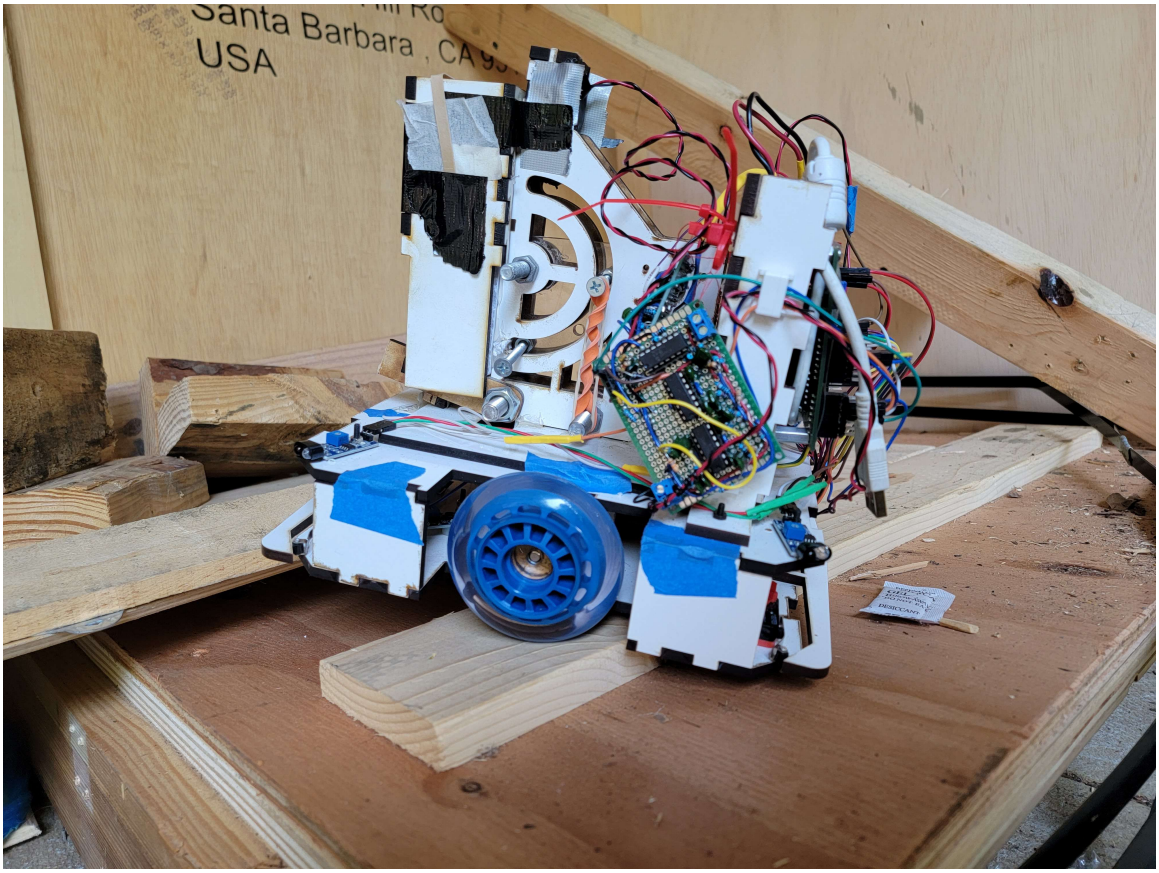
University of California, Santa Cruz

Baskin School of Engineering Electrical and Computer Engineering Department

Intro to Mechatronics Laboratory
ECE 118 Final Project Report

Aadhav Sivakumar, Kevin Jiang, Kamran Pakravan

June 2023



1 Introduction

The aim of this project is to provide a practical application of the knowledge gained in ECE-118 by tackling an open-ended problem. The objective is to design and construct an autonomous robot capable of navigating a designated game field, locating and depositing ping-pong ball ammunition into a specified basket, returning to the reload area, and maximizing the score within a 2-minute round. The ultimate goal is to create a small, independent robot (referred to as a droid) that can efficiently and reliably navigate the standardized field, identify the goal location (and optionally the goalie's pose), and accurately shoot the ping-pong ball into the goal. Points will be awarded for each successful ball delivery, with additional points granted for accomplishing challenging zones on the field. Throughout the course of the five weeks, teams of three will collaborate to design, build, test, and refine their robots until the assigned task is successfully completed. Adequate resources including practice fields, lab facilities, and expert guidance will be available to support the teams throughout the project.

1.1 Game Rules and Specifications

2 Brainstorming

2.1 Basic Strategy

Considering the rules of the game and the specific requirements, we made a strategic decision to adopt the simplest solution. By dividing the field into two sides, namely Left and Right, we reasoned that the obstacle could only be present on one side. With this insight, we implemented a wall-hugging approach, initially assuming that the obstacle would not be on our side of the field. However, if the obstacle is encountered, the robot promptly adjusts its navigation to move to the opposite side of the field, disregarding the side where the obstacle is located for the remainder of the game. To optimize efficiency, the robot minimizes navigation to the one point zone and backs up slowly to the two point zone, ensuring maximum points per cycle. We made a deliberate choice not to attempt the three point shots due to their unreliability, and the 1.5k beacon was not functioning during the implementation phase leading up to the minimum specification check-off. By prioritizing simplicity and reliability, we aimed to maximize the robot's performance within the given constraints and specifications of the game.

2.2 Initial Sketches

3 Mechanical

3.1 Base

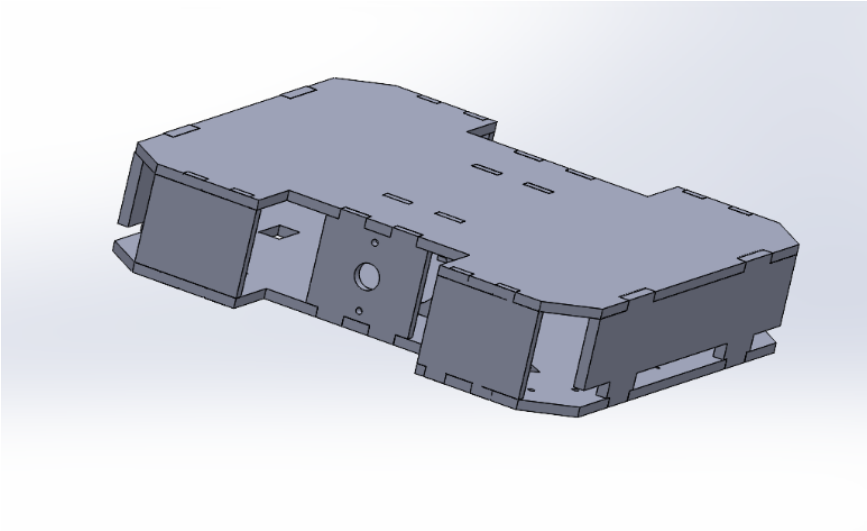


Figure 1: base side view

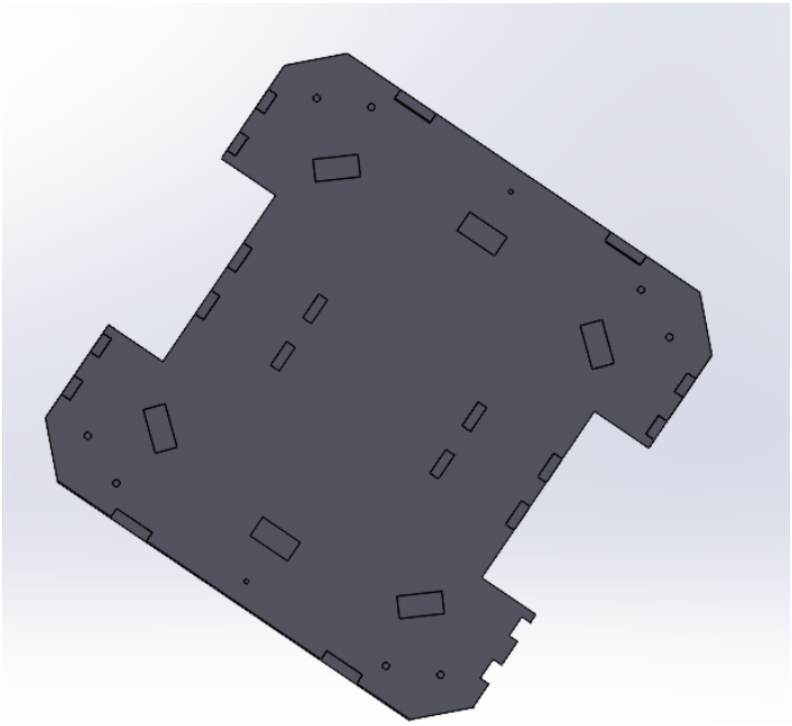


Figure 2: base bottom view

To design our robot, we cadded the bot using Solidworks to meet the specifications set. Our bot had to fit in an 11x11 cube, needed space for a battery, UNO board, motors, sensors, and an H bridge. For our design, we decided to go with two wheel drive, with ball bearings on the bottom to keep our robot stable and moving with minimal friction. This drive system was chosen instead of something such as a 4 wheel drive system as it would be fairly simple to get working, giving us more time to tackle more time intensive problems. We decided to go with a two layer design. The first layer was used for our motors and H bridge, while the other layer was used for the UNO board, our battery, and our shooting mechanism. The sensors were placed on both layers. One issue we ran into was that the two layer design did not have a lot of room to insert our battery and UNO board. To solve this problem, we installed a wall onto the back of our robot, which essentially acted as an extra platform to mount our electronics. The UNO and battery fit on the wall fairly well. For our bumpers, we opted for 4 bump detectors, one on each side of the robot. To trigger these bump detectors, we had two bumpers, one in the front and one in the back. These bumpers were mounted at their center, allowing them to flow freely when bumped on the left, right, or center.

3.2 Shooter

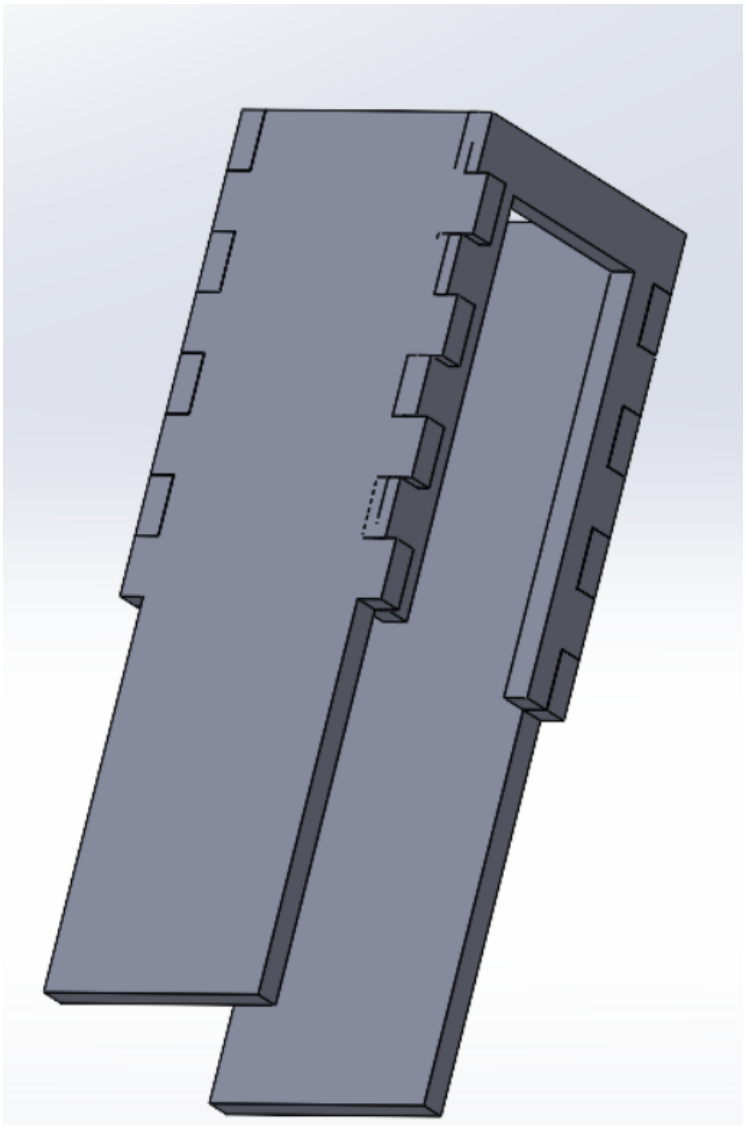


Figure 3: Loader cad model

While many groups went with a flywheel design for the shooter, we opted for a hammer design run with rubber bands and a slip gear. Essentially, we have a hammer that is cocked back by a gear. The hammer is connected to rubber bands that tighten as the hammer is cocked back, once the gear hits the side without any teeth (hence the name slip-gear), the hammer is released and slams

into the ball using the force from the taught rubber bands. This design proved effective, however there were some issues we faced when constructing it. One of the major issues was our loading mechanism. For our loading mechanism, we essentially planned to have the balls all stack on top of each other. When one fired, another one would fall into its place. However, we ran into a major problem where the ball on top would not allow the bottom ball to set properly in the firing chamber, which would then result in a poor shot. To counteract this, we added a small flap that would prevent the loaded ball from slipping out of position. This fixed our issues, giving us a fairly accurate launcher.

4 Electrical

4.1 Full sensor suite

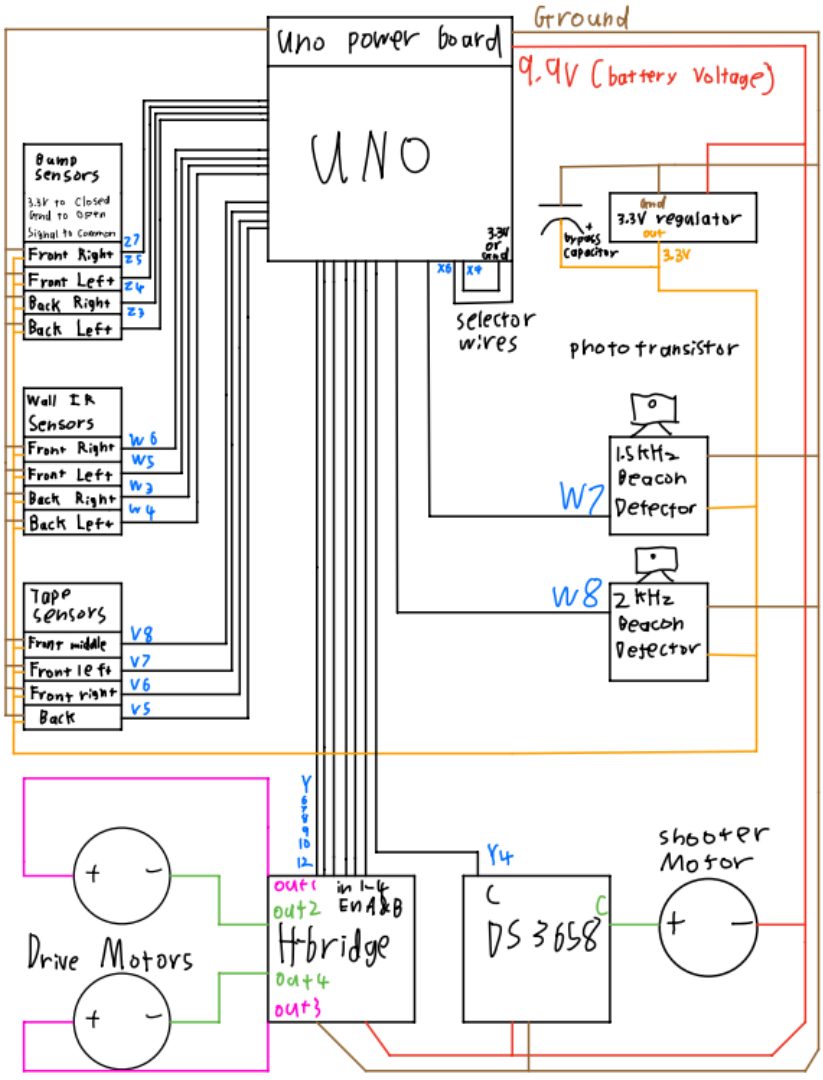


Figure 4: Diagram with all uno pin connections

4.2 Bumpers

The bumpers were the first sensors that we put on the robot. On the base there were two holes for each of the bumpers, and the bumpers were mounted with zip ties. This made them stationary, but it also allowed us to remove them in case they broke or needed to be serviced. The bumpers are just SPDT switches with 3 pins, common, normally open, and normally closed. On the NO (Normally open) pin, ground was connected. On the NC (Normally closed) pin, a 3.3v was connected which serves as a logic high. The common pin was connected directly to the UNO board, so when the bumper gets pressed that specific IO pin goes high. The bumpers were used to identify walls, as well as determine whether or not an object was in the desired path the robot wanted to travel.

4.3 Tape Sensors

The tape sensors on the bottom were the second thing that was mounted on the robot. We calibrated them to go high on the black line, and not have a high signal when on the white or detecting the yellow line. We had planned on being able to attempt 3 point shots after making a 1 point and 2 point attempt, but that plan didn't work out since the amount of extra time/effort required to make it a 3 point shot was not worth the extra points we would've gotten from it. The tape sensors were placed with 3 of them in the front and one of them in the back. After testing, we realized that the side sensors in the front didn't actually contribute anything, so the code only utilizes the middle front sensor to actually detect the zone.

4.4 Beacon Detectors

The beacon detectors were the sensors that didn't consist of just a single part. The beacon detectors that were created as a part of Lab 2 didn't suffice for the final project since they didn't detect from a good enough distance. If we add gain, it simply creates a phantom signal. A new design was made that filters with gain built in, and it was able to detect the beacon all the way from the reload zone of the field. We also created a 1.5Khz beacon detector to track where the goalie was, but we didn't end up using it because it was more beneficial to shoot as fast as possible instead of sensing when the goalie was out of the way.

4.5 Wall Sensors

The wall sensors were the ones that needed to be calibrated the most, but they were also the most useful. They were the main sensors used during the wall following. They are the same as the tape sensors, but just repurposed and turned 90 degrees. We originally were going to just use one on each side, but that wouldn't have been accurate enough and would result in a lot of quivering. Instead, we used 2 on each side and it was able to follow the wall at a particular distance and in a relatively straight line. We also ended up turning each of the sensors to point more in the corner instead of directly to the side.

5 Software

5.1 Top-Level-State Machine

In the development of our Event service driven hierarchical state machine, we constructed a top-level state machine to govern the behavior of our robot. The first state entails orienting itself towards the 2 Khz beacon and precisely positioning itself to track the corresponding wall, employing a physical switch to determine its initial orientation within the Orient state. Subsequently, the robot proceeds to navigate towards the one-point zone, adroitly managing any collisions encountered by rerouting to the alternate wall if required, within the navigation state. Upon reaching the one-point zone, the robot seamlessly transitions into the shooting state, meticulously aiming and firing twice before maneuvering into the two-point zone to deliver a final shot. Finally, in the reload state, the robot navigates back to its original reload position, taking into account the current wall it is following. This comprehensive framework ensures a smooth and efficient execution of tasks, highlighting the successful implementation of our Event service driven HSM for our robot's operation.

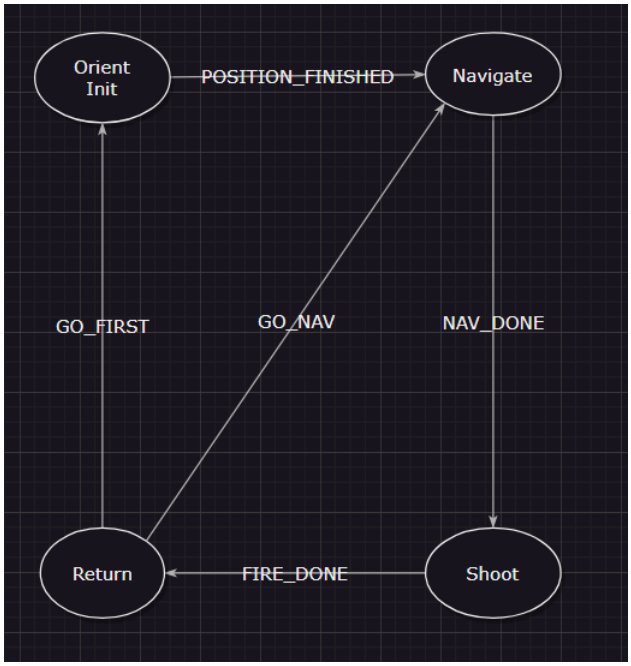


Figure 5: Top Level State Machine Diagram

5.2 Orient Sub State Machine (Level 2)

Within this sub HSM, the primary objective is to orient and position the robot accurately. To achieve this, the initial state involves spinning the robot until it detects the 2kHz beacon. Refer to the sub section below to

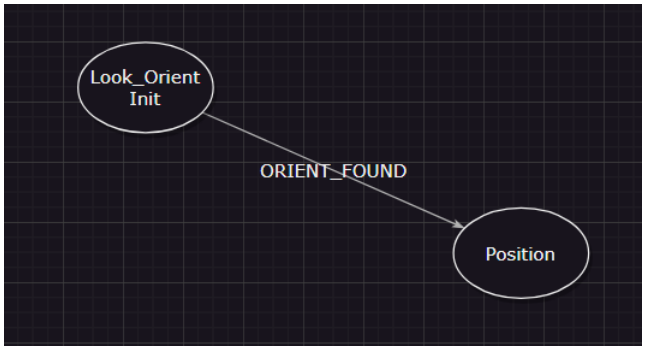


Figure 6: Orient Sub State Machine Diagram (Level 2)

5.2.1 Position Sub State Machine (Level 3)

Subsequently, the robot is instructed to back up and align itself with the back wall, followed by a 90-degree turn to establish its trajectory along a specific wall. Once the robot encounters a collision with its front bumpers, it performs another 90-degree turn to face away from the back wall, preparing for navigation towards the one-point zone.

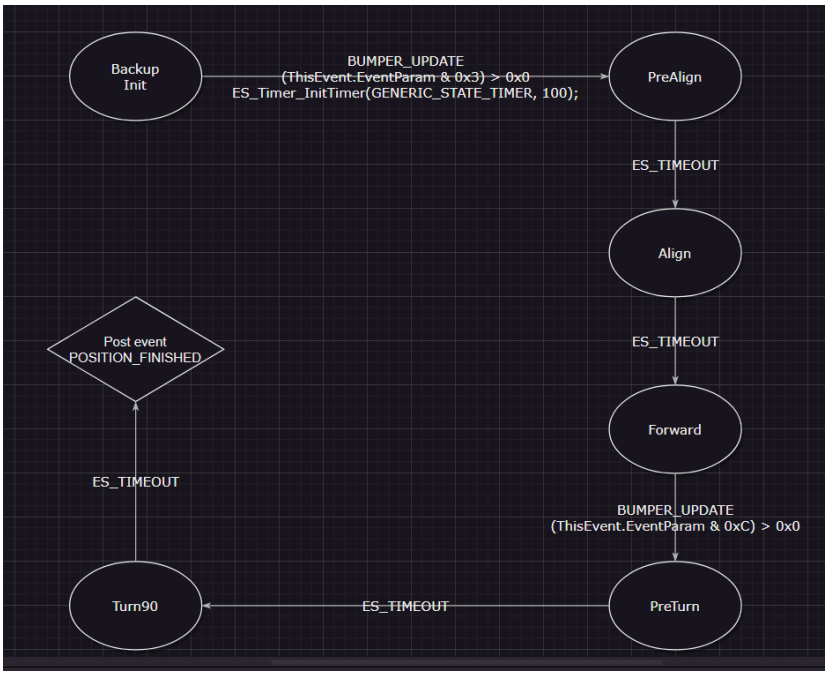


Figure 7: Position Sub State Machine Diagram (Level 3)

5.3 Navigation Sub State Machine (Level 2)

In this particular sub HSM, our focus is on implementing a straightforward navigation strategy. The initial step involves engaging in wall following until the back tape sensor detects a trigger, subsequently raising a flag. Once this occurs, we enter a waiting state for the front tape sensors to activate, resulting in the robot coming to a stop. Any potential collisions encountered during this process are addressed through a level 3 sub state machine known as "avoid." This concise sub HSM design efficiently addresses navigation requirements and effectively manages collision scenarios.

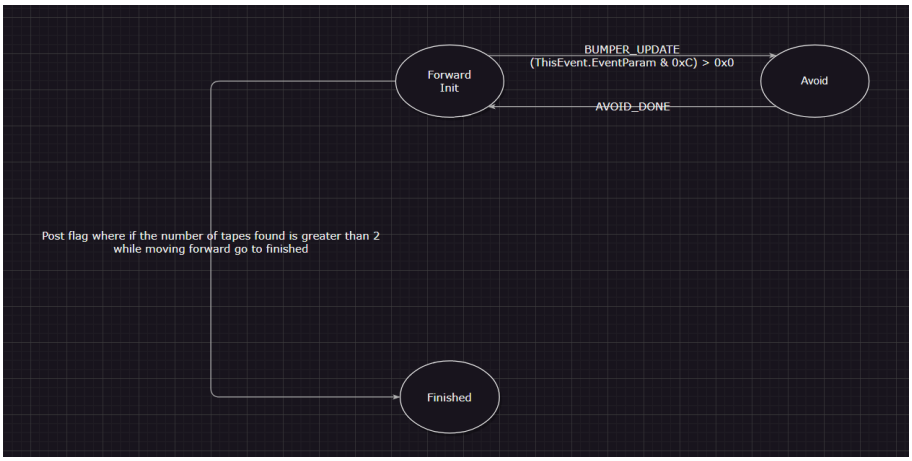


Figure 8: Navigation Sub State Machine Diagram (Level 2)

5.3.1 Avoid Sub State Machine (Level 3)

Within this level 3 sub state machine, our primary focus lies in implementing an effective mechanism for handling obstacle collisions. When a collision with an object is detected, our robot promptly initiates a sequence of actions. Firstly, it proceeds to reverse its movement, followed by executing a 90-degree turn and subsequently advancing straight towards the second wall. Since we encountered challenges due to the absence of encoders, we resorted to hard coding this behavior using a timer. Once the robot makes contact with the other wall, it performs a 90-degree turn back towards the beacon, resuming the wall following process. Importantly, this collision response also entails modifying our global flag that dictates the wall to be followed. This ensures that the robot is capable of successfully avoiding repeated obstacle collisions. However, should the situation necessitate it, the system remains adaptable to handle subsequent obstacle encounters.

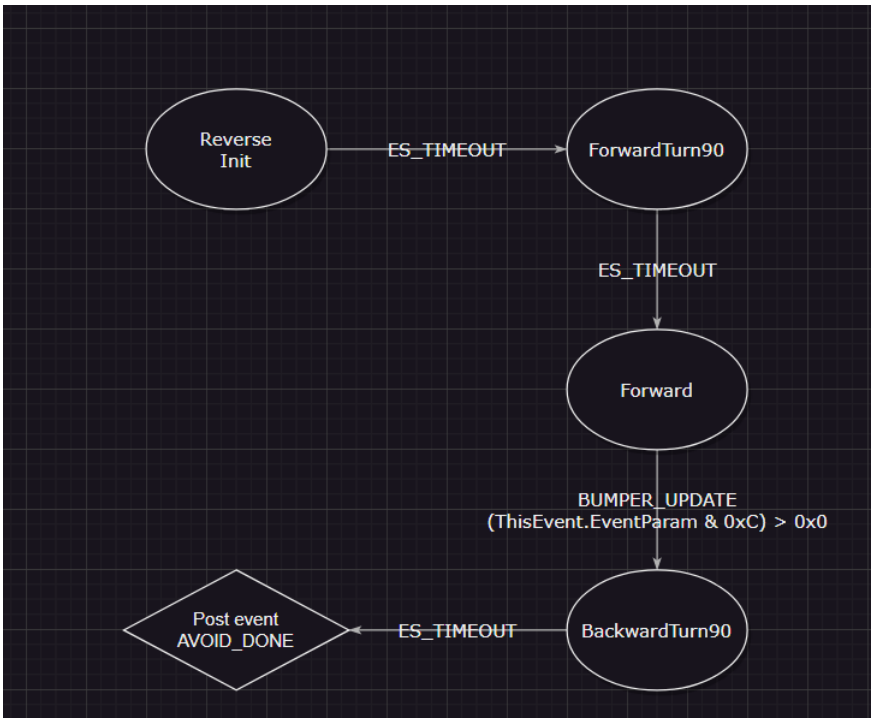


Figure 9: Avoid Sub State Machine Diagram (Level 3)

5.4 Shooting Sub State Machine (Level 2)

This sub state machine focuses on implementing the shooting behavior of our robot. Upon entering this state, the robot assumes its position within the one point zone, and initiates a rotational movement to locate the beacon. Once the beacon is successfully detected, the robot utilizes the natural offset to align its aim towards one of the goal's corners. To ensure precise calibration for aiming, we have incorporated an additional buffer state that operates on a timer. Following the beacon's identification, the state machine facilitates the firing of two balls. Subsequently, the robot strategically backs up into the two point zone, also governed by a timer, and executes a final shot before posting a "FIRE_DONE" event to signify completion of the shooting process. This sub state machine effectively manages the shooting behavior, providing accurate and controlled performance.

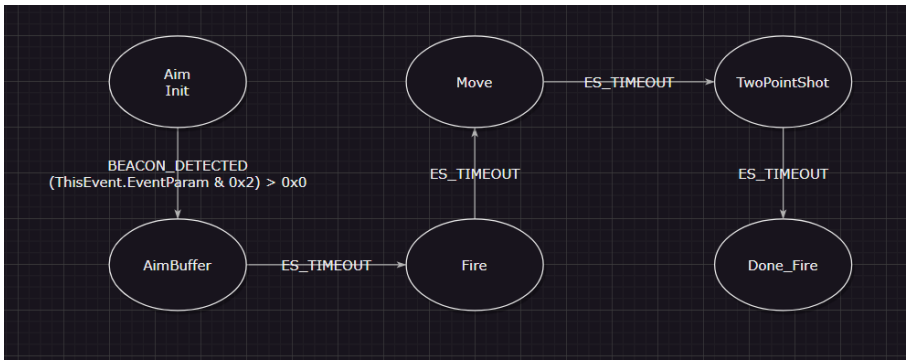


Figure 10: Shooting Sub State Machine Diagram (Level 2)

5.5 Reloading Sub State Machine (Level 2)

This final sub state machine is dedicated to implementing the reload behavior of our robot. Within this sub state machine, the robot engages in backward wall following until its back bumper makes contact. The subsequent actions depend on whether the current wall flag and reload position wall flag match or differ. In the case of a match, the robot halts its movement and waits for one second before posting the event "GO_NAV." Conversely, when the flags differ, the robot executes a 90-degree turn and resumes wall following in the forward direction until it reaches the other wall. At that point, it posts the event "GO_FIRST." This reload sub state machine effectively manages the reloading process by adapting its actions based on the state of the wall flags, enabling smooth and efficient reloading operations.

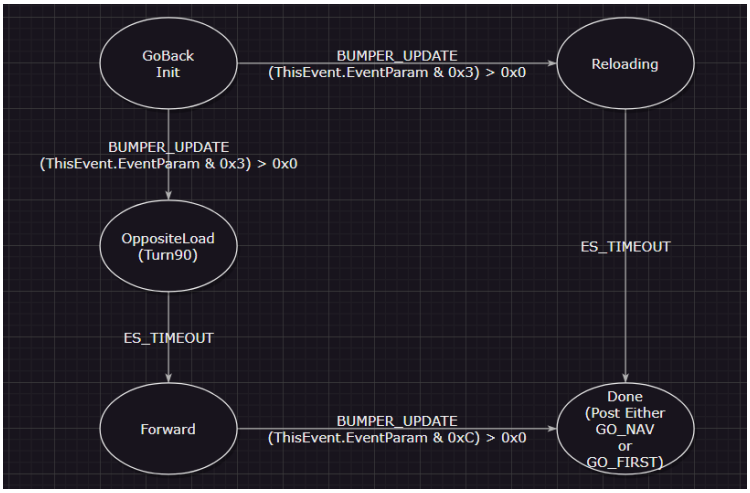


Figure 11: Reload Sub State Machine Diagram (Level 2)

5.6 Implementation

To facilitate the implementation of our code, we adopted an approach where we created a consolidated file named "robot.c" along with its corresponding header file "robot.h." These files encompass a comprehensive set of functionalities, including the initialization of AD, PWM, and digital IO pins, as well as the management of global variables, motor functions, and sensor inputs. By centralizing these elements within a single file, we were able to systematically test and debug the fundamental operations of the robot, ensuring the foundational functionality was robustly implemented. This modular design significantly contributed to the efficiency of our code development process.

```
unsigned int Robot_Beacons(void){
    static unsigned int Bit2k = 0;
    static unsigned int Bit1_5k = 0;

    unsigned int Beacon2k = AD_ReadADPin(AD_PORTW7); // 2k
    unsigned int Beacon1_5k = AD_ReadADPin(AD_PORTW8); // 1.5k
    if (Beacon2k > UPPER_THRESHOLD2k){
        Bit2k = 2;
    } else if (Beacon2k <  LOWER_THRESHOLD2k){
         Bit2k = 0;
    }

    if (Beacon1_5k > UPPER_THRESHOLD1_5k){
        Bit1_5k = 1;
    } else if (Beacon1_5k <  LOWER_THRESHOLD1_5k){
        Bit1_5k = 0;
    }
    return Bit2k + Bit1_5k; //returns bitmask where 2k is most significant,
    and 1.5k is least significant
}

unsigned int Robot_ReadTapes(void)
{
    return (((IO_PortsReadPort(PORTV)& PIN5) >> 5) +
    ((IO_PortsReadPort(PORTV)& PIN6) >> 5)+((IO_PortsReadPort(PORTV)& PIN7)
    >> 5)+((IO_PortsReadPort(PORTV)& PIN8) >> 5));
}

unsigned int Robot_ReadIR(void)
{
    return ~(((IO_PortsReadPort(PORTW)& PIN3) >> 3) +
    ((IO_PortsReadPort(PORTW)& PIN4) >> 3)+ ((IO_PortsReadPort(PORTW)& PIN5)
    >> 3)+ ((IO_PortsReadPort(PORTW)& PIN6) >> 3)) & 0x000F;
}
```

```
unsigned char Robot_ReadBumpers(void)
{
    return (((IO_PortsReadPort(PORTZ)& PIN3) >> 3) +
    ((IO_PortsReadPort(PORTZ)& PIN4) >> 3)+((IO_PortsReadPort(PORTZ)& PIN5)
    >> 3)+((IO_PortsReadPort(PORTZ)& PIN7) >> 4));
}
```

5.6.1 Motor control/Shooting

In order to operate our drive motors effectively, we employ an H-bridge configuration. This setup necessitates the use of specific pins, namely four digital output pins—two for each motor—to regulate the direction, and two PWM output pins, operating at a frequency of 1Khz, to control the motor speed. To facilitate this functionality, we develop a dedicated function that takes two parameters: speed (ranging from 0 to 1000) and direction (specified as FORWARD, BACKWARD, or STOPPED). By utilizing this function, we are able to dynamically control the speed and direction of the motors, enabling precise and responsive motor control within our code implementation.

```
char Robot_LeftMtrSpeed(unsigned int newSpeed, Direction var)
{
    if (var == FORWARD){
        IO_PortsSetPortBits(PORTY, PIN6);
        IO_PortsClearPortBits(PORTY, PIN7);
    } else if (var == BACKWARD){
        IO_PortsSetPortBits(PORTY, PIN7);
        IO_PortsClearPortBits(PORTY, PIN6);
    } else if (var == STOPPED){
        IO_PortsClearPortBits(PORTY, PIN6 | PIN7);
    }
    if (PWM_SetDutyCycle(PWM_PORTY12, newSpeed) == ERROR) {
        //printf("ERROR: setting channel 1 speed!\n");
        return (ERROR);
    }
    return (SUCCESS);
}

char Robot_RightMtrSpeed(unsigned int newSpeed, Direction var)
{
    if (var == FORWARD){
        IO_PortsSetPortBits(PORTY, PIN9);
        IO_PortsClearPortBits(PORTY, PIN8);
    } else if (var == BACKWARD){
        IO_PortsSetPortBits(PORTY, PIN8);
        IO_PortsClearPortBits(PORTY, PIN9);
    } else if (var == STOPPED){
```

```
        IO_PortsClearPortBits(PORTY, PIN8 | PIN9);
    }
    if (PWM_SetDutyCycle(PWM_PORTY10, newSpeed) == ERROR) {
        //puts("\aERROR: setting channel 1 speed!\n");
        return (ERROR);
    }
    return (SUCCESS);
}
```

In the implementation of our shooter mechanism, we have streamlined the control process by utilizing a single PWM pin. Since the shooter operates in a unidirectional manner, we only need to drive its speed through this pin. By adopting this straightforward approach, we simplify the code implementation, providing efficient control over the shooter's speed without the need for additional pins or complex configurations. This optimized setup ensures a streamlined and effective operation of the shooter within our overall system.

```
char Robot_ShooterMtrSpeed(unsigned int newSpeed)
{
    if (PWM_SetDutyCycle(PWM_PORTY04, newSpeed) == ERROR) {
        //puts("\aERROR: setting channel 1 speed!\n");
        return (ERROR);
    }
    return (SUCCESS);
}
```

5.6.2 Wall Following

The implemented code for wall following utilizing two IR sensors on each side follows a straightforward strategy: if both sensors are off or only the rear sensor is triggered, the robot banks towards the wall; if both sensors are triggered or only the front sensor is triggered, the robot banks away from the wall. To ensure the event-driven nature of the code, we introduced flags that dynamically change the behavior, rather than solely relying on events. This approach was adopted to address situations where transitions could potentially leave the robot in an idle state, leading to non-execution of the corresponding code. By incorporating flags, we achieved a more robust and reliable wall-following behavior within the code implementation.

5.6.3 Services

In our code implementation, we have developed various services to handle different aspects of our robot's functionality. Firstly, we have a service dedicated to managing the bumper, which involves detecting collisions and responding accordingly. Secondly, we have implemented a service for the IR sensors, which are responsible for detecting proximity to obstacles and enabling obstacle avoidance behaviors. To ensure reliable operation, we have incorporated a debounce

mechanism to filter out any spurious sensor readings caused by noise or fluctuations in the signal. Thirdly, our tape sensor service handles the detection of tape lines on the ground, allowing the robot to follow specific paths or navigate within designated zones. Similarly, we have employed debouncing techniques to eliminate false readings from the tape sensors. Lastly, our 2k beacon detectors service plays a crucial role in locating the beacon for precise positioning. To adjust the accuracy of the beacon detection, we have implemented hysteresis, which involves setting upper and lower thresholds for signal strength. This approach prevents rapid fluctuations in the detector output, ensuring stable and reliable beacon tracking. Overall, the inclusion of these services in our code implementation enhances the robot's perception and decision-making capabilities, enabling it to navigate its environment effectively.

5.6.4 Side selection

In order to determine the initial position of our robot and identify which wall it should hug at the beginning, we implemented two flags within our code. These flags serve as indicators of whether the robot started on the left or right side and specify the corresponding wall to hug. The flags are controlled by two jumper cables, which allow us to set the desired configuration before initiating the robot's operation. By incorporating these flags and utilizing jumper cables, we ensure the flexibility and adaptability of our robot's behavior, enabling it to effectively navigate and interact with its surroundings based on the specified starting conditions.

6 Conclusion

Our final attempt to meet the Minspec requirements took place on June 6th, 2023. In a previous attempt the day before, the robot achieved 6 points once but failed in subsequent attempts due to the box being positioned directly in front of it. To improve accuracy, we made adjustments to the turning mechanism, enabling the robot to successfully perform even when faced with obstacles in its direct path. A critical last-minute addition was the passive loading mechanism, which played a pivotal role in our overall success. Initially, we encountered challenges with the passive ball loader, resulting in the robot shooting only one ball at a time. However, through extensive testing, we identified and promptly resolved various issues, particularly regarding the ball stacking. During the final stages, we focused primarily on calibrating the shot angle and making minor adjustments to the state machine. Overall, these efforts significantly enhanced the robot's performance and brought us to meet the desired specifications.