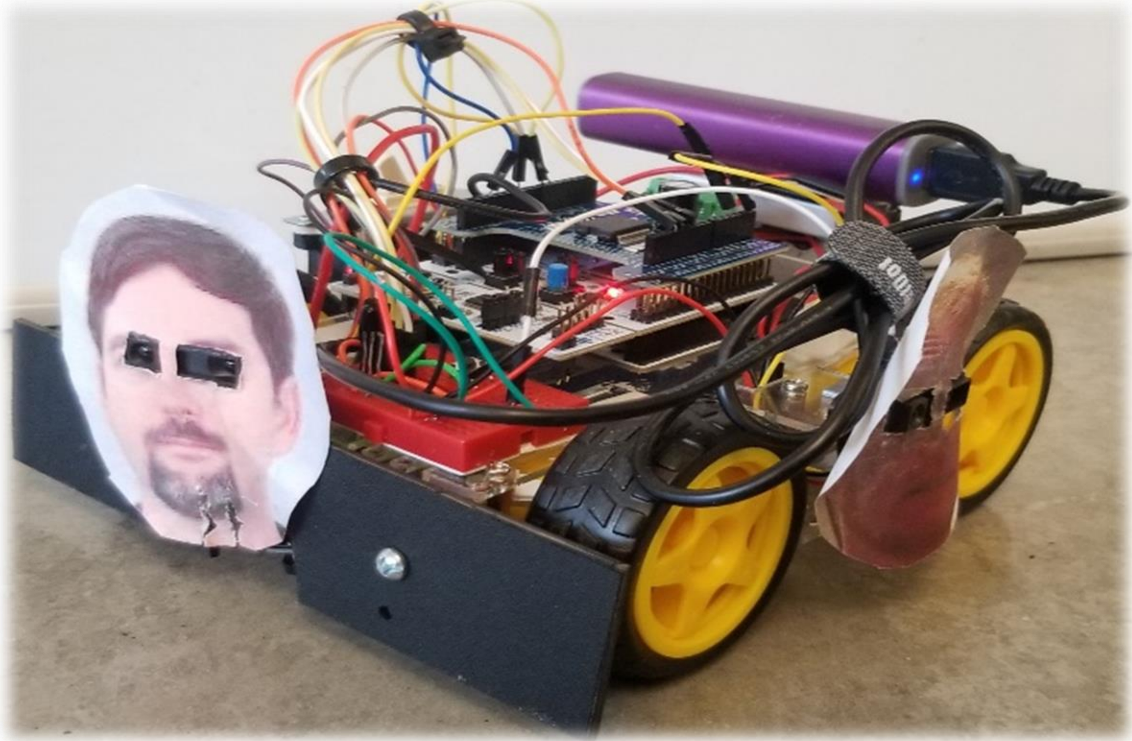
ME 405 SUMO Bot Report



Prepared for Charlie Refvam

March 16, 2020

Anthony Catello

Joseph Heald

Schuyler Ryan

Table of Contents

1: Project Overview 3
1.1: Background Information and Rules Overview 3
1.2: Project Customer 3
2: Design Specifications 4
2.1: Required Hardware 4
2.2: Safety Requirements and Competition Rules 4
3: Design Process 5
3.1: Strategy 5
3.2: Hardware Design 5
3.2.1: BOM 5
3.2.2: Component Selection Process 6
3.3: Software Design 7
4: Results 10
4.1: Hardware 10
4.2: Software 11
4.3: Overall 11

1: Project Overview

1.1: Background Information and Rules Overview

The purpose of this project is to design the software and hardware for an autonomous robot. The robot will compete in a sumo robot competition which entails two bots being placed in a four-foot diameter ring made of steel that is rimmed with a white paint outline. A single match lasts for three minutes and starts with an IR "go" signal sent to the bots. After that signal the bots act completely autonomously and do their best to be the last bot standing in the ring before the three minutes is up. If after three minutes both bots are still in the ring than the bot closest to the center of the ring is declared the winner of the match. Bots are not allowed to intentionally interfere with each other or intentionally cause damage to opponents. More detailed design specifications for the design and the rules can be found in Section 2.

1.2: Project Customer

We did our best to maximize the effectiveness of the bot while spending the least amount of money on hardware because we are students. This thought motivated much of the hardware choices we made and the software design and over strategy that we implemented. Overviews of our hardware and software design and be found in sections 3 and 4 respectively. This bot is best suited for someone who wants flexibility in their strategy and an overall low cost.

2: Design Specifications

This section will further discuss the design specification of our project. First, we will go over the required hardware, and then we will discuss the limitations and end safety requirements of the bot.

2.1: Required Hardware

Our bot is required to implement at least the hardware components found in table 1.

Table 1: Required Hardware

#	Hard Ware Requirment
1	Emergency shut off
2	Two Motors
3	Horizontal Motion Sensor
4	Opponent Detection Sensor
5	Line Detection Sensor
6	IR Reciever
7	Power Source
8	Chassis
9	ME 405 Board

Table 1 collects all the required hardware components for the completion of our bot. This list was determined using research and the problem statement documentation provided to us by the ME 405 lab. Limitations on these components was also imposed, these quantitative restrictions can be found in table 2.

Table 2: Hardware criteria and limitations.

#	Specification	Requirment
1	Weight	5lb MAX
2	Footprint	8"X8" MAX
3	Height	8' MAX
4	Main Battery Capacity	15Wh MAX
5	Nominal battery Voltage	24 V MAX
6	MotorPower	100 W MAX
7	Top Speed	1.5 ft/s MAX

2.2: Safety Requirements and Competition Rules

This sections further details the rules and regulations of the competition that do not pertain to hardware requirements. Of these rules the main one is that we aren’t allowed to disrupt or intentionally interfere with or harm an opponent’s bot.

3: Design Process

In this section we will outline the process of selecting our hardware components and writing our software.

3.1: Strategy

Going into the project our primary goal was to spend the least amount of money but get the most performance out of our bot. Because of this we decided that going for the most powerful motors and high-end sensors wasn't what we wanted. We decided that in order to win bot matches we would avoid confrontation, because we wouldn't have the motor torque or sensor quality to effectively seek out and then push an opponent out of the ring. So, we decided to bet on the fact that opponents wouldn't be as good at staying in the ring. Our overall strategy was to avoid contact with the other bot but circling the ring and speeding up or slowing down when the opponent got closer than we wanted them. In addition to opponent avoidance, we were hoping to "bait" opponents into approaching the edge of the ring to hunt us only to get out the way before they reached us. A diagram of what we wanted our code to do can be seen in figure 1. The bot would make an initial right turn and then detect the edge to make subsequent left turns.

Inside of Bot

Outside of Bot

Figure 1: Diagram of our bot strategy

3.2: Hardware Design

This section will first list the components that we used and then compare them to our design requirements.

3.2.1: BOM

The final cost of our components was approximately $97. Our components were sourced from the ME 405 lab, Amazon, group members' personal storages, and even the trash. In general, we chose the least expensive and simplest components that we could by being resourceful.

Table 3: Bill of Materials for Sumo-bot project.

Component Name	Qty.	Total Cost	Source	NOTE
Nucleo Microcontroller Board	1	-	ME 405	
Motor Expansion Board	1	-	ME405	
IR Reciever	1	-	ME 405	
Wiring	-	-	ME405	
Board Power Source	1	-	Group Member	Joey already had one
Chassis	1	$18	Amazon	
Motors	4	-	Amazon	Included with Chassis
IR Range Finders	2	$22	Amazon	
Edge Dectection Sensors	2	-	Group member	Found by Schuyler in garbage
Motor Power Source	1	$13	Amazon	
Magnets	2	$10	Amazon	
Optical Encoder	2	$22	Amazon	
Misc. Mounting Supplies	-	$12	Target/Home Depot	
	TOTAL	$97		

3.2.2: Component Selection Process

The microcontroller board that was provided to us was the Nucleo-L476RG micro-controller with a X-Nucleo-IHM04A1 DC motor driver expansion. Because this board was free and capable of running all the tasks that we needed it to, we implemented it into our design. Also provided to us was the VISHAY TSOP382 IR receiver module which fulfilled the requirement. From there we selected a chassis. Considering we already knew that we didn't want to spend a lot of money on the project and planned to avoid bot to bot contact we chose a very reasonably priced chassis that also came with 4 DC brushless motors. The exact power of the motors is unknown but seeing as we found the chassis and four of them for under $20, we figured there was no way that they could be over specification. This fact was confirmed during the competition when one of the other bots pushed us off without slowing down. For power we used two separate batteries, a TURNIGY 2200mAh 3S 25C LIPO Battery Pack for our motor power supply, and an Insignia portable battery pack for our micro-controller. The LIPO Battery Pack supplied a 11.2 DC voltage power supply, and while this is outside the 15 Wh specified, it only means that the bot is going to last longer without charges and the cost of the battery was cheaper when compared to other more specialized batteries. The Insignia portable battery pack supplied 5 volts through a USB-A to USB-mini cable.

From here we selected a sensor package. We used a total of six sensors including the IR receiver that was already mentioned. We wanted the ability to detect the edge of the ring on at least the front and outside of our bot, so we used two Model TCRT5000 IR Reflection Sensors, because they meet our requirements and were scrounged by a team member making them free. To simplify things we used the analog output and ignored the digital signal. To keep track of horizontal position we utilized a simple, inexpensive single channel encoder. The HC-020K Double Speed Photo-electric Encoder had a 20-slot encoder wheel that enabled us to detect 40 edges. These were

somewhat problematic and will discuss it further in section 4.1. The encoders were mounted above motors that came with our chassis. To manage cables better we drilled additional holes in our chassis. For opponent detection we selected the GP2Y0A21 Sharp IR Analog Distance Sensors because we were running us of time to put together the bot, they had two-day shipping and were inexpensive and seemed simple to implement.

Each of these sensors was assigned a pin on our board (see simple wiring diagram in figure 2).

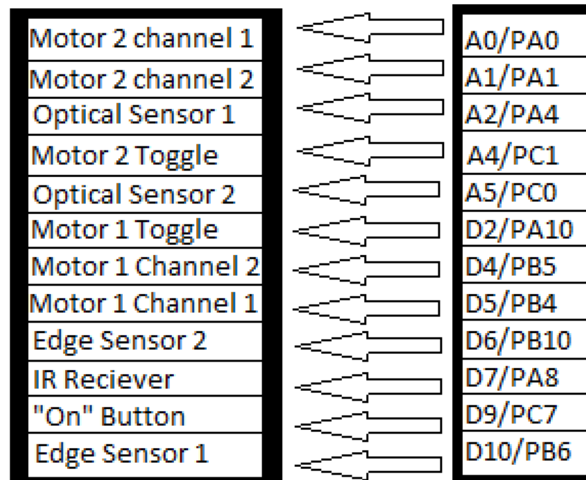


Figure 2:Diagram that details what pins are used for what purpose.

For an emergency kill switch we used a simple wall mounted light switch, this was placed in series between the battery and the motor header.

We also included an “ON” button on our bot, which functioned as an alternative to the IR remote. This was incredibly useful when more than a few groups were testing their SUMO-bot at once, since there were only 2 IR remotes provided by the lab.

To inspire fear in our opponents and add a completely necessary intimidation factor, we opted to hot glue the faces of the instructors over our IR distance sensors, which we paid for in the nightmares it induced in us.

3.3: Software Design

To control our bot, we wrote software that would pass information from the sensor package to a mastermind task that would “make decisions” and control our motors. Our mastermind task was called Strategy.

Strategy used shares and queues from our sensors to choose between “SharpLeftTurn()” or “EdgeTurning()” task functions that would set the PWM of the motor to turn the bot 90 degrees or get it to follow the edge of the ring. The code for “SharpLeftTurn()” is shown below.

```
def SharpLeftTurn():
    '''This function is run every 25ms and makes the SUMO bot take a
sharp turn approximately 90 degrees counterclockwise.'''
    while True:
```

```
        #only operate when the sharp_left_turn flag is set
        if sharp_left_turn.get():
            #reset edge sensor flag for front(1) sensor
            EdgeSensor1Flag.put(False)
            #reset sharp_left_turn flag
            sharp_left_turn.put(False)
            #sets duty cycles for each motor train then
resumes forward movement
            DutyCycle1.put(-100)
            DutyCycle2.put(-25)
            Right.set_duty_cycle(DutyCycle1.get())
            Left.set_duty_cycle(DutyCycle2.get())
            utime.sleep_ms(320)
            DutyCycle1.put(35)
            DutyCycle2.put(35)
            Right.set_duty_cycle(DutyCycle1.get())
            Left.set_duty_cycle(DutyCycle2.get())
            yield(0)
        yield(0)
```

It is worth noticing that instead of tracking the encoder during our turning we time the function. This is because we determined that doing some testing and using a timer instead of using our rather unreliable optical encoder. This also allowed us to make turns in place that didn't accumulate any horizontal distance.

A Task diagram and State Diagrams for Critical tasks along with timing justification calculations are included in the appendix. We chose to use these tasks because it allowed us to determine what situation the bot was in and then use that information to alter our bot's behavior, in a simple and effective way.

Besides the code in our Main file, we imported 7 modules in total, only 1 of which we wrote. The 'pyb' module we imported allowed us to control our microcontroller with python and the 'Pin', 'Timer', 'TimerChannel', and 'ADC' classes. The 'pyb' module also allowed us to enable and disable interrupts with the 'enable_irq' and 'disable_irq' methods. The 'utime' module was used to keep track of the match timer, and to ensure that certain actions took a precise amount of time to complete. The 'gc' module refers to "garbage collector", and it was used to regularly defrag memory in case we had any memory issues. The 'alloc_emergency_exception_buf' module was imported from micro-python and used to allocate memory for the interrupts we used.

The only imported module we wrote ourselves was called 'motor' and allowed us to control the PWM duty cycle of our motors through a 'MotorDriver' class we created. The 'MotorDriver' class also initialized all the pins necessary to control the motors with the 'Pin', 'Timer', and 'TimerChannel' classes from the 'pyb' module. We ultimately didn't end up needing to use either the 'encoder_timer' or 'controller' modules from prior labs. The 'controller' module wasn't used because our final strategy didn't require the motors to have feedback control. The 'encoder_timer' module wasn't used because our encoder only had 1 channel, and 'encoder_timer' required a quadrature encoder. The code for which is shown below.

Besides tasks we also had three interrupts. One was worked with our IR receiver and used a queue called InterruptQueue to talk to our IR receiver task, one tracked edge in our encoder and incremented or decremented a total tick counter, and the other gave us button functionality. All three ran at 65535 Hz because we needed them to run as fast as possible to minimize total system lag.

4: Results

This section will discuss the results of our project and evaluates the effectiveness of our design and execution of that design along with any problems that we ran into along the way.

4.1: Hardware

The biggest fault of our bot was the lack of torque in our motors, which forced us to avoid confrontation instead of seeking out and pushing opponents. Instead, our motors were designed more for speed, which is why we spent most of our matches running away and evading our opponents.

Our edge sensors worked great, which helped us avoid driving off the ring ourselves accidently. Our optical sensors delivered a noisy voltage signal which made it difficult to reliably determine our opponent's distance from us. Additionally, every 1ms our optical sensors sent a pulse that lasted for several hundred microseconds. Our attempt to counteract this issue was introducing a filter shown in the "OpticalSensor1()" code below.

```
def OpticalSensor1():
    '''This function runs every 25ms and reads the optical sensor on the front
    of the SUMO bot and raises a flag if it reads a distance less than 8".'''
    #The front optical sensor, named Charlie
    #Initialize
    pinA4 = pyb.Pin(pyb.Pin.board.PA4, pyb.Pin.IN)
    Range1 = pyb.ADC(pinA4)
    while True:
        #If the bot isn't shut down:
        if not ShutDownFlag.get():
            #Read the optical sensor 4 times at 175 micro second increments
            #This is to account for and ignore a pulse the sensor sends every 1ms
            volt1lst = []
            for x in range(4):
                volt1lst.append((5*Range1.read()/4095))
                utime.sleep_us(175)
            #take the lowest of the 4 values because the pulse is always high
            Volt1 = min(volt1lst)
            #raise the Front_Detection flag if the voltage is high enough
            if Volt1 >= 3.5:
                Front_Detection.put(True)
            yield(0)
        yield(0)
```

The IR receiver we used worked well, but the limited range and angle made it unreliable at times, which is why we implemented our "On" button. Our button, however, was excessively sensitive, and would trigger whenever we gently brushed against it or even just approached it. We tried solving this with a pull-down resistor in the pin we allocated to the button, but this didn't help.

Additionally, it seems that the time-of-flight sensors on many of the bots would interfere with our bots' optical sensors and IR receiver, which made start up very difficult with some opponents.

During assembly and wiring of our bot, we frequently made mistakes that forced us to disassemble and reassemble our bot. In the day leading up to the competition, we accidently shorted one of our very few ADC pins, which required us to ask for a new microcontroller and motor driver expansion. The new set we got worked exactly as intended.

4.2: Software

Our Software worked beautifully, and we are convinced that if we had better hardware and were willing to spend more money, we would have easily won more matches. Many of our problems stemmed from the basic and in some cases faulty hardware we purchased. The only notable bug that persisted on the day of the competition was one where we would execute the avoidance maneuver upon receiving a signal from our front optical sensor and pick up another signal in the meantime and execute it again. Doing this maneuver two times back-to-back generally resulted in our bot going backwards off the edge of the brink.

4.3: Overall

Our bot preformed the way we wanted it to. If we were to change anything in the future it would be to buy better hardware and refine our strategy to accommodate it. Being able to safely move backwards would be ideal.

5: Appendices

Appendix A – SUMO Bot Task Diagram

Appendix B – State Diagrams

Appendix C – Doxygen Code

6: References

[1] http://www.st.com/en/microcontrollers/stm32l476rg.html, pp.92

[2] https://pythonhosted.org/pyserial/

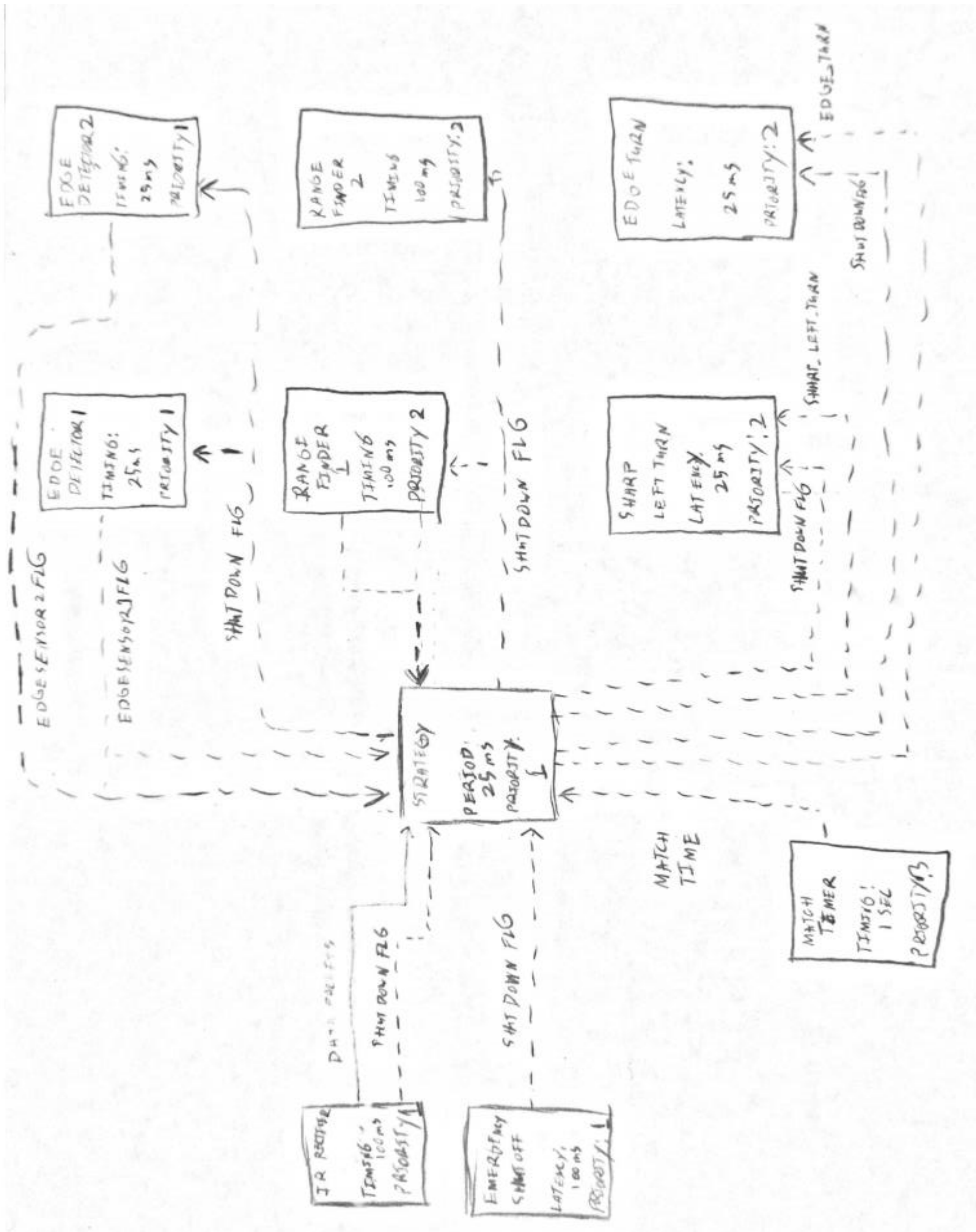
[3] https://docs.micropython.org/en/latest/library/pyb.html

[4] https://docs.micropython.org/en/latest/library/utime.html

[5] ME 405 Code at https://canvas.calpoly.edu/courses/7664/pages/scheduler-files?module_item_id=21289

[6] Our source code at http://wind.calpoly.edu/hg/mecha12

Appendix A: Task Diagram



Appendix B: State Diagrams

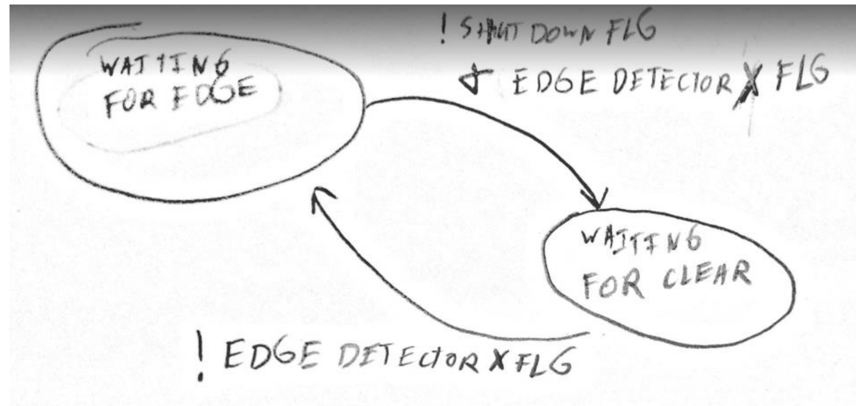


Figure 3: State Diagram for Edge Detection Tasks

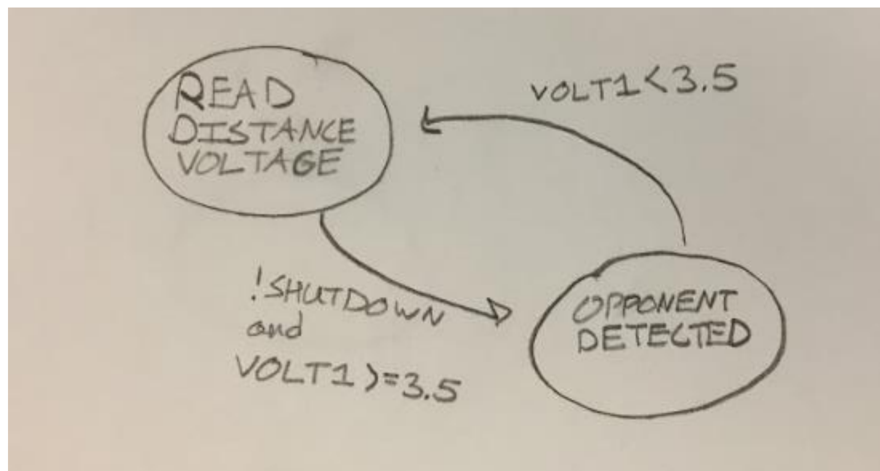


Figure 4: State Diagram for IR Range Finder.

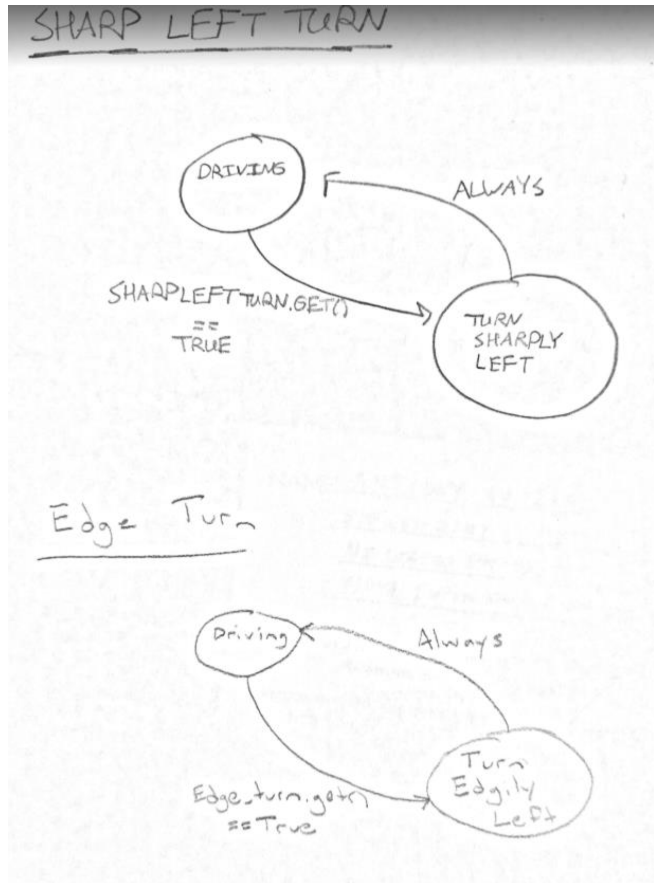


Figure 5: State Diagrams for turning function.

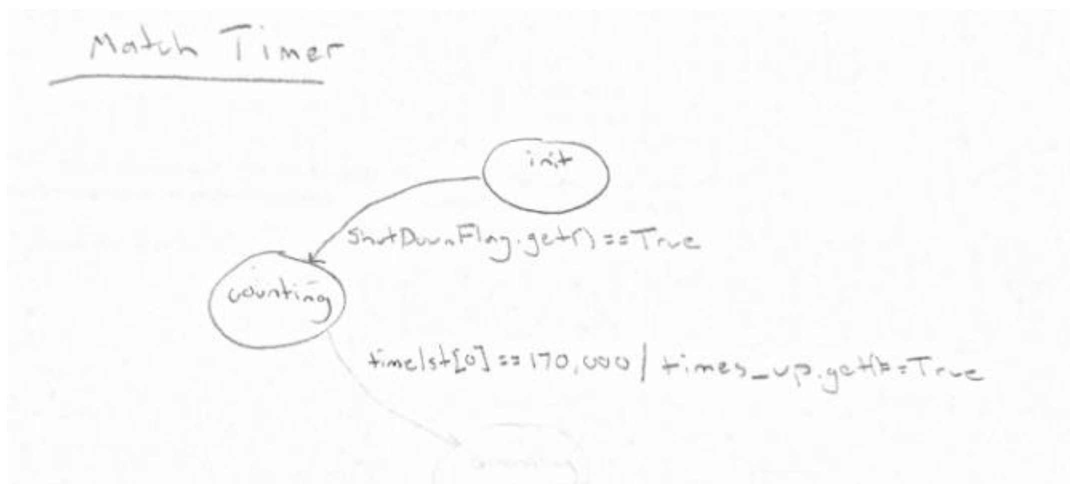


Figure 6: Match timer state diagram.

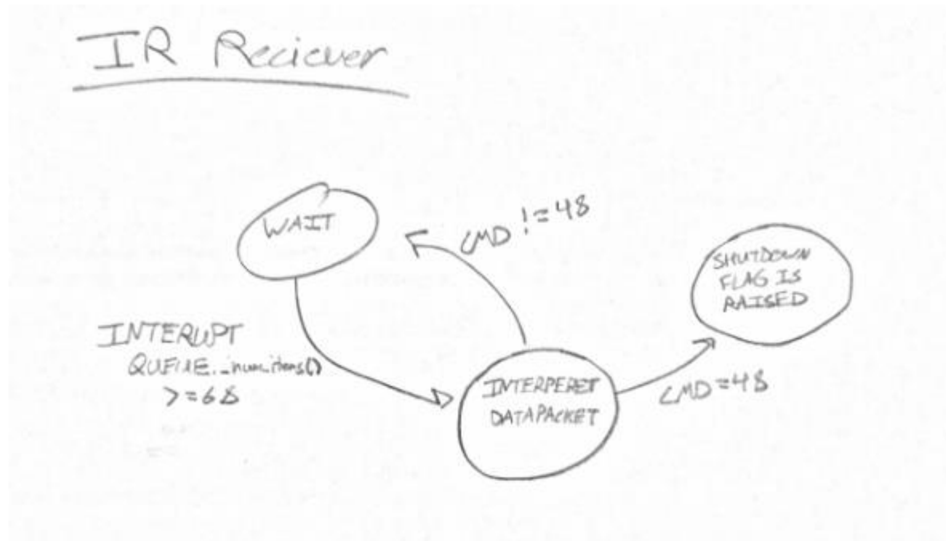


Figure 7: IR Receiver State Diagram