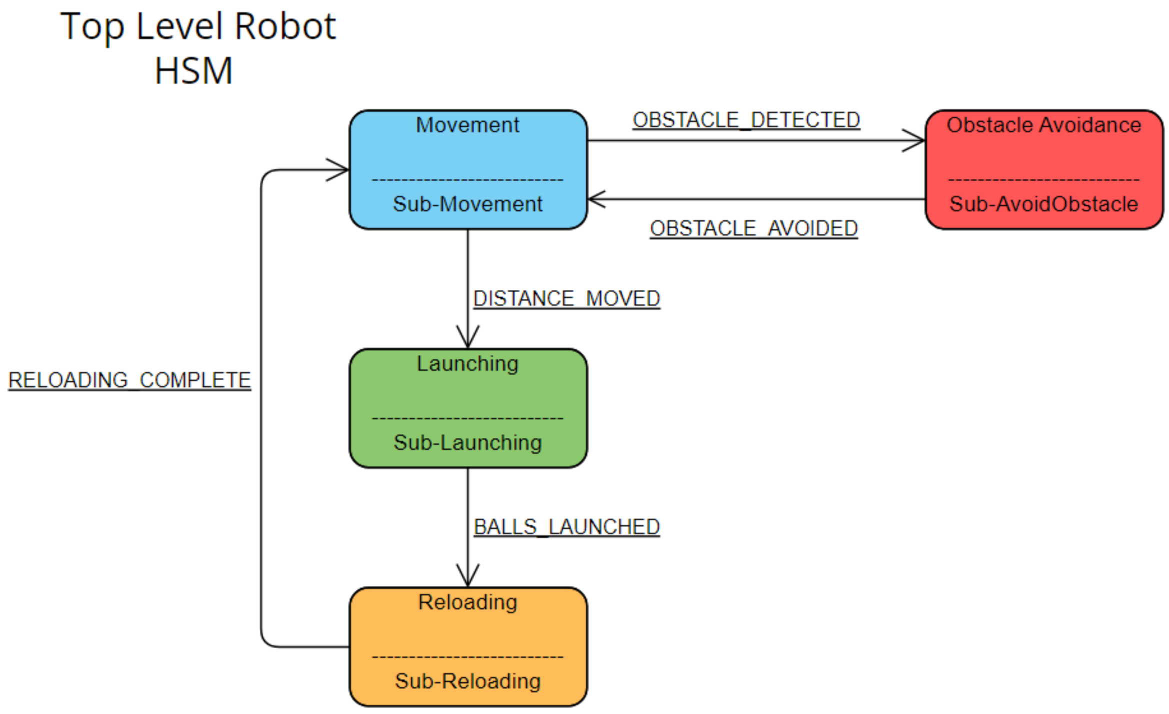

*Figure 4.1.1 Top Level Robot HSM*

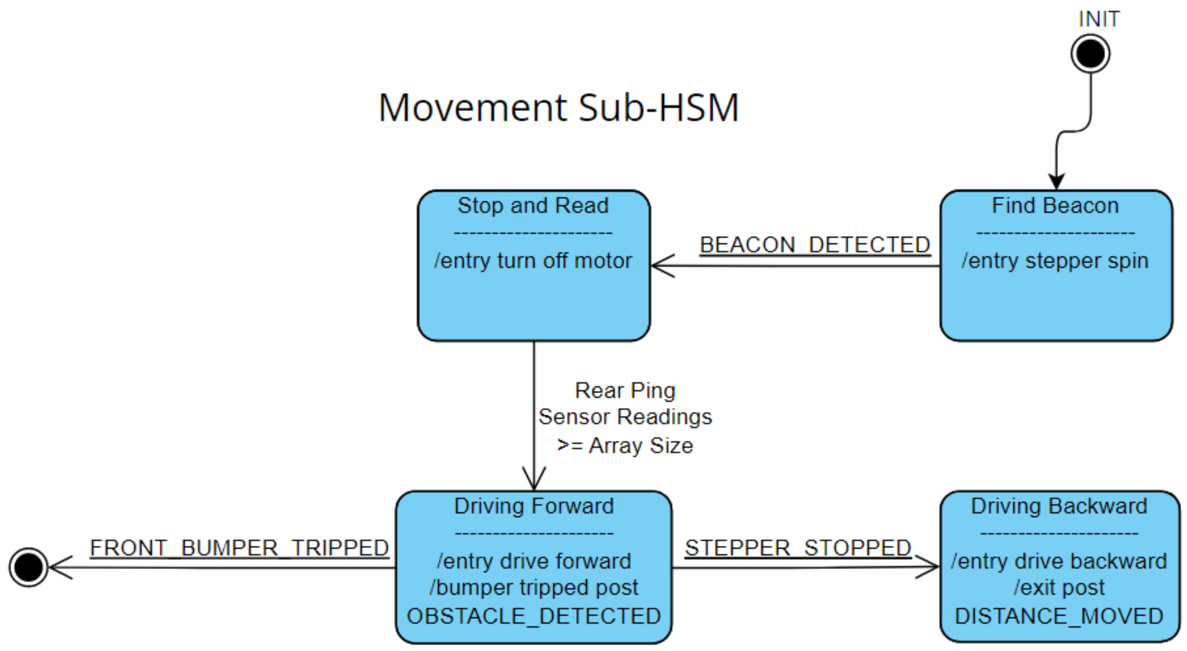

*Figure 4.1.2 Movement Sub-HSM*

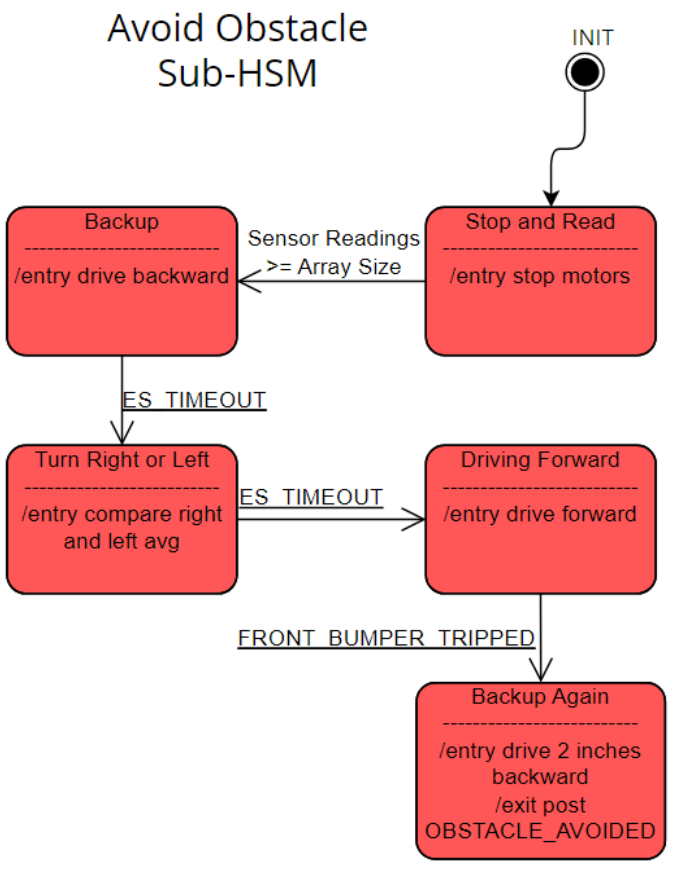

*Figure 4.1.3 Obstacle Avoidance Sub-HSM*

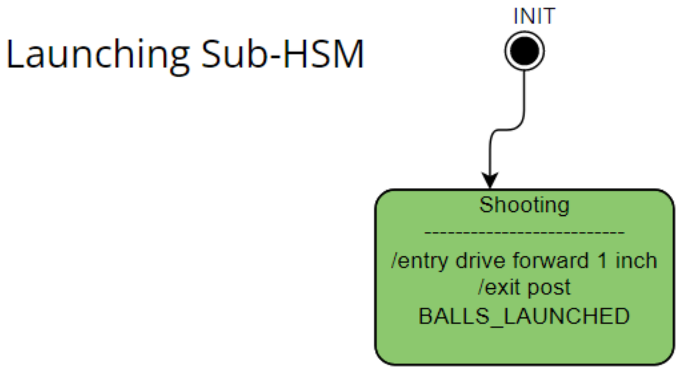

*Figure 4.1.4 Launcher Sub-HSM*

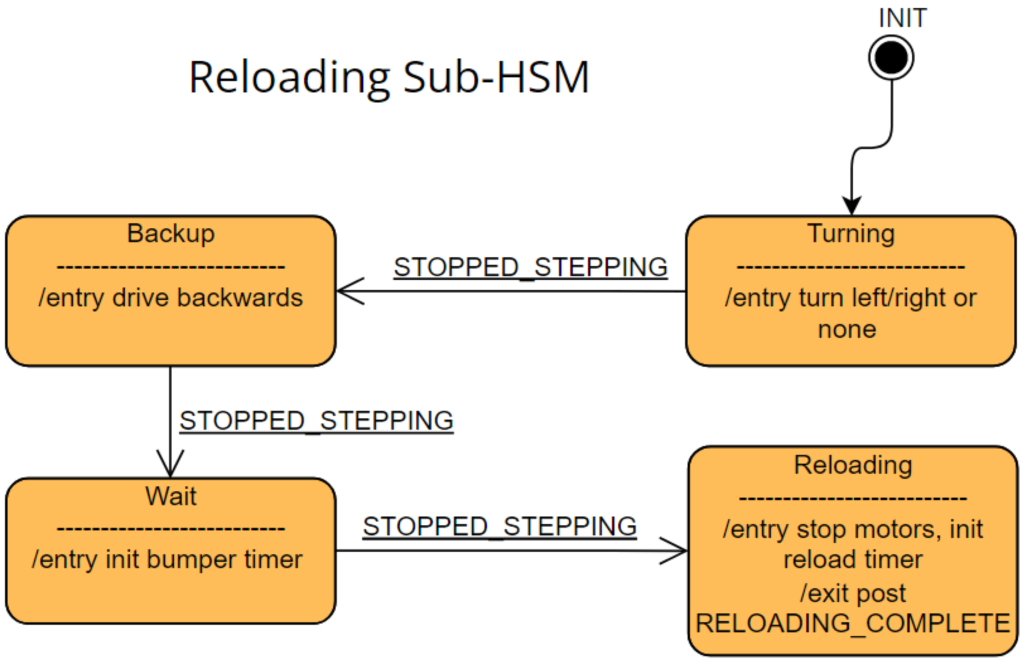

*Figure 4.1.5 Reloading Sub-HSM*

## Part 5: Bill of Materials

| Name | Price (ea) ($) | Amount Used | Total ($) |
|---|---|---|---|
| HiLetgo Universal Breadboard Double Sided Perfboard 1.6mm | 0.72 | 2 | 1.42 |
| 1N5817 Schottky Barrier Rectifier Diodes | 0.06 | 1 | 0.06 |
| Motor with gearbox (launcher) | 28.45 | 1 | 28.45 |
| TMC2209 v2.0 Stepper Motor Driver | 4.34 | 2 | 8.68 |
| Ultrasonic Sensor HC-SR04 | 2.18 | 4 | 8.73 |
| Nema 17 Stepper Motor | 10.56 | 2 | 21.11 |
| Wheels SharGoo OD 2.55” 1/10 RC | 3.28 | 2 | 6.56 |
| Hex Coupler 5mm | 2.46 | 2 | 4.92 |
| 250v 5A Lever Micro Switches | 0.76 | 4 | 3.06 |
| Ball Caster Bearings 33lb | 1.10 | 4 | 4.40 |
| MDF ⅛” | 2.18 | 2 | 4.36 |
| | | | **91.75** |

*Table 5.1 BOM for what was used in construction of the robot*

## Conclusion

Our final project was the accumulation of ten weeks of learned experience in which we discovered the full limitations and advantages of our chosen parts and tools. Through thorough testing and numerous alterations, we designed and implemented an autonomous robot that could navigate, score, and reload with a cohesive event driven program. Doing so, we gained valuable knowledge on project management, the engineering design process, and improved upon our mechanical, electrical, and software capabilities.