EE175AB Final Report

Kitchen Helper

EE175AB Final Report
Department of Electrical Engineering, UC Riverside

Project Team Member(s)	Sunil Alexander, Richard "Dylan" McGee, Grant Beatty
Date Submitted	March 14, 2022
Section Professor	Dr. Roman Chomko
Revision	Revision 1.0
URL of Project YouTube Videos	https://www.youtube.com/watch?v=ewehUymT7uM&ab_channel=GrantBeatty https://www.youtube.com/watch?v=yvtiDKHStDQ&t=12s https://www.youtube.com/watch?v=M6Imj_xpfE4 https://www.youtube.com/watch?v=NrHi9aHbBfA&t=10s
Permanent Emails of all team members	gbeat002@ucr.edu rmcge002@ucr.edu salex009@ucr.edu

Summary

The contents of this report outline the process of design and development for a simple kitchen helper robot. This document contains details regarding the product's subsystems, their construction, and their integration.

Revisions

Version	Description of Version	Author(s)	Date Completed	Approval
0.1	First Draft — Many Errors	Sunil Alexander, Richard "Dylan" McGee, Grant Beatty	3/9/2022	
0.2	Final Touches and Format Changes	Sunil Alexander, Richard "Dylan" McGee, Grant Beatty	3/12/2022	
1.0	Final Report	Sunil Alexander, Richard "Dylan" McGee, Grant Beatty	3/13/2022	

Table of Contents

1. Executive Summary

The goal of the project was to design and produce a kitchen helper robot that could be of service to the general public. The robot can be implemented to take over various tasks in the kitchen reducing the total required domestic chores of a given household. The project is composed of four primary elements: sensors, which are used to gain information on movement and surrounding objects, motors, which are used to move the device and grab objects, hardware, which consists of several microcontroller units, and software, which is primarily libraries, functions and variables utilized in a finite state machine.

The sensors of the robot function as the primary inputs. The sonic sensors perform two functions: sensing obstacles and sensing proximity to the target object. The IMU module senses the turning direction of the system. The encoder readings sense the movement of the wheels. The camera senses the target object. All of these components allow the kitchen helper robot to accurately traverse through various obstacles in its surroundings and acquire vision of a target object.

The motors are the primary outputs of the system. DC motors propel the wheels of the system and they are connected directly to the custom printed circuit board. The robot arm used to grab objects is propelled by servo motors which are connected to an elevated perfboard. These motors allow the robot to move around and manipulate its environment.

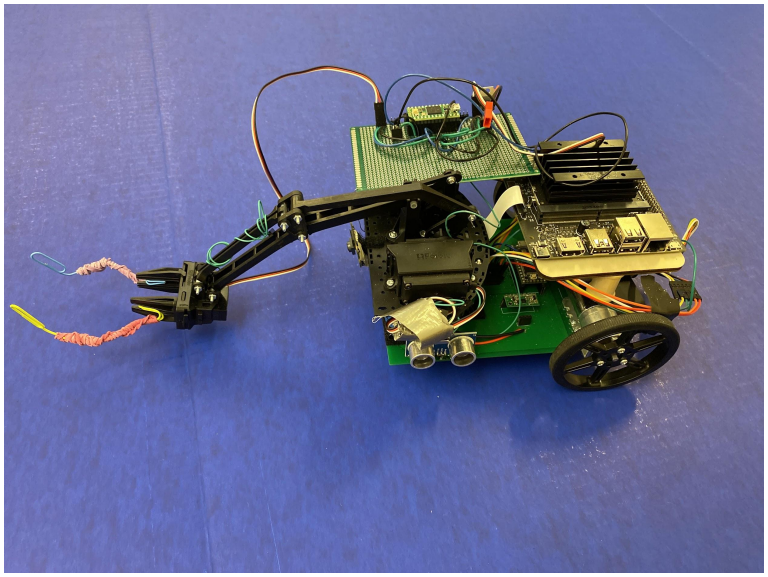
The hardware of the system allows the robot to use the data from the sensors to perform its basic functions in an organized manner. The microcontroller units that are used are two Teensy 4.0 chips and a Jetson Nano. The finite state machine, the controller of the system is contained in the Teensy which is soldered to a custom printed circuit board. The other Teensy controls the robot arm and is soldered to the perfboard (Figure 1.1). The Jetson Nano is connected to a Pi camera module and is used to perform image processing on camera data which allows the robot to identify objects.

The software of the system was c/c++ for both Teensys and Python for the Jetson Nano. One Teensy primarily performs all the main functions. It communicates with the sensors and other microcontroller units to make decisions about how to move around. The industry standard communication protocols used are serial communication through USB and I2C.

Conducting this design project helped us develop a sense for our community. During our brainstorming sessions, we would think about how sometimes in households the function of setting up and taking down kitchen utensils and dishes could sometimes be a hassle and time consuming. This led us to think of how many people would be relieved if an electrical device could somehow alleviate stress in this part of a person's daily life.

Integrating the motors, sensors, software, and hardware together, the kitchen helper robot was created. It has the ability to move accurately, map out its environment, recognize colored objects, and acquire them.

Figure 1.1: Final Design



2. Introduction

The design of the robot was inspired by many of the autonomously moving robots that are used in industries today. It incorporates several aspects of electrical engineering: robotics, power systems, and embedded systems. These aspects of the design are integral to its functionality and allow for significant complexity.

2.1 Design Objectives and System Overview

The objective of the project was to design a robot capable of helping with tasks on the kitchen table. The goal was to create a robotic system that would function as an assistant capable of identifying kitchenware and moving around to set or clear the kitchen table.

We designed the kitchen helper with several technical objectives. The robot was designed to move between square cells in the directions up, down, left, or right. The sonic sensors on the system would need to be able to detect obstacles in the cells beside or in front of the system. The base of the robot needed to fit within a 1 foot by 1 foot square cell. It needed to be able to accurately move straight and turn within an accuracy of 2.3 degrees. When grabbing the test object the robot arm could only be off by 3 cm. The object detection done by the computer vision system had to be as fast as 10 frames per second and be able to send the accurate centroid data to the main system.

2.2 Backgrounds and Prior Art

One of the robots that inspired our design was the Kiva (Figure 2.2.1), an Amazon warehouse robot capable of moving 450 kg shelves and moving at a pace of 5km/h while navigating through a dynamic environment. The robot is only 2.5 by 2 feet and 1 foot high. A thousand of these robots cost 15-20 million dollars. The other robot that influenced our design was the Butter Robot (Figure 2.2.1), created through the collaboration of Adult Swim and Digital Dream Labs, and based on a fictional robot in the cartoon television series *Rick and Morty*. This robot was designed to pass butter across a table.

Figure 2.2.1: Kiva Robot — a warehouse shelf mover

Figure 2.2.2: Butter Robot

2.3 Development Environment and Tools

Hardware Development

For the PCB design we made multiple pcb libraries since we couldn't find parts for them on eagle so we made multiple custom ones such as for the motor encoder connections, the voltage regulator, the motor brackets etc. (they can be seen in the github files in Appendix C) Also for the pcb a top and bottom layer to have a common 5v and GND layer. All pcb design was in Eagle Cad.
To supply power to the system we used an INIU power bank which supplies 5V and has 10,000mAh. It powers the Jetson Nano through a USB cable and the Jetson nano connects to the main Teensy through another USB cable which supplies power and serial communication. A 7.4V LiPo battery, downregulated to 5V, supplies power to the robot arm and the Teensy that controls it.

Software Development

Both Teensy 4.0 chips are programmed with the Arduino IDE using an add-on called Teensyduino. The language the teensy was coded in was similar to c/c++ the arduino ide uses. The CV Python scripts were first developed and implemented in an IDE called Google Colaboratory on pictures taken from an Android smartphone camera. The program was then moved onto the Jetson Nano which uses the Linux4Tegra operating system. The script compiles and runs using Python 3.4 in the command terminal.

2.4 Related Documents and Supporting Materials

USB specifications can be found on this website:
https://www.usb.org/sites/default/files/CabConn_Legacy_3_1_Compliance_Rev_1_1.pdf

I2C specifications can be found on this website:
https://www.nxp.com/docs/en/user-guide/UM10204.pdf

2.5 Definitions and Acronyms

Arduino — open source electronics platform
CPU — Central Processing Unit
GPU — Graphics Processing Unit
I2C — Inter-Integrated Circuit
IEE — Institute of Electrical and Electronics Engineers
IDE — Integrated Development Environment
IMU — Inertial Measurement Unit Module
Jetson Nano — mini-computer with GPU and ARM CPU
MCU — Microcontroller Unit

OpenCV — Open Computer Vision Library
PCB — Printed Circuit Board
Pololu — electronics and robotics store
PWM — Pulse Wave Modulation
SSH — Secure Shell
SUDO — SuperUser DO
Teensy 4.0 — ARM Cortex-M7 microcontroller
Teensyduino — add-on for the Arduino IDE that allows for the programming of the Teensy
UCR — University of California, Riverside
USB — Universal Serial Bus

3. Design Considerations

3.1 Assumptions

The resulting product is designed to be in a very specific environment. Future work could allow the robot to function in many different circumstances.

3.2 Realistic Constraints

3.2.1 Weight and Size Constraints

The Robot chassis needs to be small enough to fit within a 1 foot by 1 foot cell and the weight can not be so much that the DC motors cannot move the load.

3.2.2 Physical Constraints

The Robot must be able to move straight and turn with an accuracy of 2.3 degrees. It should maintain within the 1 foot by 1 foot cells at all times. The robot arm must be able to grab the demo cup with an accuracy of 3 cm.

3.2.3 Project Time and Budget Constraints

The project had a budget of 600$ and a time constraint of 10 weeks.

3.3 System Environments and External Interfaces

The Teensys use the Arduino IDE with the Teensyduino add-on and additional libraries. The Jetson runs with the Linux4Tegra operating system and runs Python 3.4 scripts with additional libraries.

3.4 Industry Standards

This project uses I2C and USB industry standards
I2C allows the main MCU to communicate with the IMU module to acquire directional data.
Serial communication via USB cable enables the Jetson to communicate object centroid data when pinged by the main MCU.

3.5 Knowledge and Skills

The designing and construction of the robot required several branches of engineering such as robotics, control systems, embedded systems, computer vision, electronic circuits, and power management.

The following skills are required: Eagle for PCB design, soldering, wire management, multimeter use for wire testing, programming skills in Arduino, C/C++, Python, and Linux.

Courses and Experience of Group Members

Grant Beatty

EE 1A/B - Engineering Circuit Analysis
CS 13 - Intro to Computer Science for Engineering Majors
EE 120A/B - Logic Design and Intro to Embedded Systems
EE 128 - Sensing and Actuation for Embedded Systems
EE 146 - Computer Vision
Additional Experience:
-Arduino

Skills learned

- Python on Jetson Nano
- OpenCV on Jetson Nano
- Serial Communication
- Additional Linux Commands

Richard Dylan McGee

EE 1A/B - Engineering Circuit Analysis
EE 120A/B - Logic Design and Intro to Embedded Systems
EE 144 - Introduction to Robotics
EE 146 - Computer Vision
Additional Experience:
-Arduino,FSM coding, Eagle PCB design

Skills learned

- Serial Communication
- Additional Linux Commands

Sunil Alexander

EE 1A/B - Engineering Circuit Analysis
CS 13 - Intro to Computer Science for Engineering Majors
EE 120A/B - Logic Design and Intro to Embedded Systems
EE 132 - Automatic Control

Additional Experience:

-Project Management
-Soldering
-Oral Presentation

Skills learned

- C++
- PCB Design
- Project Management
- Subsystem Integration
- Additional Linux Command

3.6 Budget and Cost Analysis

Parts	Quantity	Cost
Jetson Nano	1	$106
sn754410 motor driver	1	$2.95
sonar sensor(HCsr04)	4	$9
prototype breadboard	3	$34.56
wires package	3	$7.98
LSM6DS33 imu	1	$15.95
l7805cv voltage regulator	1	$0.63
servos to make robot arm	1	$27
32 gb Micro SD card	1	$16
INIU power bank	1	$20
Raspberry Pi Camera	1	$25
motor Brackets Pair for 99:1 25D dc pololu motors	1	$7.95
aluminum mounting hub pair	1	$6.95
Wheels Pair	2	$7.95
DIY Cardboard Robot Arm	1	$49.00

KeyeStudio 4 dof arm	**1**	**$56**
pololu arm kit	**1**	**79.95**
bigger dc motors 25D 99:1	**2**	**34.95**
TOTAL COST:		**$663**

Overall the Cost of the robot was a little above 500 dollars, one of the ways we could've cut down costs is to have only one PCB design board manufactured but we made a mistake with two of them and had to order multiple to get a working PCB.

3.7 Safety

While constructing the robot through soldering I made sure to wear a mask to mitigate the solder vapor going into my lungs. Also while drilling the holes for the motor bracket I used proper precautions such as proper gloves, made sure the pcb board in question was held still by a device etc. I Also built into the design of the circuitry for the battery connections we used a JST battery connector in order to make sure the circuit didn't short circuit through accidentally connecting power to gnd and vice versa. Also, for testing, to prevent motors from running prematurely and hurting anyone, a sonar sensor check was made to stop any robot movement until it was ready for testing.

3.8 Documentation

While constructing, coding , and testing the robot after any major construction, or coding test we recorded a video of whatever was just finished. Then we put the video/pictures into a shared google drive. Also for any substantial milestone in the code or pcb design we put it into github in order to revert back to previous commits if need be.

3.9 Risks and Volatile Areas

One volatile area was power supply, if the battery was plugged in the wrong way (as in power to gnd and vice versa) the whole pcb board would be fried and then become useless. The way we mitigated this was to make a pcb library to add a JST battery connector that ensures the battery can only be plugged in the right way.

For the pathfinding the risk was 2 fold one was if the sonar sensor didn't read on an obstacle in a cell correctly and therefore would not see it and then the pathfinding wouldn't know an obstacle was there. The solution to mitigate this was to have the sonar take 10 readings at a time when scanning for obstacles to mitigate that risk. The other was the pathfinding not giving the correct direction given the current state of the cell it's in and it's neighbors. This was mitigated during copious amounts of tests in order to make sure the pathfinding algorithm gave the correct path in multiple different maze obstacle enviroments.

4. Experiment Design and Feasibility Study

4.1 Experiment Design

4.1.1 Sonar Sensor

Dylan was responsible for this task

Objective: Make Sure Sonar Sensors can give good enough readings for objects within a 18” length away from it.

Setup: Code was commented except for the get_sonar_dist function and Serial.Prints were used with the Serial Monitor.

Procedure: Have the handmade block obstacles moved in increments of 4 inches and look at the Serial monitor to see if distance readings changed accordingly.

Expected Results: The lower the reading the closer the object was the higher the reading the further away the object was.

4.1.2 Basic Motor Control

Dylan was responsible for this task

Objective: To make sure the dc motor pair can do the basic traversal functions such as forward,forward_w_speed,reverse, halt, left_turn and right_turn

Setup: Had code running in a loop that went forward , forward_w_speed with left motor going faster than the right , then forward_w_speed with right motor going faster than the left, then left_turn and right_turn. Between all of these halt was used/tested to make the differences.

Procedure: Have this code running and observe that the robot does the expected motor functions at the correct order as in the code.

Expected Results: Each motor traversal functions works as expected such as forward going forward, halt stopping the robot, left_turn the robot turns left etc.

4.1.3 IMU PID Controller

Dylan was responsible for this task

Objective: To see if using a IMU is a good enough basis for the going straight PID controller

Setup: Set up the PID controller class to use the imu_angle as the current value and assign it relevant setpoints,kp,kd etc.

Procedure: Make robot only do pid forward function and then see if it goes straight

Expected Results: Robot should go straight.

4.1.4 Differential Velocity PID Controller

Dylan was responsible for this task

Objective: To see if using a differential velocity pid controller is a good enough basis for the going straight PID controller

Setup: Set up the PID controller class to use the difference in velocities between the motors as the current value and assign it relevant setpoints,kp,kd etc.

Procedure: Make robot only do pid forward function and then see if it goes straight

Expected Results:Robot should go straight.

4.1.5 Concurrent Separate Velocity PID Controller

Dylan was responsible for this task

Objective: To see if using a IMU is a good enough basis for the going straight PID controller

Setup: Set up multiple PID controller classes to use the velocity for the respective motor(left or right) as the current value and assign it relevant setpoints,kp,kd etc. (as per the biases of the specific motor), Then add the error to a global left_pwm and right_pwm values to adjust the motor speeds

Procedure: Make robot only do pid forward function and then see if it goes straight

Expected Results:Robot should go straight.

4.1.6 Pathfinding Simulation

Dylan and Grant were responsible for this task

Objective: In order to make sure the pathfinding algorithm to be used (Treamux) would solve the 5x5 maze with any given obstacle placement we need to run the pathfinding simulation code.

Setup: Have the Treamux Pathfinding algorithm, Serial print the maze and the current states of the cells in the maze including where the robot currently is.

Procedure: Input different obstacle configurations and make sure the Tremaux algorithm outputs a path from the starting cell to the maze and also make sure the algorithm correctly changes the visited number for each coord (the amount of times the coord has been visited) whenever it goes to that coordinate

Expected Results: To have the Tremaux algorithm simulation for any obstacle configuration given there's an opening to the start and goal find a path.

4.1.7 Computer Vision Methods

Grant was responsible for this task

Objective: In order to construct a proper object detection algorithm for the Pi camera and Jetson Nano we tested several methods of object isolation with OpenCV.

Setup: Use Google Colaboratory to run various built in CV algorithms: SIFT, SURF, ORB, Contour Matching, and Color Filtering.

Procedure: Take several picture samples and test to see if they are able to utilize the algorithms. Display matched keypoints and/or isolated pixels to determine if the algorithm is working properly.

Expected Results: The CV algorithms should be able to isolate the target object and calculate its centroid data.

4.1.8 Camera Test

Grant was responsible for this task

Objective: The Jetson Nano requires a compatible camera and so a suitable one must have confirmed functionality.

Setup: Gather the Jetson Nano and the USB and Pi cameras for testing.

Procedure: Test to see that the cameras can connect and function properly using the built-in streaming app then test with a python script.

Expected Results: The camera should be able to stream and frame data must be accessed through a python script.

4.1.9 Subsystem Communication Teensy to Jetson Nano

Dylan and Grant were responsible for this task

Objective: Make sure the Serial communication works properly and that the nano can send a char array of 8 chars starting with 'a' and ending with 'f' (to denote the end of a transmission) and that the teensy can convert the 6 chars in between to 2 ints in the hundreds

Setup: First attach the USB serial cable to the teensy and Jetson Nano.
Have the Nano just send continuously the char array of ‘a’,’1’,’0’,’0’,’3’,’3’,’1’,’f’ and then set up the code on the teensy so that if the int conversions come out to row_number = 100, and col_number = 331 to have the robot move forward.

Procedure: Setup the connection and run the python script of the Jetson nano and see to watch if the base robot moves forward starting from a halt position.

Expected Results: To have the base robot move forward.

4.1.10 Arm Selection

Sunil was responsible for this task

Objective: An arm that is small enough to fit on the chassis yet big enough to grab a cup.

Setup: Build various arms to see if they fit our standards.

Procedure: Connect the arms to the circuit Sunil made with the arduino attached that had arm code to test to see if each servo was working.

Expected Results: An arm that could properly fit onto the chassis and be able to pick up a cup.

4.1.11 Arm Functionality

Sunil was responsible for this task

Objective: To find the correct angles positions the servos have to be placed so that the main robot knows what distance to be in order for the arm to pick it up perfectly.

Setup: I (Sunil) built a breadboard setup that had 2 joysticks attached to it and an arduino with code that I wrote.

Procedure: The code functioned so that when I moved the joystick, the servos would move the arm and the angle values would print on the screen. So I attached the arm to the chassis and had Grant read me the values from the computer and he wrote down the values.

Expected Results: When the robot is in the right position to pick up the cup, the arm should be able to pick it up with no problems.

4.1.12 Battery Selection

Grant and Sunil were responsible for this task

Objective: A battery that could adequately power the Jetson Nano as well as the main Teensy and motors was required.

Setup: Gathering the nano attached to the chassis along with the batteries.

Procedure: Plug in the various power sources and see which ones work

Expected Results: Optimally the smallest battery tested is able to power the systems.

4.1.13 Chassis Construction/Mounting Methods

Objective: Figuring out an optimal way to construct the chassis to create equilibrium and provide versatility.

Setup: I bought velcro and wooded platforms to help attach various parts to the PCB

Procedure: I experimented with different wrapping and placement techniques for the arm, jetson nano, and perf boards. I made longer spokes on the robot arm and wrapped the bottom in velcro and stuck it to the front of the board. I created a wooden platform that would sit right on top of the motors in the back of the robot and stuck the Jetson nano on it. This allowed for the camera to be able to go through the bottom of the arm and mounted with velcro on the front of it so we could change the orientation if need be.

Expected Results: A chassis that could be taken apart without damaging any components as well as being stable enough to have optimal PID control testing.

4.2 Experiment Design , Data Analysis, and Feasibility

4.2.1 Sonar Sensor

Results:The sonar sensors worked splendidly and gave lower distance values when the obstacle was close and higher values when the object was close , The sonar gave readings in the 100-950 depending on the range an obstacle was away, and if it was too far away the readings were in the 10,000’s

Feasibility: Yes.

4.2.2 Basic Motor Control

Results: After some initial debugging the motors did all of the basic motor traversal functions in the correct order

Feasibility: Yes.

4.2.3 IMU PID Controller

Results: The IMU PID controller would go straight for 1 12” cell but after the accumulated error drift was too much and it wouldn’t go straight for more than 4 seconds straight consistently

Feasibility: No.

4.2.4 Differential Velocity PID Controller

Results: The robot did not go straight

Feasibility: No.

4.2.5 Concurrent Separate Velocity PID Controller

Results: The robot went straight, the reason this worked is because even if both motors are trying to get to similar velocities the PWM signal required to both motors is different due to minute differences in the motors themselves.

Feasibility: Yes.

4.2.6 Pathfinding Simulation

Results: The Simulation Code had the robot found a path to the goal from the start for a variety of hardcode obstacle coordinates.

Feasibility: Yes.

4.2.7 Computer Vision Methods

Results: All of the methods were unsuccessful except for the color filter which was marginally successful.

Feasibility: Yes (for color filter).

4.2.8 Camera Test

Results: The USB camera worked but there was no easy method to access the camera data from a Python script so this option was ruled out. The Pi camera that was a success was the Pi camera V2.

Feasibility: Yes (for Pi camera).

4.2.9 Subsystem Communication Teensy to Jetson Nano

Results: The robot went forward after receiving the char array.

Feasibility: Yes.

4.2.10 Arm Selection

Results: We bought an arm that I made an extension for that was successfully able to pick up the cup.

Feasibility: Yes

4.2.11 Arm Functionality

Results: The values that were acquired from this didn't have to be changed and picked the cup up multiple times, so we obtained successful arm functionality.

Feasibility: Yes

4.2.12 Battery Selection

Results: The regular phone charger power bank worked for the Jetson Nano but it did not adequately power the other systems. However, the INIU power bank was successful at powering them.

Feasibility: Yes

4.2.13 Chassis Construction/Mounting Methods

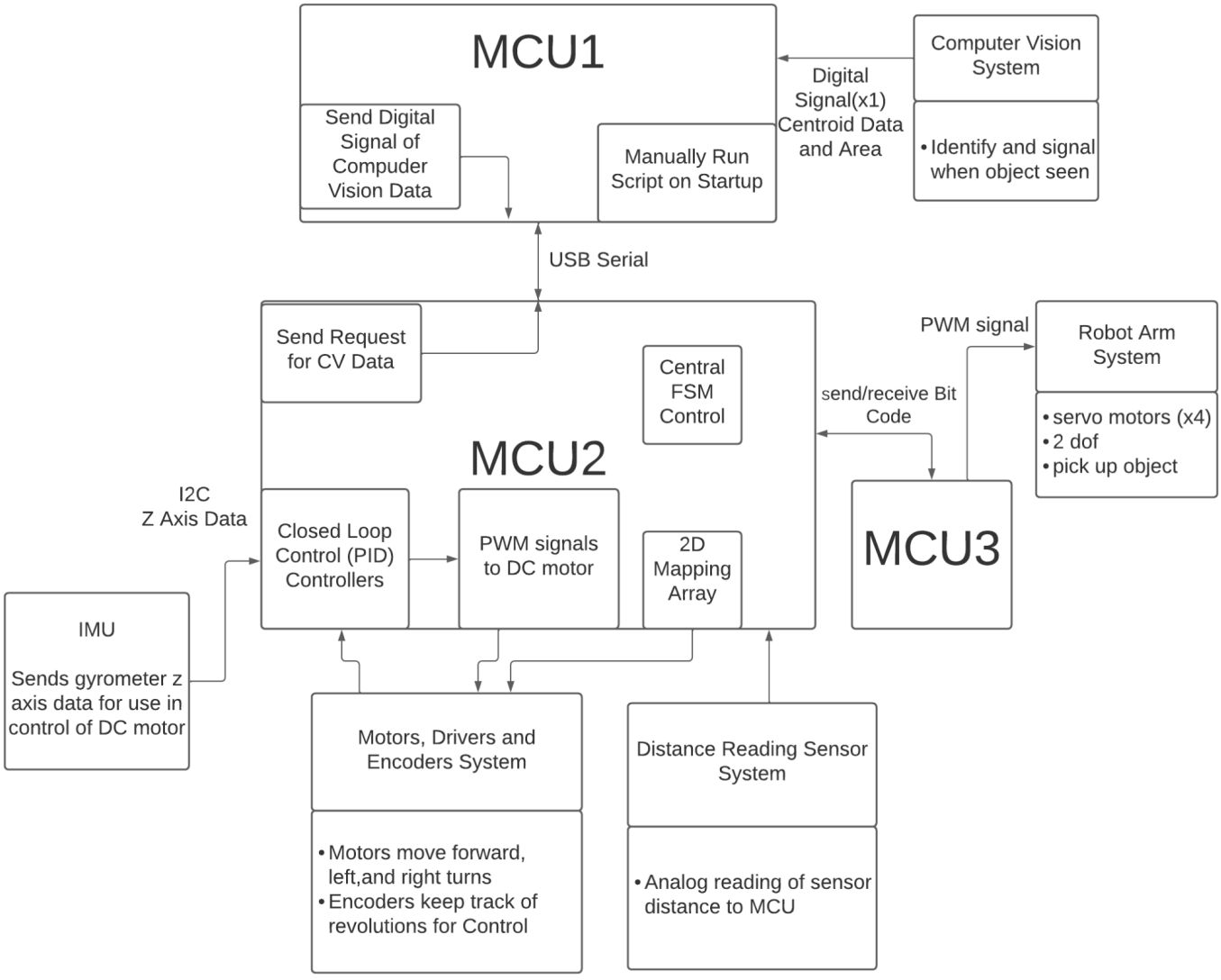
Results: There were many times that we needed to inspect the PCB and having the ability to take different parts off with ease was very important. Our chassis performed very well considering it was able to accurately navigate through objects, locate the cup, and pick the cup up.

Feasibility: Yes

5. Architecture and High Level Design

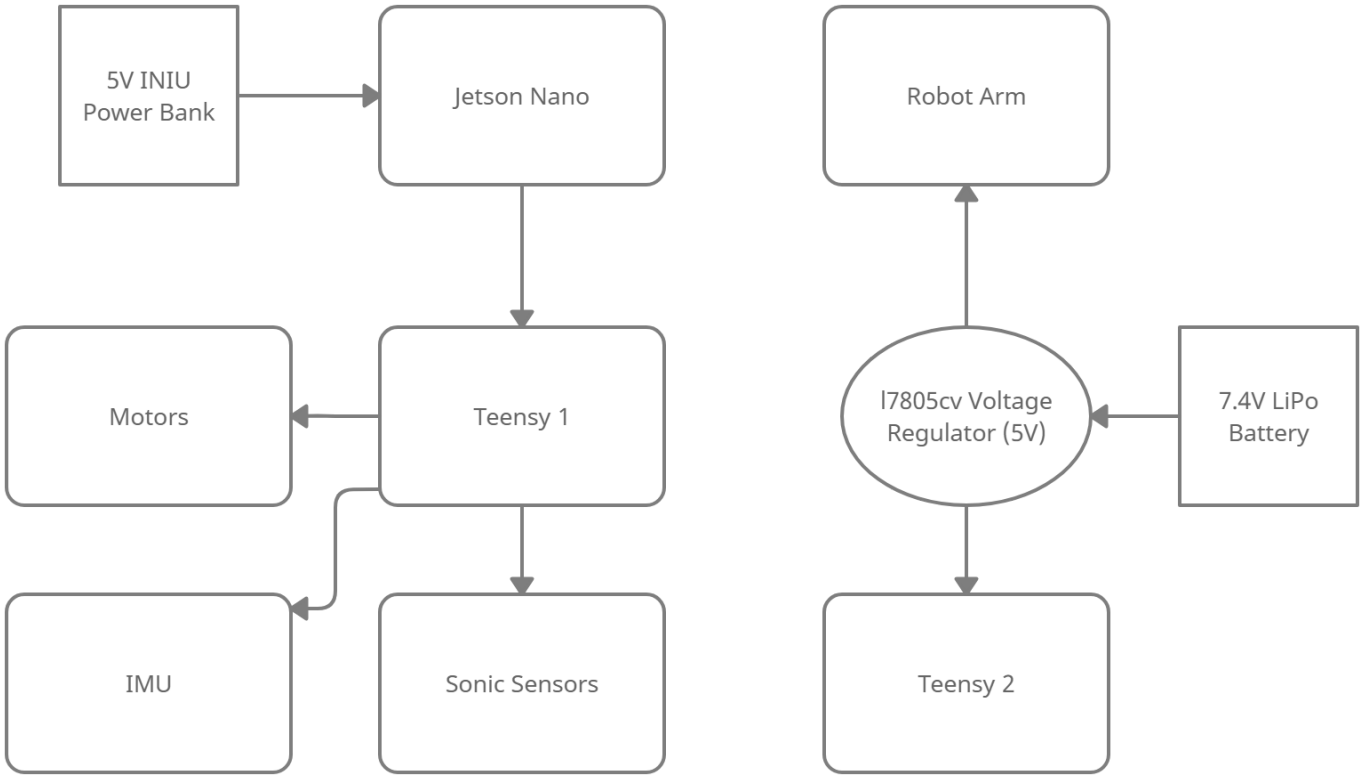
5.1 System Architecture and Design

Figure 5.1.1: System Block Diagram



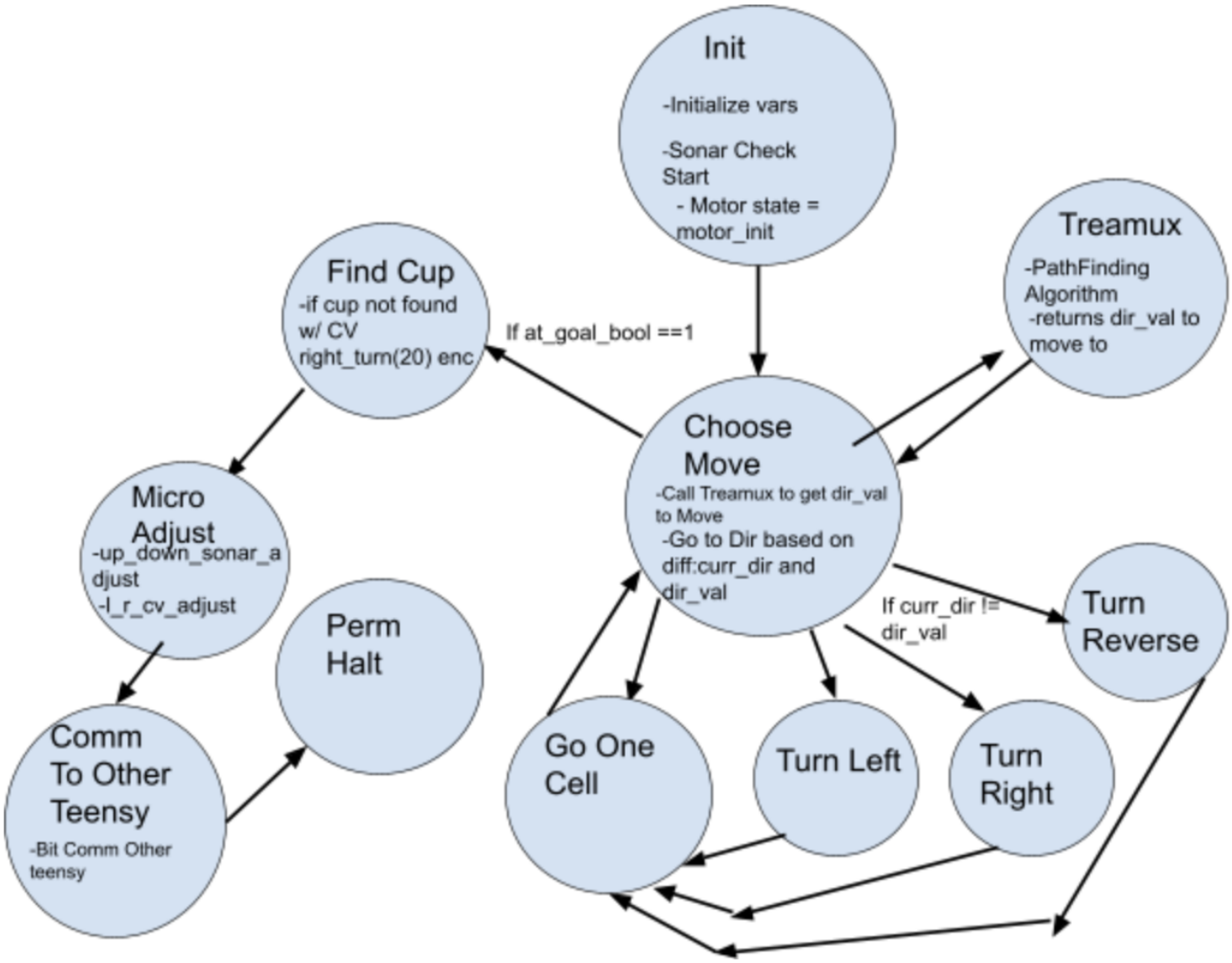
5.2 Hardware Architecture

Figure 5.2.1: Hardware Block Diagram



5.3 Software Architecture

Figure 5.3.1: Finite State Machine



5.4 Rationale and Alternatives

Figure 5.4.1: Battery Analysis

	INIU Power Bank (10000mAh)	Cell Phone Power Bank (2200 mAh)
Cost	20$	6$
Weight	0.32 kg	0.08 kg
Dimensions	13.2 x 6.9 x 1.3 cm	9.5 x 2.2 x 2.2 cm
Power	(3A)(5V)	(1A)(5V)

Figure 5.4.2: CV MCU Analysis

	Jetson Nano (2 Gb Model)	Raspberry Pi 4 (2 Gb Model)
Cost	200$	130$
Processing Power	1.43 Ghz 64 bit quad core	1.5 Ghz 64 bit quad core
RAM	4GB with 1600 MHz	4GB with 1600 MHz
Power	5W-10W	2.56W-7.30W
Wi-Fi	not built in	built in
GPU	128 core GPU	no GPU

The reason we used the approach of having the teensy be the main computer is that we had not figured out how to integrate the Jetson Nano with the main Teensy until a couple of weeks before the deadline and most of the code that integrated the IMU module, motors, and sonic sensors was already on the Teensy.

In hindsight we also should have used a Raspberry Pi over a Jetson Nano. Since we thought we would use more machine learning centric approaches, the Jetson Nano's extra processing power and GPU made it appear to be the superior option but we ended up using a more fundamental OpenCV approach so the extra computational power was unnecessary. Also if we had used the Raspberry Pi we could've used a smaller cheaper usb cellphone battery instead of the bulkier INIU power bank battery since the Raspberry Pi requires a smaller power supply then the Jetson Nano.

6. Data Structures

Base Robot Teensy (Dylan)

senior_design_proto_5.ino

- The main file for the main teensy code, this imports all of the other required header files that the robot uses to run, also includes the setup which sets up all the pins and sets up the starting state for the main FSM. Then all that's left in the main file is to call the motor_tick() function to call the FSM

global_values.h

- This includes all of the variables used in the main robot program. It's in a separate highest priority header file to make sure the variables here can be used in any file of the robot

gyro_func.h

- This includes all of the gyro functions such as the update_gyro function and the complementary filter used with it as well. All of this is to get the angle of the robot during turning functions

misc_functions.h

- This includes misc functions that don't really fit anywhere else in the code. This mostly includes pid functions and pid controller classes that were not used in the final build.

motor_func.h

- This is a really important part of the entire code. Not only does it have all of the traversal functions such as forward, halt, reverse etc. It also includes the working pid controller class of the separate concurrent velocity for the left and right motors, the encoder_pid function that calculates the velocity and then calls the pid controller class and then ads the P_term and D_term to the left and right pwm values, it also includes the go_one_cell function (which is the movement function that makes the robot go approximately 1 12" cell length,also it has both of the micro adjust functions for the sonar adjust and the cv left and right adjust ,and also includes the entire FSM which handles what the entire robot does and when.

sonar_func.h

- This includes all of the functions used to get the distances values from the sonar, it also includes a separate get_sonar_dist_test function that has a delay and prints out the values to the serial monitor for use if we need to test if the sonars work real quick instead of commenting and uncommenting the delay and prints in the main get_sonar_dist function

treamux_func.h

- This header file handles the entire pathfinding algorithm, so it includes the custom struct class for the coordinates (coord in the code) that has the y and x coordinates , the visited_num int (the number of times a coord is visistied. It also includes the functions of neighbors_func which when given a coord returns the 4 neighbors to it the up, right , left and down neighbors in that order. In the main treamaux_func first looks for obstacles with the sonars to set the coord struct as inaccessible. Then it does the pathfinding algorithm with all available neighbors and then returns the optimal direction as per the algorithm. The entire main function is called once in the CHOOSE_MOVE state of the main FSM.

CV Jetson Nano Code

ReduceSizeAndPad.py

- This function takes the frame array and pads the outside of the frame as well as reduces the size of the frame using built-in functions.

simplified.py

- This function applies a color filter on the frame data so that the only pixels highlighted on the frame are specific shades of red, green, and blue.

IsolateObj.py

- This function uses connected component sequential labeling to identify all of the objects in the frame. Objects that do not meet the size threshold are filtered out as well as objects that do not fit the shape criteria. The centroid of the largest object is calculated and returned.

six_chars.py

- Splits the centroid data into a six char array.

serial_test.py

- Calls six_chars to format the centroid data then sends each digit 1 byte at a time through serial communication.

test2.py

- The main file gathers frame data from the camera at 10 frames per second and combines all of the above CV functions. It sends the centroid data of the target object if a ‘z’ character is sent by the Teensy through serial communication.

Robot Arm Teensy

grant&sunil's_servo_code.ino

- This code runs the robot arm grab function when it receives a high signal from the other Teensy. It also sends a signal back when it is done.

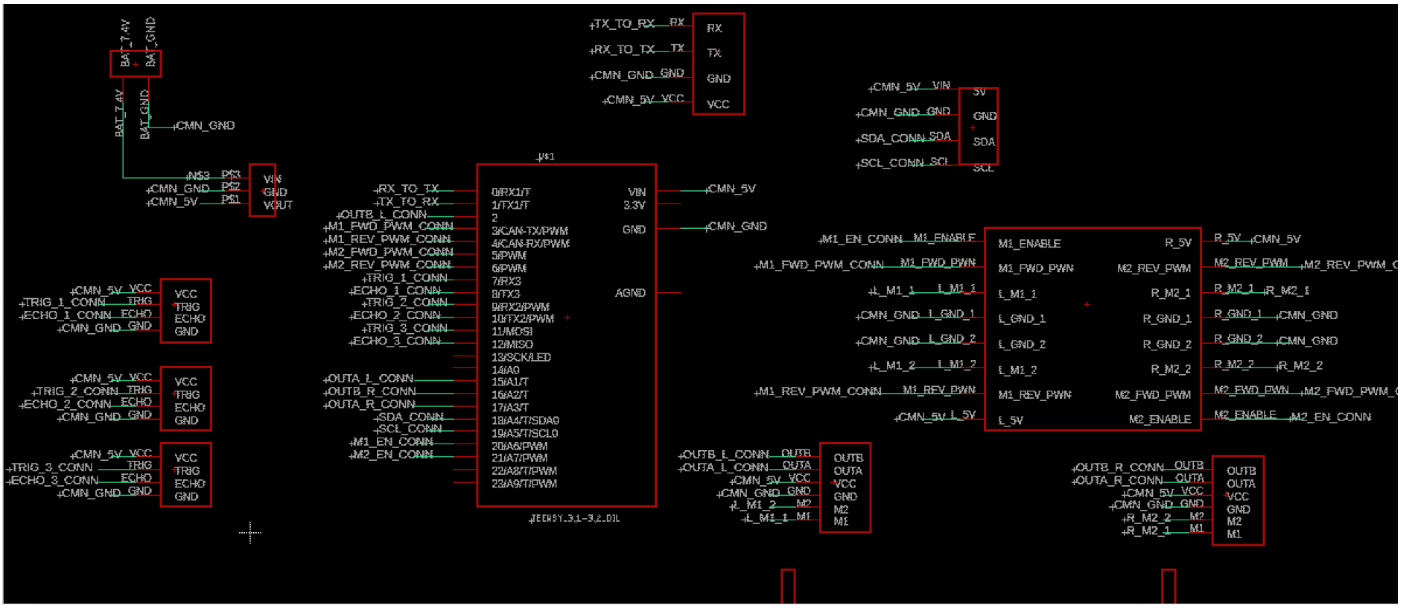
7. Low Level Design

Dylan performed the low level design

7.1 Base Robot Circuit

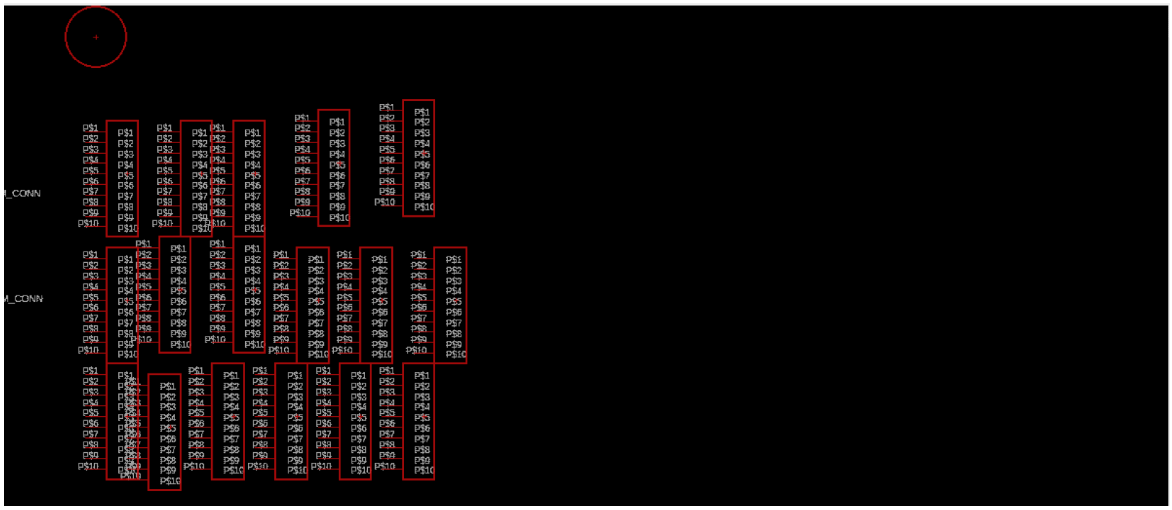
- This is the base robot circuit for the kitchen helper robot

Figure 7.1.1: PCB Circuit Schematic



- This is the main circuit involving the sonars, power regulator to battery connection, the teensy 3.2 used as the main MCU, the HC-05 bluetooth module, the IMU using I2C connection , the motor driver , and both motor encoder pins headers. In order to make the schematic easier to debug I used label connections so if I wanted to connect the TRIG_1_CONN from a sonar sensor to Pin 7 of the teensy.

Figure 7.1.2: Main MCU Circuit

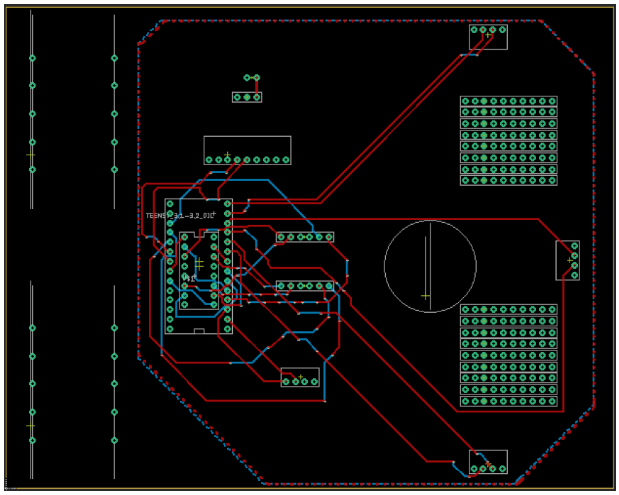


- This is the test rows that I used while testing different components without having to change the whole pcb board and above that is the circuit symbol for the space for the ballcaster of the robot

7.2 Base Robot PCB

- This is the base robot PCB design that I made for the kitchen helper robot

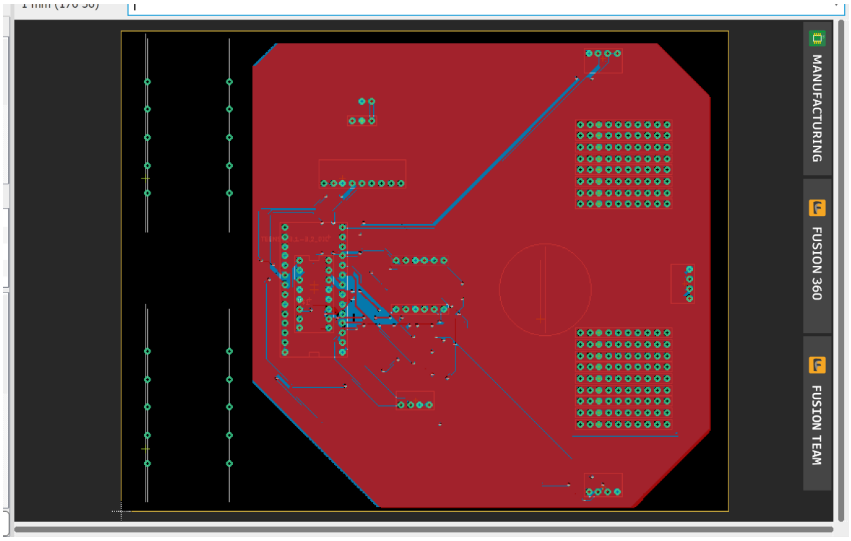
Figure 7.2.1: PCB Circuit Board



-This is the PCB that was made for the robot. It includes all of the trace connections between all of the components. It also has 2 layers of connections the top red layer was the one used the most but the bottom blue layer easy used if the top layer was too full for a trace so vias were used in order to make those connections happen. Also 2 polygon layers were used the top layer of Common 5V was red and the bottom layer of the Common Gnd was blue. These layers were used in order to cut down on the amount of traces needed; that way there doesn't need to be a common gnd and 5v trace throughout the entire board.

- Also as you can see I put the SN7 motor driver beneath the teensy to save space since it could fit in heighwise beneath the teensy
-here's a picture of the gnd and power layers when ratsnested so it's clear to see where the 5V and Gnd layers are (they mostly cover the same area so the blue is covered by the top red layer)

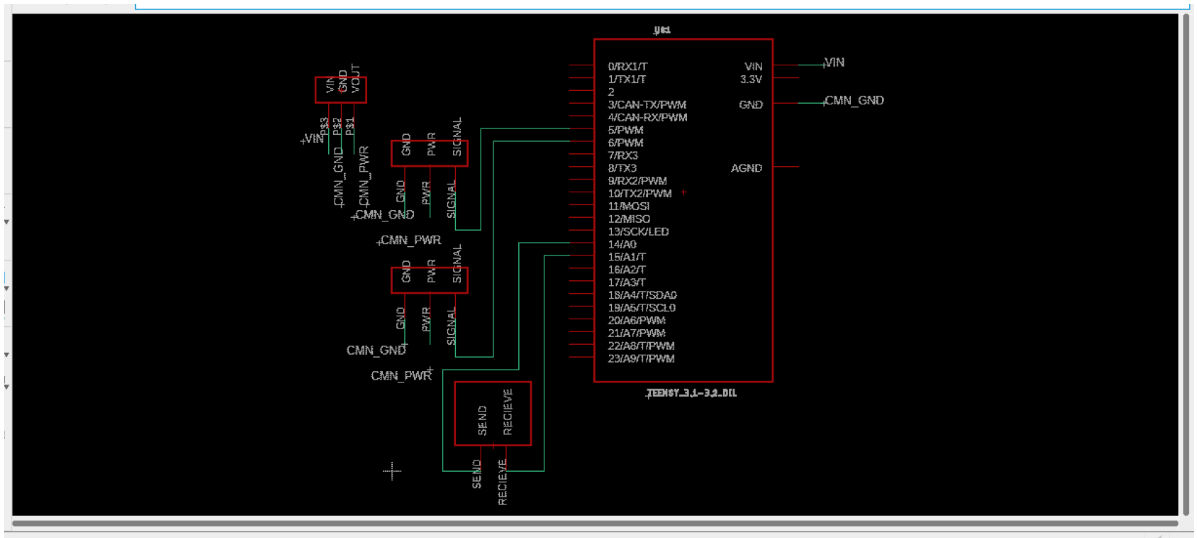
Figure 7.1.1: PCB Ground and Power Layers



7.3 Arm Connection Schematic

- This is the arm connection schematic used to make the perboard that was used for the arm teensy.

Figure 7.1.1: 2nd Teensy Circuit Schematic

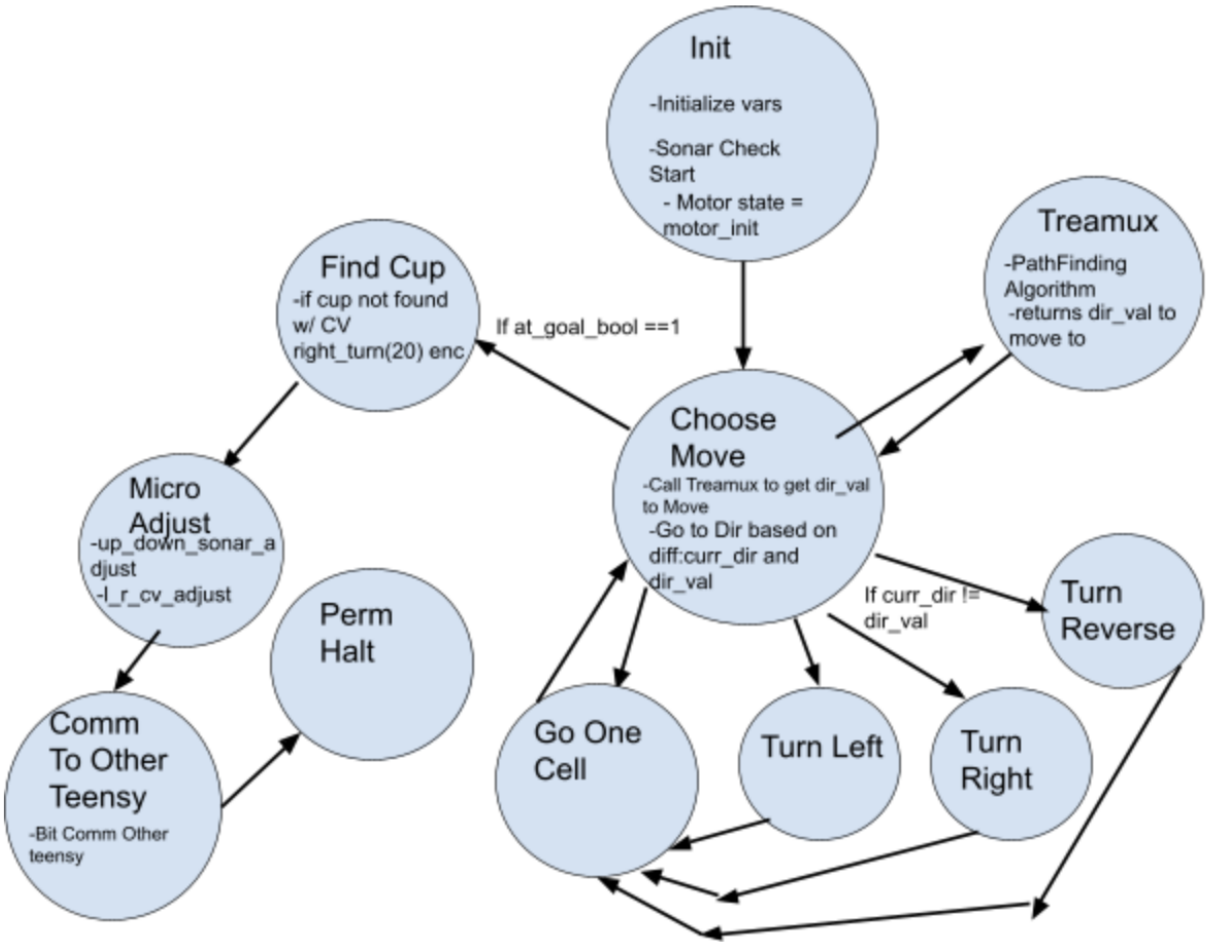


- The arm connection was mostly just having a common 5v and gnd for all of the servo connections , and connecting the signal from the servo connection to a pwm pin of the teensy. The

only other thing was to make a send receive pin symbol for the custom bit connection that our robot used to communicate between teensy's

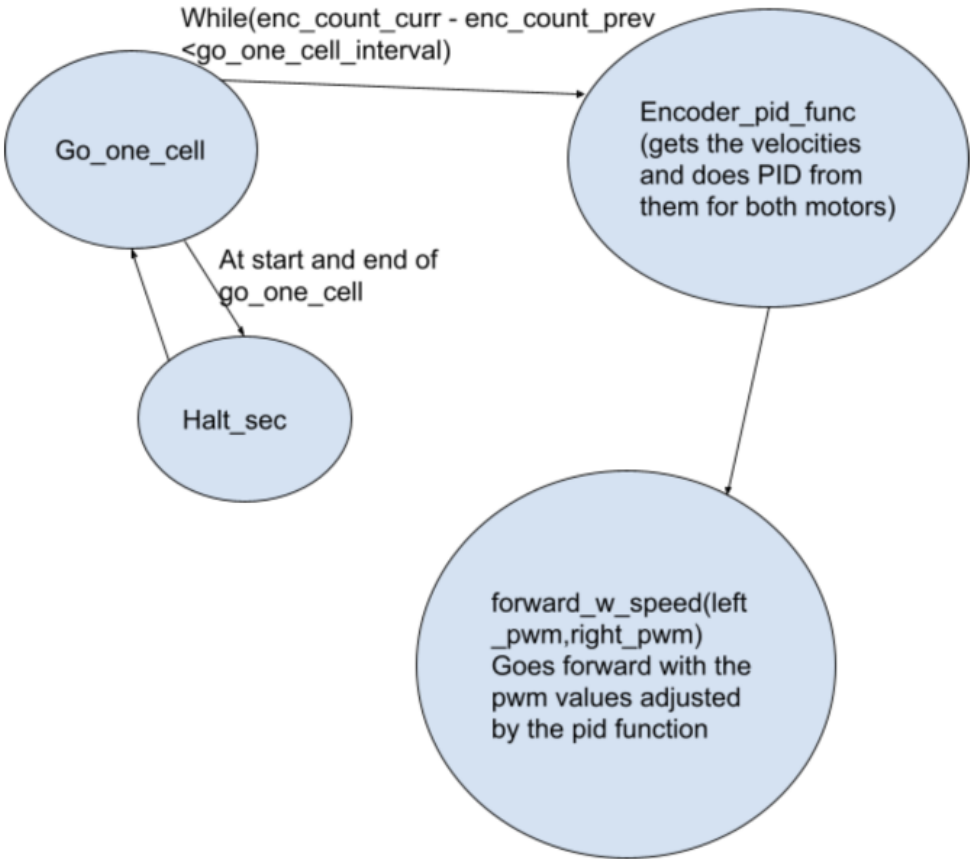
7.4. Total FSM Flowchart

Figure 7.4.1: Total FSM Flowchart



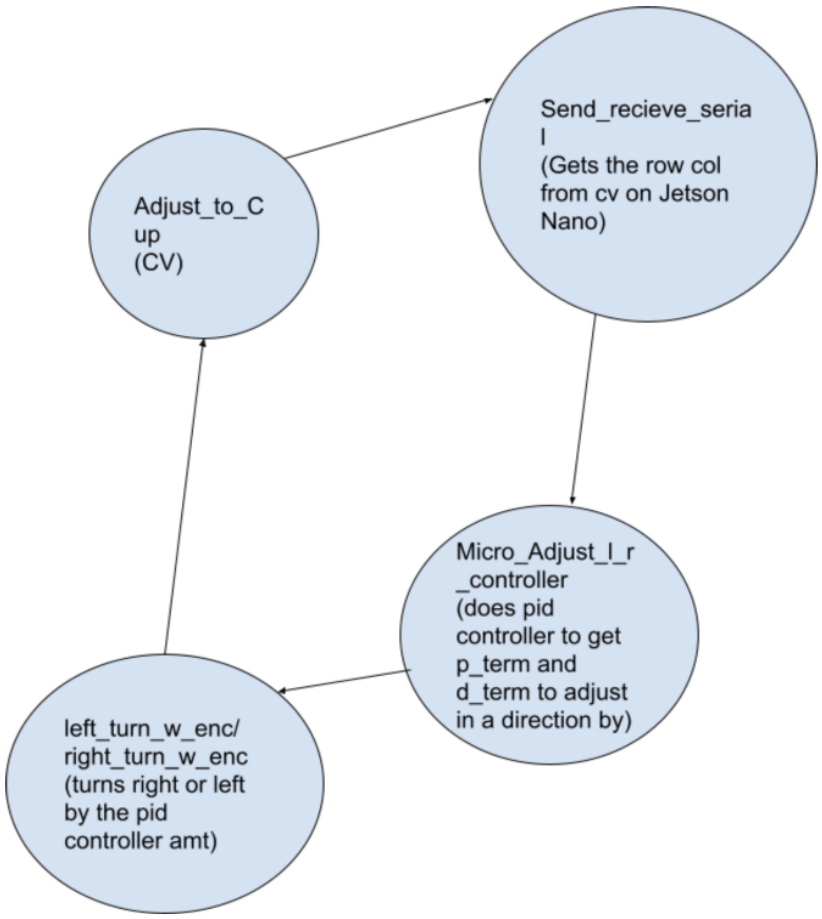
7.5 Go_One_Cell Flowchart

Figure 7.5.1: Go_One_Cell Flowchart



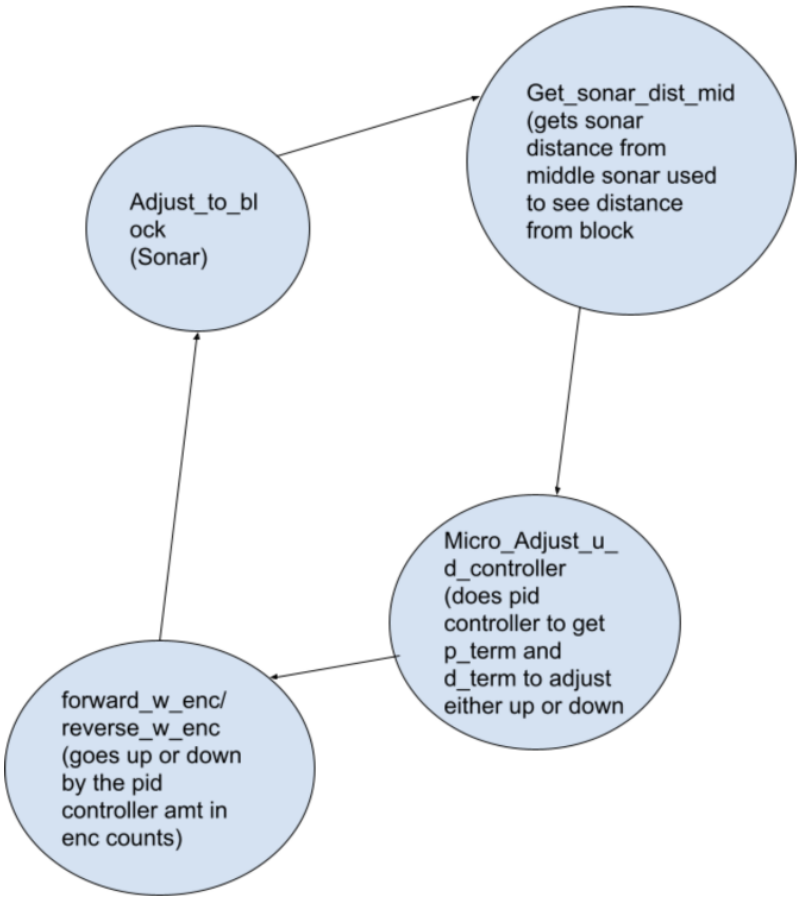
7.6 CV Micro Adjust Flowchart

Figure 7.6.1: CV Micro Adjust Flowchart



7.7 Sonar Micro Adjust Flowchart

Figure 7.7.1: Sonar Micro Adjust Flowchart



7.8 Tremaux Flowchart

Figure 7.8.1: Tremaux Flowchart

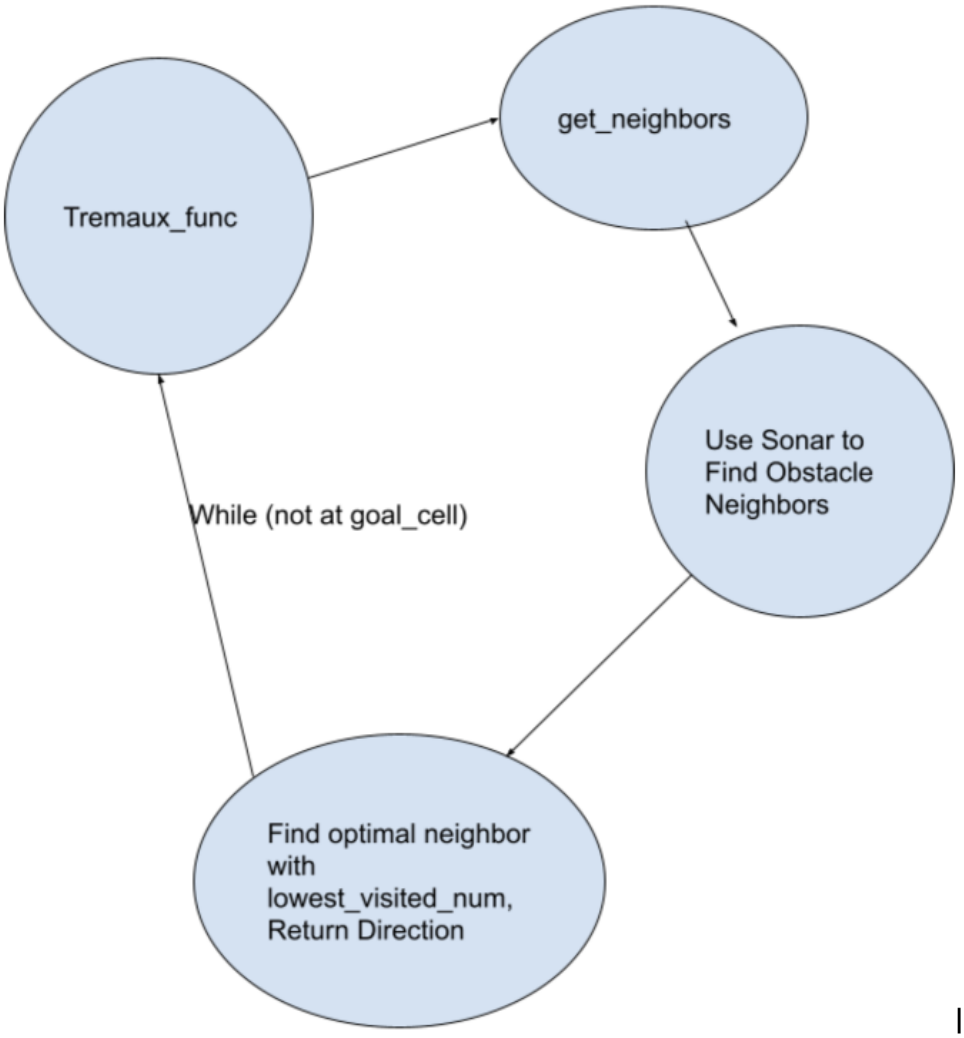
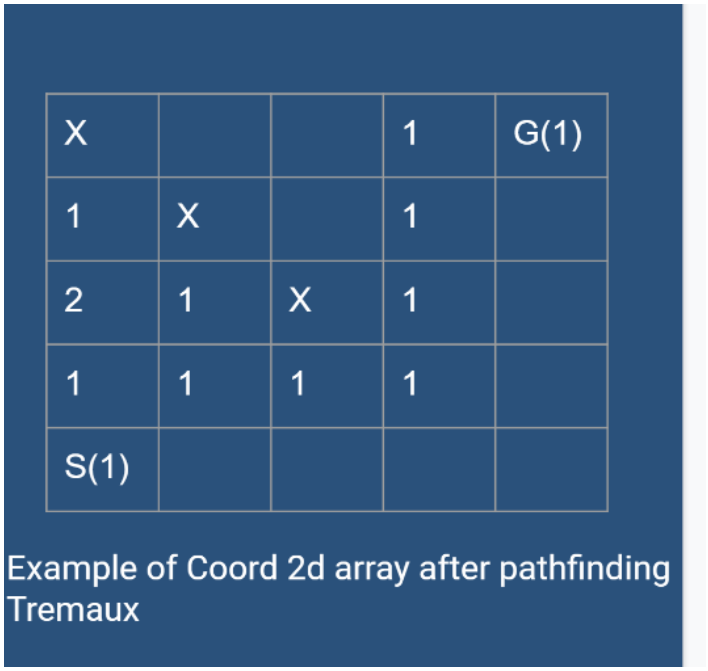


Figure 7.8.2: Tremaux Algorithm Example



7.9 Extraneous Explanation

- For a lot of the things in the PCB and schematics above I had to make my own custom PCB library for those parts. All the custom PCB library parts used can be found on my(Dylan) github in appendix C

8. Technical Problem Solving

8.1 Problems

8.1.1 Sonar Distance Issues

Dylan was responsible for this task

The Sonar would sometimes give off crazy high values and even in instances in where the reading before and the reading after it are the same sometimes the reading in the middle are high reading values out of nowhere

8.1.2 Going Straight PID Issues

Dylan was responsible for this task

Finding the right pid controller was an issue since for the longest times the multiple pid controller I coded did not work in terms of making the robot go straight. I settled on the separate concurrent

8.1.3 Pathfinding Issues

Dylan was responsible for this task

Choosing an appropriate pathfinding algorithm was a challenge I wrote implementations not just for Treamux but for A* search and flood-fill as well. But there was issues with them due to me making their implementation not only in c++ but also in an embedded system environment which disallowed certain ways to code them such as recursion for floodfill due to the limited stack space on the microcontroller.Also there were difficulties integrating these algorithms into the state machine and also integrating a find new obstacles and put them into an obstacle array function

8.1.4 Accumulated Traversal Error

Dylan was responsible for this task

Since there’s no definite walls to do PID off of (such as taking the sensor difference between 2 parallel walls that the robot is in the middle of) accumulated error happens. So if there’s any small difference in terms of going straight or the correct encoder count. While the robot can traverse up to 40 or so cells it can stay straight and also still be inside the correct cell area any more leads to more and more accumulated error.

8.1.5 CV Color Problem

Grant was responsible for this task

I decided to use a color filter to isolate the object. Determining the thresholds for the colors was a hard problem to solve. The RGB color values could not simply be guessed. I had 3 kinds of cups that I wanted to isolate 1 red, 1 Green, and 1 Blue.

8.1.6 CV Object Noise

Grant was responsible for this task

After the color filter was working there was still some additional noise left in the processed images as well as background objects. This caused the centroid data to not accurately represent the center of the objects that I was trying to isolate.

8.1.7 First Iteration Arm Issue

Sunil was responsible for this task

We had initially had an arm called the meArm which included 4 servos that had 4 different functions. A gripper function, a turn function, an extension and retraction function, and an up and down function. I bought 2 variations of this arm, one made out of wood and one made out of aluminum. This arm was good but I decided that it had too many servos and the flat bottom would make it hard to mount.

8.1.8 Chassis Issue

Sunil was responsible for this task

Our chassis had multiple mounting issues that I had to think about. Our Jetson Nano, robot arm, arm teensy circuit board, and battery didn't have proper mounting ideas in order to keep it balanced and PCB components easily accessible.

8.1.9 New Arm Gripper Issue

Sunil was responsible for this task

The new arm we acquired did not have a gripper wide enough to fit the cup we were going to use. We could try to fashion a smaller object to pick up but the demonstration would communicate our proof of concept more if we could pick up an actual cup.

8.2 Solutions to the Problem

8.2.1 Sonar Distance Issues

To fix this I took the average of several readings for the final distance reading used , also I made a low pass filter in code in order to filter out absurdly high sonar values out of nowhere

8.2.2 Going Straight PID

I fixed this by using the finalized 2 separate concurrent velocity controllers as specified in section 4. Also I found out the best way to get the velocity was to use the double data type and also divide the difference in encoder counts of current - prev by a constant time so I used a millis timer for that.

8.2.3 Pathfinding Issues

I fixed this by pivoting to the Tremaux algorithm since the way it was formatted lended itself to it just returning a value for each direction the robot decides to go through which lends itself well to the state machine well since I could just call it and get the next direction versus the path return of A* search. Also treamux it was easier to integrate the obstacle detecting function as well,since I could just update the neighbor cells and use sonar to detect obstacles while traversing it compared to A* star search which the method I was using knew where the obstacles were beforehand. Overall though, using the Tremaux algorithm was the right call in terms of making the pathfinding work with the rest of the project and being functional on it's own.

8.2.4 Accumulated Traversal Error

This is the only thing I couldn't figure out a good solution to since adjustiting for the small minimal error that happens while going straight or the small minimal amount of error of it being 0.1 cm off while going the distance of a cell. (if I had more time I would like to look into ways to use a Integral controller to somehow counteract these small accumulated errors)

8.2.5 CV Color Problem

To create the color filter I took pictures of each of the 3 cups that I wanted to isolate in various lighting conditions. I calculated the mean, median, minimum and maximum of the RGB values of each cup. This data informed how I thresholded the color values to isolate the colored cups.

8.2.6 CV Object Noise

To reduce noise I tried closing the image by eroding and dilating the image. This method was good for small pixels in the frame but was not good for larger error objects. I decided to fix this issue by using connected component sequential labeling. This way I could identify the objects in the frame and sort them out. I filtered out the objects by length-width ratio and size. This reduced the objects down to only the cups. I then have the algorithm select the largest of the cups in the frame (see figure 8.2.6.1).

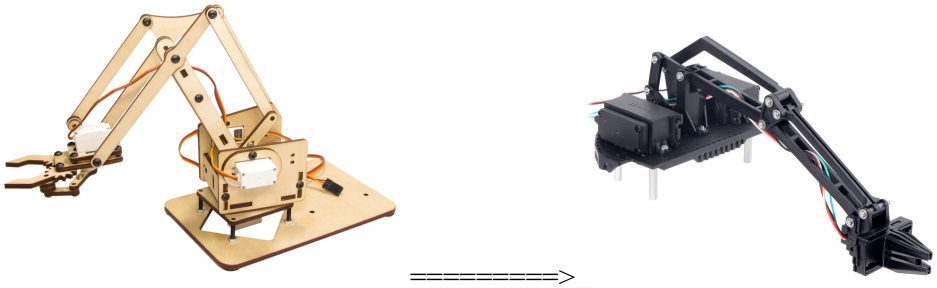
Figure 8.2.6.1: CV

(Original) (Color filter) (Size-Shape Filter)

8.2.7 First Iteration Arm Issue

I fixed this issue by constructing a new arm where we would only use 2 servos.

Figure 8.2.8.1: First Arm and Final Arm

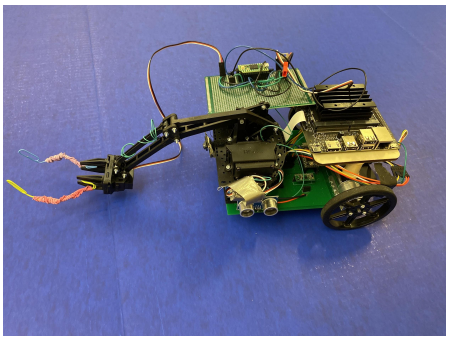


==========>

8.2.8 Chassis Issue

I solved these issues with different types of mounting techniques. I mounted the arm with velcro so that we could take it off and make repairs. I also built a platform so that the Jetson nano can be easily accessed and we wouldn't have to use putty. I attached the 2nd teensy perf board to the jetson nano and the arm. I also velcroed the battery pack underneath the PCB.

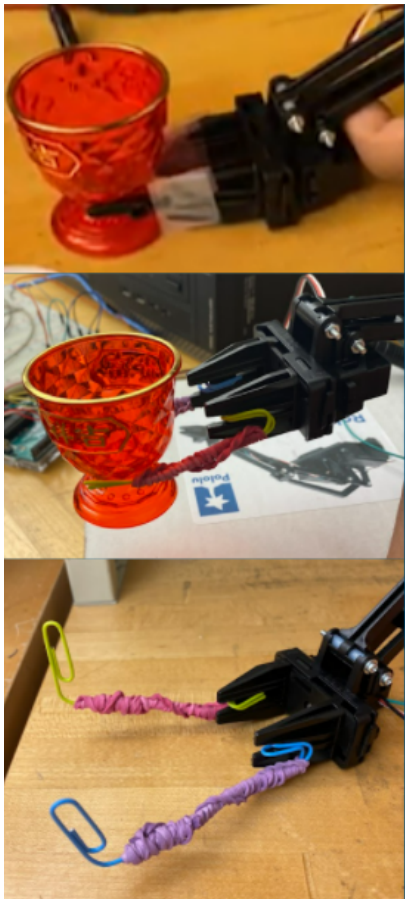
Figure 8.2.8.1: Full Robot Chassis



8.2.9 New Arm Gripper Issue

I solved this issue by creating various versions of an arm extension, and with a large amount of testing, I finally constructed an efficient arm gripper that can be formed to meet whatever shape we wanted to pick up. I also wrapped the extension in rubber in order to increase grip.

Figure 8.2.8.1: Gripper Iterations



9. Test Plan

9.1 Test Design

9.1.1 Test 1: Turning

Dylan was responsible for this task

Objective: The purpose of this test is to see if the IMU module can be used to accurately turn the chassis of the robot 90 degrees in an accurate manner.

Setup: The code for accessing the IMU data must be created and the motors must be properly set to turn until reaching the desired 90 degree angle.

Procedure: Place the robot on the ground and run the code to see how far it turns.

Expected Results: Optimally the robot turns around 90 degrees with an error range of 2.3 degrees.

9.1.2 Test 2: CV Code on MCU

Grant was responsible for this task

Objective: Run the CV code that was developed on the Google Colaboratory on the Jetson Nano.

Setup: Port the code files onto the Jetson Nano and use them to perform CV algorithms on each frame of the camera stream.

Procedure: Run the camera, place a green cup in view of the camera, and print the bounding box and centroid values of the cup onto the screen.

Expected Results: The camera should be able to detect the cup and display centroid values consistently without any error.

9.1.3 Test 3: Pathfinding Traversal

Dylan and Grant were responsible for this task

Objective: In order to traverse its surroundings properly the robot must sense its surroundings and place any obstacles that it finds onto the mapping array. It must also avoid all obstacles and follow the Trémaux search algorithm which should land the robot at its final destination.

Setup: By this point the turning and movement of the base motors pid controllers should be fine tuned and the sonic sensors should be properly calibrated.

Procedure: Set up various test obstacles in the 5 foot by 5 foot environment and run the code on the robot to see what happens.

Expected Results: The Trémaux algorithm should work and the surrounding objects should be detected as obstacles and the robot will always find a path to the goal cell.

9.1.4 Test 4: Sonar / CV Micro Adjust

Dylan and Grant were responsible for this task

Objective: The robot needs to be in a particular position and direction so that the grab command on the arm can properly pick up the cup. We will use the CV and sonar data to accomplish this.

Setup: PID controllers will be needed for the centroid data and sonic sensor data in order to properly control movement. The communication between the Jetson Nano and Teensy should be set up allowing the Teensy to utilize the centroid data and the sonic sensor should be calibrated.

Procedure: Attach the camera onto the base of the robot arm so that it is in the front and center of the device. Place a cup on a platform in front of the robot. Run the files on the Jetson Nano and Teensy.

Expected Results: Using the CV data and the sonic sensor data the robot should be at the proper proximity and angle for the robot arm to pick up the cup.

9.1.5 Test 5: Grab Signal to Other Teensy

Sunil was responsible for this task

Objective: To make sure that the main Teensy can communicate with the arm Teensy to perform the grab commands.

Setup: Connect the input and output pins of the Teensys so that they can communicate with high-low signals. Connect the arm teensy to the robot arm.

Procedure: Run the code on both Teensys and make sure that the grab commands run when they are supposed to.

Expected Results: The arm should do nothing for a delay time then perform the grab command.

9.1.6 Test 6: Micro Adjust to Grab

Dylan and Grant were responsible for this task

Objective: To have the robot identify a cup that is placed in front of its camera and have it move into position and pick it up.

Setup: Attach all parts of the robot together and make sure that the other subsystems work.

Procedure: Place a cup on a platform in front of the robot and run the experiment.

Expected Results: The robot should align and turn itself properly and then perform the grab command to pick up the cup.

9.1.7 Test 7: Final

We were all responsible for this task

Objective: At this point the robot should be mostly working. The robot should traverse through the environment, reach the goal cell and pick up the cup.

Setup: Calibrate all of the subsystems and make sure that they work. Make sure the camera is aligned properly.

Procedure: Run the experiment with various obstacles and see which of them work.

Expected Results: The robot should be able to perform all functions properly.

9.2 Bug Tracking

In terms of Bug tracking when we tested subsystems we just kept track of which thing was acting up at any one time all in my head. For example when testing the traversal functions we kept in mind of what part could cause which problem such as if the robot doesn't see an obstacle that was the sonar or if the robot wasn't going straight that was the pid, If the micro adjust wasn't going well there needed to be

adjustments in the Micro adjust states of the FSM etc. Me (Richard “Dylan” McGee and Grant Beatty did most of the testing)

9.3 Quality Control

To ensure that we would encounter a minimal amount of issues with our project, we made quality control procedures a priority. We would all meet in the same lab room and keep our workstations organized and clean. We made sure to obtain multiple parts in case anything breaks. We tested our circuits with a multimeter so we could be sure our connections were stable.

9.4 Identification of Critical Components

9.4.1 Pi Camera V2

The Raspberry Pi camera was a critical component that required attention because whenever the project is assembled it must be placed properly at the front of the robot so that it reads the proper centroid data. If it is not placed properly then the target centroid value in the main Teensy must be recalibrated.

9.4.2 Jetson Nano

The Jetson Nano is a critical component because it is the most expensive part and so replacing it would be costly.

9.4.3 Sonic Sensors

The sonic sensors were critical components because the soldering joints on them were fragile and they would break easily.

9.4.4 SN754410 Motor Driver

Putting in the motor driver in correctly was vital because without it the motors would not function at all there was issues when putting it together in that the pins of it would become bent which would then harm the connection to the motors so first preliminary motor testing meant double checking the motor driver was properly on

9.5 Items Not Tested by the Experiments

9.5.1 l7805cv Voltage Regulator

The voltage regulator was not tested by itself because it was assumed that it would work.

9.5.2 Bluetooth Module

The bluetooth module would have allowed the user to control the kitchen helper robot remotely. The bluetooth module was already attached to the PCB. However, it was never tested or implemented because of time constraints.

10. Test Report

10.1 Test 1: Turning

Iteration 1:

1. The result of the test was overturning when going right and underturning when going left.
2. The chassis was significantly off from turning 90 degrees in both directions.
3. We had expected the IMU to be sufficient by itself for turning. We had not expected that the drift of the IMU module would have a significant effect.
4. For the next test we recalibrated the target turning angle based on this information.

Iteration 2:

1. The result of the test was that it showed that the robot was able to turn properly.
2. The turning was within the 2.3 degree margin of error which was expected.
3. The tweaking of the values worked to improve the accuracy of the turning in an efficient manner.
4. No further tests were performed because the results were sufficient.

10.2 Test 2: CV Code on MCU

Iteration 1:

1. The result of this test was that it showed that CV code on the Jetson Nano is capable of recognizing red and green cups.
2. The actual results were less promising than what was expected.
3. The cups could be recognized but only from specific lighting conditions and angles. Rotating them slightly would cause them to be not recognized by the camera.
4. For the next test we removed the Hu Moments requirement from the shape filter and added more lighting to the viewing environment.

Iteration 2:

1. The results of this test showed more consistent object tracking.
2. The results met expectations.
3. The cups were tracked consistently with very little error values.
4. No further tests were performed because the results were sufficient.

10.3 Test 3: Pathfinding Traversal

Iteration 1:

1. For the first one the pid tuning was off and therefore the robot wasn’t going straight

2. It was able to do the traversal right in terms of knowing which cell to go to but there was issues in terms of the thing going straight and it was way off center in terms of being at the goal cell
3. Also this was only using pre coded obstacles into the pathfinding algorithm but it avoided them correctly
4. Need to do more extensive tuning of the pid controller

Iteration 2:

1. Robot was traversing much more accurately
2. This is more in line with the expected results of being able to traverse more accurately albeit without any sonar detecting obstacles in it
3. Extensive tuning was done to the pid controller for the concurrent velocity pid controller for both motors so both of the motors were going straight and the robot got to the end cell in a much more accurate way
4. Need to check the sonar obstacle detection was going well

Iteration 3:

1. Sometimes the sonar would not see the obstacles and therefore would not see them as obstacles
2. Not expected since it was not detecting obstacles correctly
3. Need to make sonars more accurate in detecting obstacles
4. For later tests this was changed to make it so the sonar did multiple readings along with resoldering the left and right sonars back on to make the readings more accurate. Also another thing that was the issue was sometimes the robot would go in a suboptimal direction such as going left instead of right (to where the goal was) for the next test I changed the code to go off of a (up , right , left , down) priority where it takes the neighbors in consideration in that order to make the robot traverse a more optimal path

Iteration 4:

1. After all of those changes were done as you can see in the traversal demo videos the robot was able to traverse the maze’s in a very accurate manner

10.4 Test 4: Sonar and CV MicroAdjust

Iteration 1:

1. For the first time testing the cv distance left and right function was not going to plan in that it was doing the opposite of what was intended,
2. The reason this was because the camera was inverted so the robot needed to turn in the opposite direction base robot needed to turn
3. Also the robot while doing the sonar would sometimes see crazy high values for a second which would mess up the up and down adjustment to the block where the cup was

4. For the next test I changed the code to make it so the sonar adjust functions take the average of multiple readings and if the reading was too high it would not use that reading, also since the camera was inverted, I just inverted the directions the robot would move in those cases.

Iteration 2:

1. Testing it now there was still issues in that the robot still wouldn't always get straight in front of the cup so 1 things were changed , but it would go up and down / right to left adjustments correctly
2. Needs to be able to be more centered in on the cup
3. Before the sonar would run for a number of times and then the cv adjust would run for a number of times but I changed it after this test for them to run interchangeably so that way there's no residual error if one ran before the other because sometimes the up and down adjustment would leave the cup off center

Iteration 3:

1. After all of those changes the micro adjust now worked accurately in terms of making sure it lined up right to the cup

10.5 Test 5: Grab Signal to Other Teensy

Iteration 1:

1. The first grab function worked well since it was just testing the custom bit communication between teensyies but there was one small problem
2. The arm would keep doing the grabbing motion over and over again unlike what was wanted
3. Need to make sure it doesn't keep doing the grabbing over and over again
4. Added in the code to make sure it only sends high once from the main teensy code and then waits until it is sent High back

Iteration 2:

1. Grab signal worked correctly

10.6 Test 6: MicroAdjust to Grab

Iteration 1:

1. This worked well since the align-tilt and the grab functions all worked separately all that was needed was to just make sure the transition from the micro adjust states of the sonar and cv to properly go into the communication with the other teensy state to grab the cup and it worked a majority of the time.

10.7 Test 7: Final

Iteration 1:

1. The tests would mostly always find a path to the final goal cell but sometimes there would be issues in picking up the cup at the end
2. Needs to always pick up the cup the majority of the time
3. Corrective measure was to add a extra state in the FSM to be find cup in where the robot would go to the right by 20 encoder count which is small to slightly adjust the robot until it sees the cup

Iteration 2:

1. The small edge cases in which the robot didn't detect the cup was minimized through the new find cup state in the FSM also after more tuning of the pid controller the robot would accurately get to the goal cell straighter than ever before.

11. Conclusion and Future Work

11.1 Conclusion

The resulting kitchen helper robot was a success and a viable proof of concept. There were a few features that were not added to the design due to time constraints, but overall most of the intended functions were implemented. The technical objectives of the design required a combination of several elements of electrical and engineering. We utilized state machines, robotics, computer vision, PID controllers, and several communication protocols which allowed for the combination of subsystems. Combining these concepts allowed for the creation of a significantly complex product.

Despite getting most of the primary functions to work in an organized manner we were unable to get the robot to perform several functions. The robot initially was supposed to check each cell for the target object but we had to simplify our design so it only checks for objects adjacent to the goal cell. We also wanted to have the robot take the target object and move it to a bin near the start cell. This required extensive use of movement and pathing and that would have added significant complexity so this idea was never fully realized. Bluetooth control of the robot was also not implemented because of this reason.

Ultimately the robot is able to perform a variety of tasks involving the integration of various subsystems. Using PID controllers, the chassis is able to go straight and turn very accurately with DC motors, which is a tedious task in of itself. It successfully navigates through cells on a table top sized space using sonic sensors and mapping to detect and keep track of obstacles. Upon reaching the destination it uses computer vision and a robotic arm to identify and grab a target object.

The tasks that Grant was responsible for were computer vision, MCU integration, and testing design. Sunil was responsible for the robot arm, chassis design, and project management. Dylan was responsible for the whole base robot code, PCB design, finite state machine ,subsystem communication, and control systems.

Grant:

Throughout the project I learned a significant amount about solving physical problems using elements of engineering. I further developed my programming skills in several ways. I had never coded in python before or used the OpenCV library, both of which I used for this project. Most of my experience with CV was with basic algorithms in MATLAB. Initially there was a bit of a learning curve but Python is an easy language to figure out so I mastered it relatively quickly. I learned to never use for-loops in Python because they are very slow and that built-in CV functions are extremely fast. I got the opportunity to use the tools from my EE 146 CV class to solve technical problems which was a very engaging process. My knowledge of embedded systems and MCUs were improved by working with the Jetson Nano and the Teensys. Figuring out how to perform serial communication on these devices was an interesting task. Working with a team was a useful experience.

Communicating deadlines, planning, developing, and meeting up for testing are professional skills that I will take away from this project. Figuring out how to research a problem also aided me significantly in the design process especially with learning Python. The importance of contingency plans was an aspect of designing that I initially underestimated at the beginning of the project. I will make sure to focus on this area significantly more in future projects. Overall the project was a success because of extensive cooperation and communication between myself and my group members.

Sunil:

Over the course of this design project I learned so many skills that would aid me in my engineering pursuits for years to come. I programmed the arm in C++ and therefore strengthened and retained my programming skills. I also created a test board and created a circuit that had joysticks and an arduino attached to it so I could conduct very thorough testing of the servos and the arm as a whole. I learned how to problem solve in a very efficient way while dealing with the arm extension process and chassis modification/constructing.

I was in charge of overall project management for this design project and it made me realize that I would like to possibly pursue a career in a field related to professional development. I organized our google docs in such a way that we had discussion notes from every lab section we attended so we could go over what needed to be done for the future. I created two comprehensive Gantt charts that I learned how to make on google spreadsheets, one for Fall quarter and one for Winter quarter.

Throughout this project I learned that the most ethical responsibility an engineer can have while making a product is to make sure that this product would make someone's life better. If that mentality isn't at the heart of our project then we would be doing our customers a disservice. I believe our Kitchen Helper Robot was designed with these ideas in mind and I am thankful for all of the knowledge I have gained from helping this project come to fruition.

Dylan:

I learned a lot doing this project since I did all of the base robot functions so all the traversal code, the pathfinding, the pid controllers, the FSM and the pcb design and construction of the base robot, I also handled all of the circuitry issues of the project through first testing with a multimeter and then soldering any bad connections together.

For specific things I learned I learned how to properly implement a velocity pid controller which is something I didn't know how to do before. I learned a lot about I also learned a lot more about PCB design then I did before in order to make schematics better and also easier to read.Also I learned more on how best to make a PCB board in I also got more practice with implementing FSM into code because I made the whole program run off of a FSM. Also I learned more about how to get subsystems to communicate through custom digital connections using custom spinlock code and also communicating through a Serial Connection between a Jetson Nano and a Teensy.

Overall I'm happy with how the project turned out since it could traverse a 5x5 maze (while going straight with pid) and go and pick up a cup at the goal cell

11.2 Future Work

There is plenty of future work that could be done on the system. Obviously the aspects of the project that went unfinished fit into this category. The improvements that could be made on the CV system would be to have the camera be able to identify various kitchenware objects and not just cups. Computer vision using neural networks could solve this problem. There could also be modification of the arm grabbing system as well. The arm would have to be able to come up with a variety of different ways to pick up a plethora of objects such as plates, candles, and silverware. Additional sensors or another camera would be needed to calculate the optimal way to grab each object. More efficient mapping and more complex movement protocols would also be very helpful for faster traversal.

On our end, as the developers of this product, we need to make sure that the materials and parts of our robot are ethically sourced. This means not cutting corners on buying products that are cheaper, but could very well be made unethically by a company that doesn't pay its workers properly. We should also be conscious about how our robot will affect the environment. We should keep in mind what types of materials are sustainable and also what kind of impact our electronic waste could have on the environment.

Our Kitchen Helper Robot can help in larger kitchens like in restaurants and catering services that need multiple cups and utensils set in a specific order everytime. If restaurants find a need for our robot then hotels would soon follow and that would help people out all over the world. Then after this, we could develop a smaller and user-friendly version that could uniquely fit into the everyday life of a working class family and ease their burdens instead of causing them.

11.3 Acknowledgement

- Dr Roman Chomko
- Merrick Campbell

12. References

Arduino Libraries and Add-ons:

Teensyduino Add-on	https://www.pjrc.com/teensy/td_download.html

Jetson Nano Python 3.0 Packages:

NVIDIA Developer Kit	https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-2gb-devkit#prepare
Numpy	https://github.com/numpy/numpy
Python Pi Camera	https://github.com/JetsonHacksNano/CSI-Camera/blob/master/simple_camera.py
Serial	https://github.com/pyserial/pyserial/releases

Datasheets Used:

HC-SR05 Sonic Sensor	https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf
Teensy 3.2	https://www.pjrc.com/store/teensy32.html
SN754410 Motor Driver	https://www.ti.com/lit/ds/symlink/sn754410.pdf
L78 Voltage Regulator	https://www.mouser.com/datasheet/2/389/cd00000444-1795274.pdf

13. Appendices

13.1 Appendix A: Parts List

Parts	Quantity	Link
Jetson Nano	1	https://amzn.to/3Fl8jiC
sn754410 motor driver	1	https://www.pololu.com/product/24
sonar sensor(HCsr04)	4	https://www.sparkfun.com/products/15569
prototype breadboard	3	https://jlcpcb.com/
wires package	6	https://www.amazon.com/Jumper-Female-Wire-Arduino-raspberry/dp/B01MT530B8
LSM6DS33 imu	1	https://www.pololu.com/product/2736
l7805cv voltage regulator	1	https://www.mouser.com/ProductDetail/STMicroelectronics/L7805CV?qs=9NrABl3fj%2FqplZAHiYUxWg%3D%3D
servos to make robot arm	1	shorturl.at/dAGM5
32 gb Micro SD card	1	https://www.amazon.com/32gb-Micro-Sd-Card/s?k=32gb+Micro+Sd+Card
INIU power bank	1	shorturl.at/cqCEH
Raspberry Pi Camera	1	https://www.raspberrypi.com/products/camera-module-v2/
motor Brackets Pair for 99:1 25D dc pololu motors	1	https://www.pololu.com/product/2676
aluminum mounting hub pair	1	https://www.pololu.com/product/1081
Wheels Pair	2	https://www.pololu.com/product/1420
DIY Cardboard Robot Arm	1	shorturl.at/hjyC1
KeyeStudio 4 dof arm	1	shorturl.at/dvIN0
pololu arm kit	1	https://www.pololu.com/product/3550
bigger dc motors 25D 99:1	2	https://www.pololu.com/product/3207

13.2 Appendix B: Equipment List

Glue_gun
Multimeter
Wire Solder
Soldering Iron Tip Cleaner
Solder Sucker Pump
Stranded 22Ga Wire (Assorted Colors)

13.3 Appendix C: Software

13.3.1 Software list

PCB Modeling: Eagle
Software Development: Arduino IDE, Google Colaboratory, Python IDE

13.3.2 Program Files

Grant Beatty Code:

https://github.com/gbeat002/Senior-Design
CV -ReduceSizeAndPad.py -simplified.py -IsolateObj.py -six_chars.py -serial_test.py -test2.py

Richard “Dylan” McGee Code:

https://github.com/anorakalot/Senior_Design-/tree/main/senior_design_proto_5
Base Robot Teensy

(in Senior_Design_Proto_5 Folder)

senior_design_proto_5.ino
global_values.h
gyro_func.h
misc_functions.h
motor_func.h
sonar_func.h
Treamux_func.h
(tremaux folder)

PCB Design Custom Libraries

Sonar_Sensor.lbr
Test_row.lbr
Motor_bracket_25D.lbr
Motor_encoder_25D.lbr
Arm_Connection.lbr
Custom_bit_comm.lbr

https://github.com/anorakalot/PCB_DESIGN
IMU.lbr
Battery_input.lbr
Voltage_regulator.lbr

Sunil Alexander Code:

https://github.com/Sunil98/Senior-Design

Teensy Arm

-grant&sunil's_servo_code.ino
-meArm_Servotest.ino

Link to Google Drive w/ all other project videos:

https://drive.google.com/drive/folders/1-0Ty6ppuRTMtv3moIdNHyPeYwxIwMtAl?usp=sharing

https://drive.google.com/drive/folders/1HYhf_CaxBAEnVH4NM4gNPAFn-sjZ5ZCw?usp=sharing