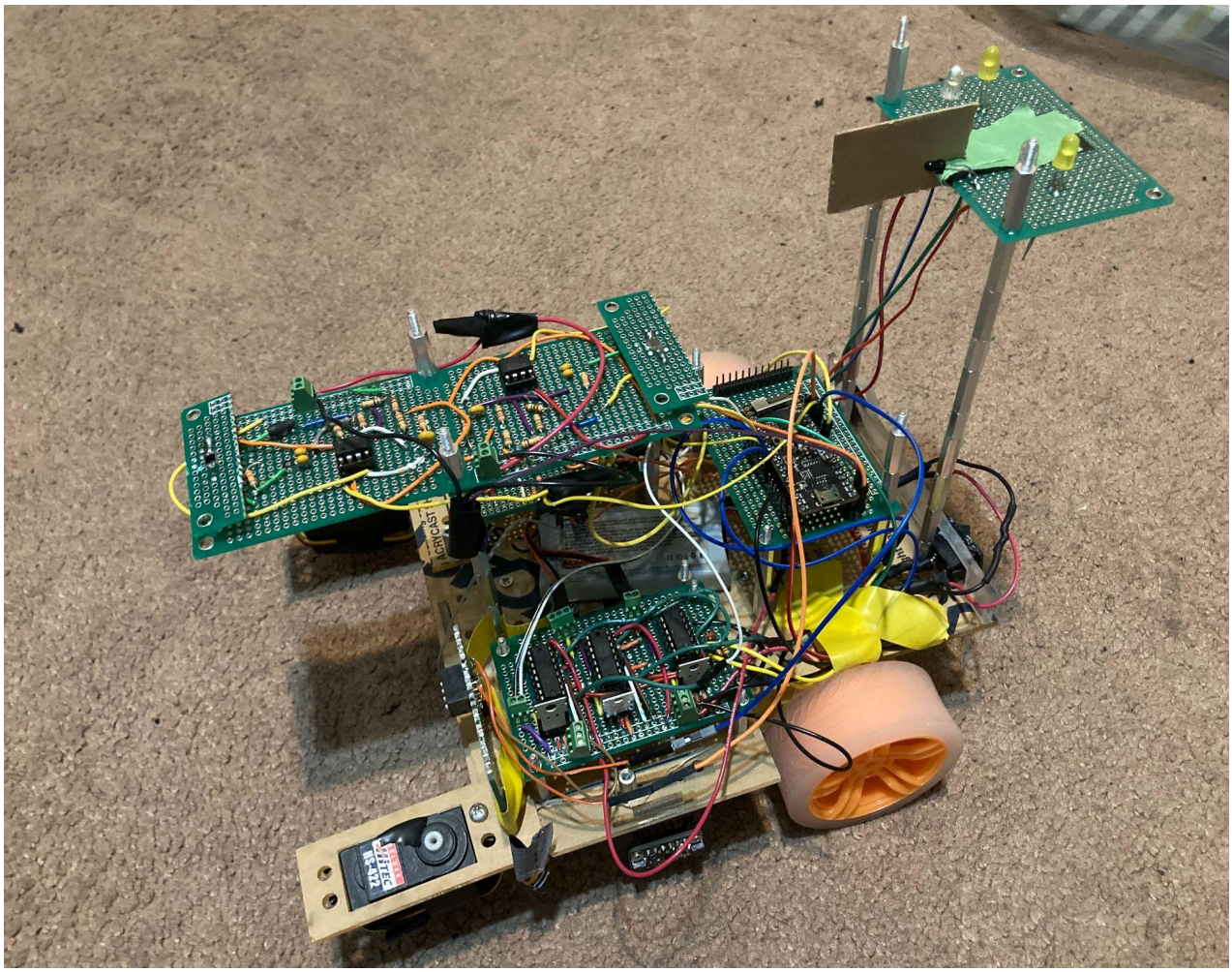
MEAM 510 Design of Mechatronic Systems

Lab 5 GTA 2021C Report

Group 26

Chaitanya Gaikwad, Divyanshu Sahu, Vanshil Shah

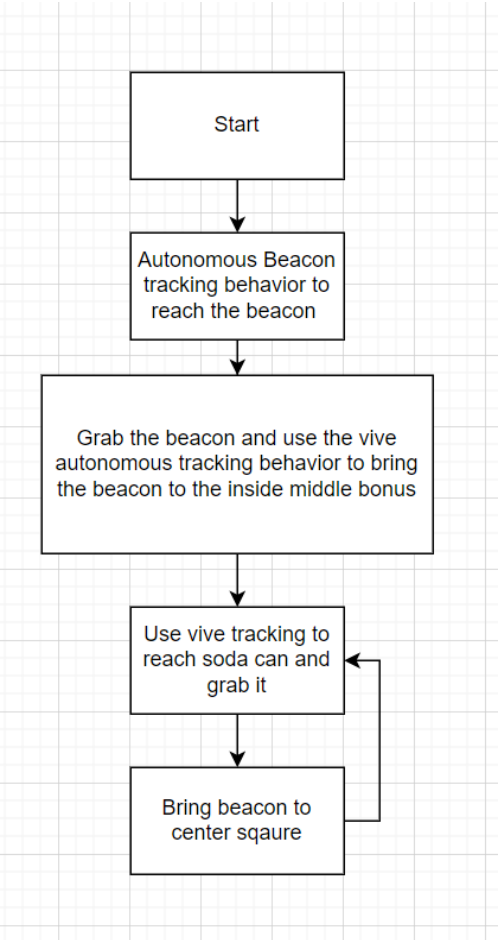


Functionality

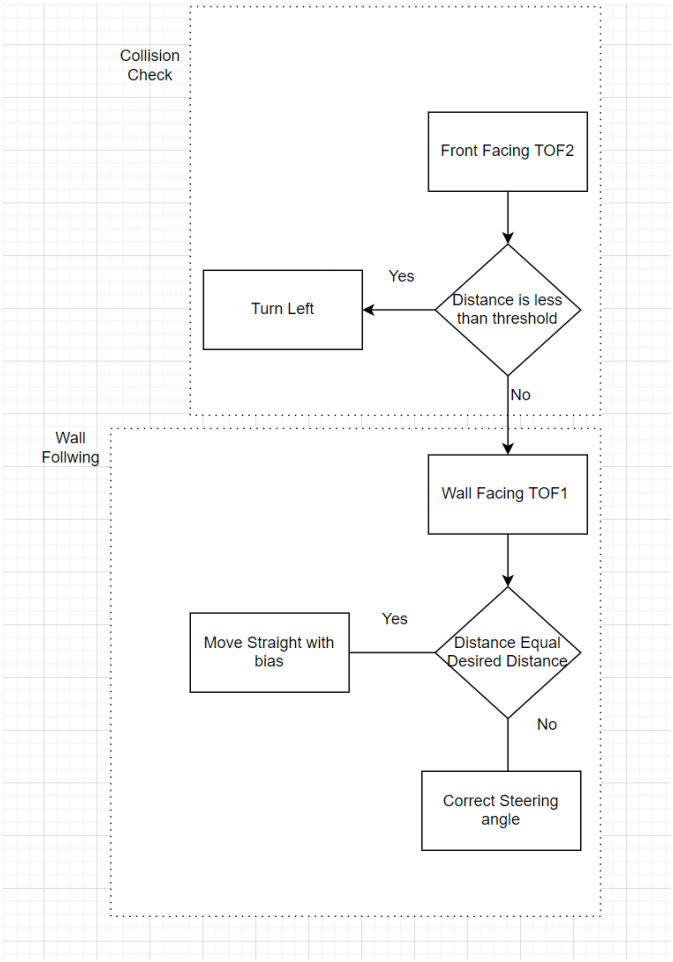
For the final Grand Theft Autonomous 2021c competition, our strategy was to first keep our cans and beacon in the double points square area, which we intended to do manually in order to do it quickly to save time. Then our strategy was to look for the beacon on the other side and go into the autonomous mode to ‘steal’ the beacon and bring it back to our side. We wanted to go for the beacon as it contributed the most points to the opponent's side.

Followed by this we wanted to use the locations of the Cans broadcasted via UDP to everyone and then make our robot go the cans autonomously onto the other side then steal the cans and bring them back to our side. We intended to do this by setting Can ID from the web interface.

For backup our strategy was just to focus on Cans and Beacons from our side and keep them in the double points square area only and just try to create chaos on the other side autonomously. We wanted to use an active gripper using two high powered servo motors and then have two motors in differential drive mode and along with a caster wheel for robot motion. Considering the complexity of the sensors, algorithms and electronic integration we first decided to work with Time of Flight(TOF) sensors for wall following, then beacon tracking and finally with vive circuit and UDP functionality.



1. Wall Following



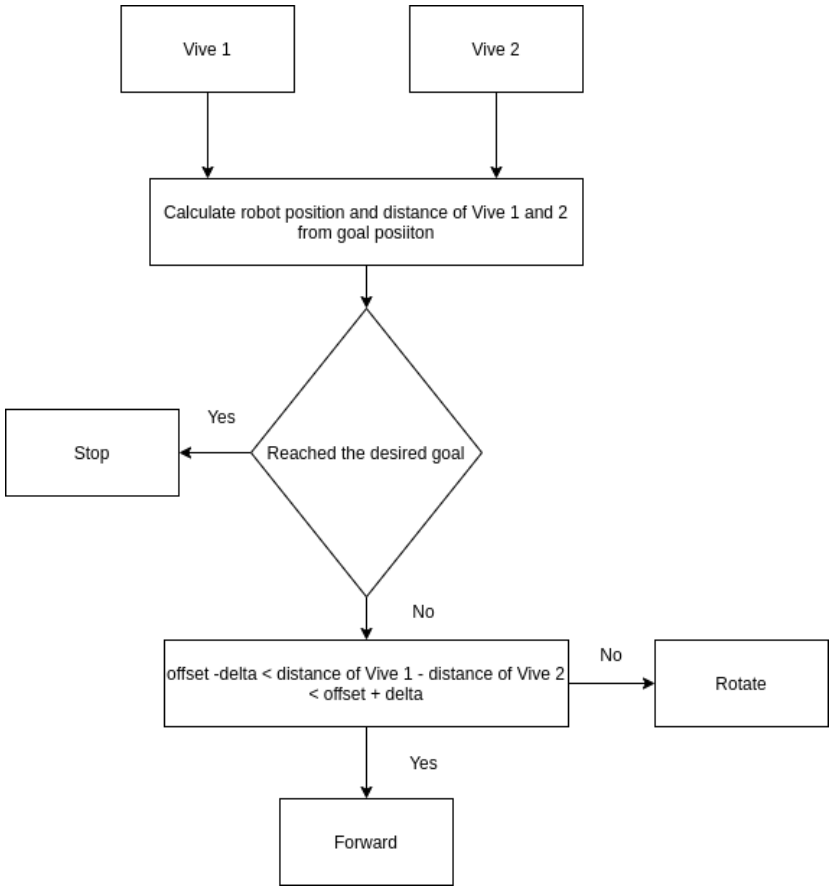
We designed a system that incorporated two distance sensors. One for side wall sensing and one for front wall sensing. The basic logic was to maintain a constant distance from the side walls which was maintained by controlling the PWM value for the two motors driven in differential mode through a bias.

The algorithm for following the wall was to maintain a constant distance between wall and the robot using the side facing distance sensor. If the distance increased from the specified threshold value then we commanded the robot to steer towards the wall and if the distance decreased from the threshold value we commanded the robot to move away from the wall. The front-facing distance sensor helped in deciding when to turn right when approaching a wall in front.

We were using a TOF sensor for front wall sensing and an ultrasonic sensor for side wall sensings. The sensors were mounted on the robot so as to avoid sensing the ground and yet be able to measure the side wall considering the height of the side walls. A threshold was used to separate readings from the grounds.

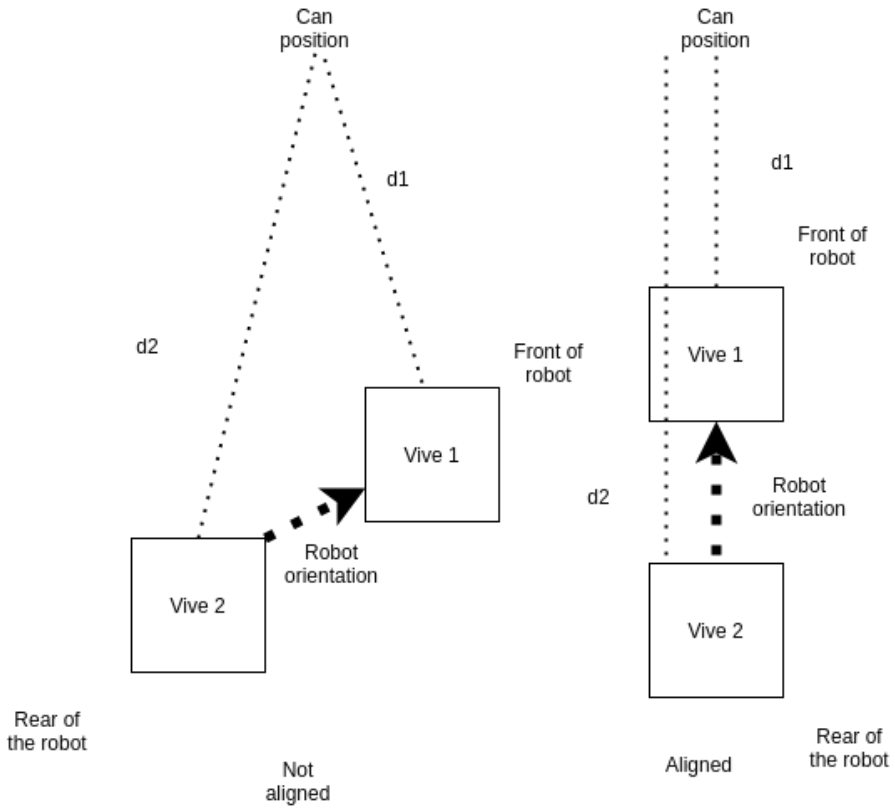
We initially planned to use two TOF sensors, however as they shared the same data lines with the ESP32 MCU control was difficult and hence we shifted to one TOF and one ultrasonic sensor combination. Further details are mentioned in the Electrical Design section of this report.

2. Can following

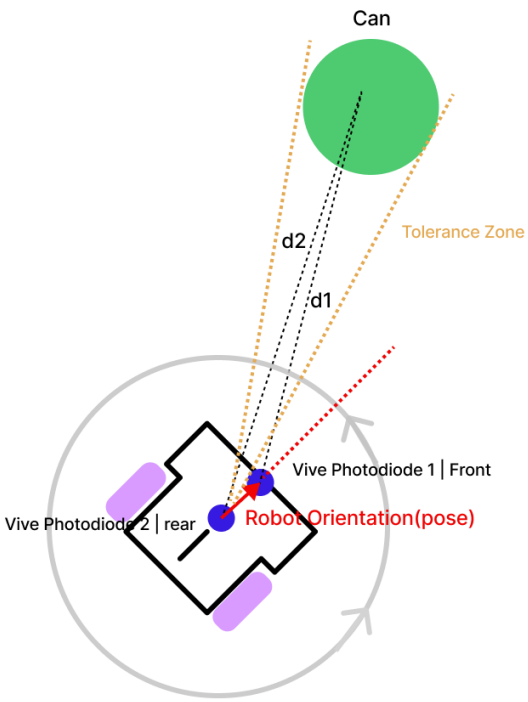


We placed the two photodiodes in a straight line on top of the robot in order to avoid any blocking of the vive sweep in order to not lose any connection with the vive and get noisy values. The implementation of the circuit was an initial step for Can tracking that was easy to implement. However, we faced an issue of noise from a photodiode when using two different circuits even when other photodiodes gave correct readings.

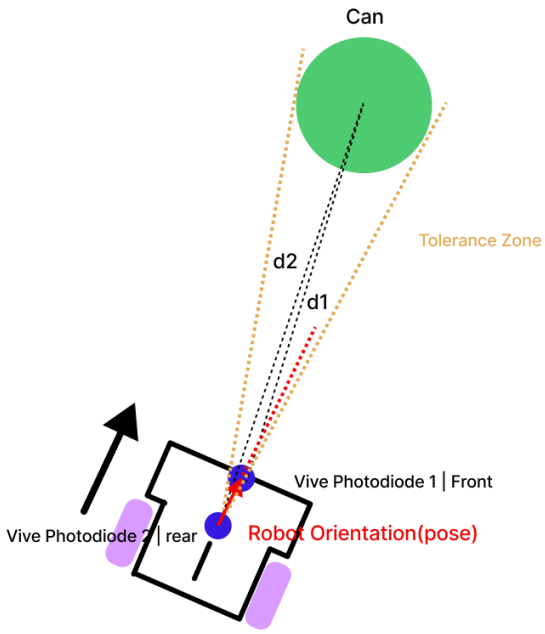
Once we had the X,Y location of the front photodiode and rear photodiode we got the heading and pose of the robot. Our initial approach towards achieving the can position was by calculating the angle difference between the heading of the robot and that of the Can to find the final orientation of the robot. This however created a lot of challenges as we observed discontinuities in the orientation calculation of the robot. This was because we were using the tan inverse to calculate the final heading which had discontinuities. This caused the robot to oscillate at the discontinuity indefinitely and get stuck.



To resolve this issue, instead of looking at the angle and orienting the robot, we switched to seeing the distance itself, which was scalar and helped us to deterministically follow the can everytime. Given the Can positions, we calculated the distance of both the vive coordinates from the Can we wanted to pursue. From calibration, we knew that when the robot was aligned with the Can, the difference between the vives and the Can positions would be constant. Using this concept, we made the robot rotate about its own axis until the two vivie's were aligned with the can. As we knew the constant difference value from calibration when the robot aligned with the Can, if the distance was in the certain range of the fixed difference value, we assumed that the robot was aligned with the Can.



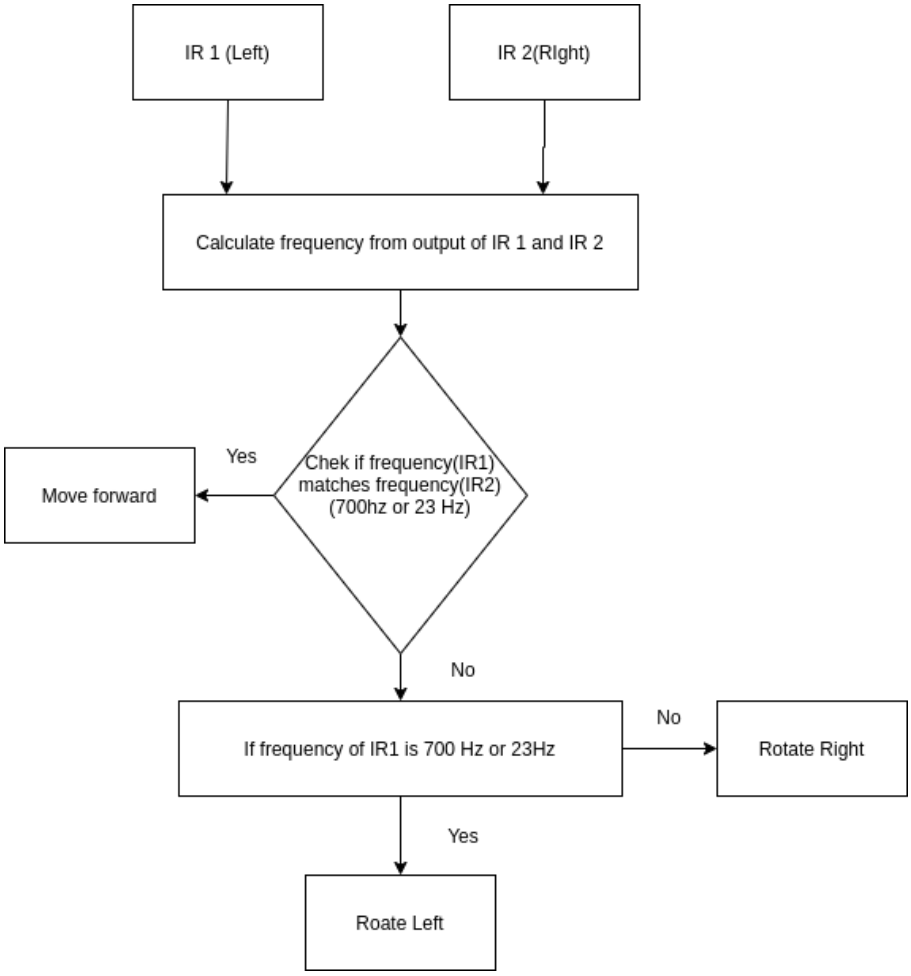
Not Aligned



Aligned

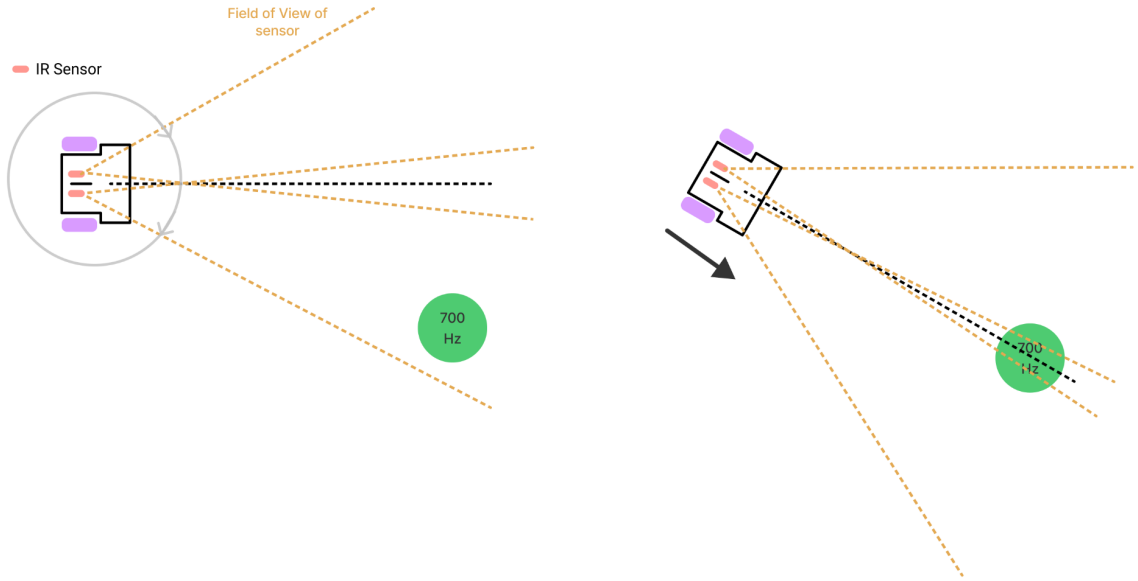
It is also worth noting that there are two possibilities where the robot will be aligned with the Can as per the distance concept explained above. These positions will be 180 degrees opposite to each other. We knew that when the robot was aligned with the Can, the first vive was closer to Can than the second one. Hence, we simply checked the distance from the vive to the Can and ensured that if the robot was aligned with the robot, the distance of the Can from the first vive was to be smaller than the second vive. This helped us avoid ambiguity and made our algorithm deterministically follow the Can everytime.

3. Beacon tracking



We had mounted two IR photosensors separated by an opaque surface to deterministically sense the direction of rotation for the robot given the frequency of the beacon. This design allowed us to eliminate guessing the direction in which the robot should rotate in case one of the sensors lost its lock with the beacon. For beacon tracking, we give an input frequency of either 23Hz/700Hz.

The robot then rotates at its position randomly at first to find the emitted frequency. Once a single sensor has found the beacon, the algorithm deterministically knows which side the beacon is w.r.t the robot. Using this information, we rotate the robot and align the robot until both the sensors have locked on to the beacon. Once both the sensors sense the same frequency it means that the beacon is right in front of the robot and the robot starts going forward.



In case one IR sensor loses signal then the robot corrects itself based on the mounting side of the IR sensor i.e. if IR sensors on the left side sense the signal then the robot rotates left until both sensors sense the signal then starts moving forward. We found this logic to work really well and was robust with us able to sense signals from a distance more than the field itself. However, we found that sensing the distance in presence of the overhead and surrounding lights to be a bit more difficult due to interference.

4. UDP transmission with DIP switch

The UDP transmission logic was built upon the code given in the example section on Canvas. The x and y coordinates of the robot given from vive were transmitted along with the id of the robot. We used ESP in STA mode and the packets were broadcasted through the router on IP address 192.168.1.255. The id of the robot was decided by reading the state of the DIP switch. We also implemented a check in our code to limit the transmission of UDP packets. They were transmitted in a 2-second interval.

5. Extra Credit: Lighting a green LED when in autonomous mode

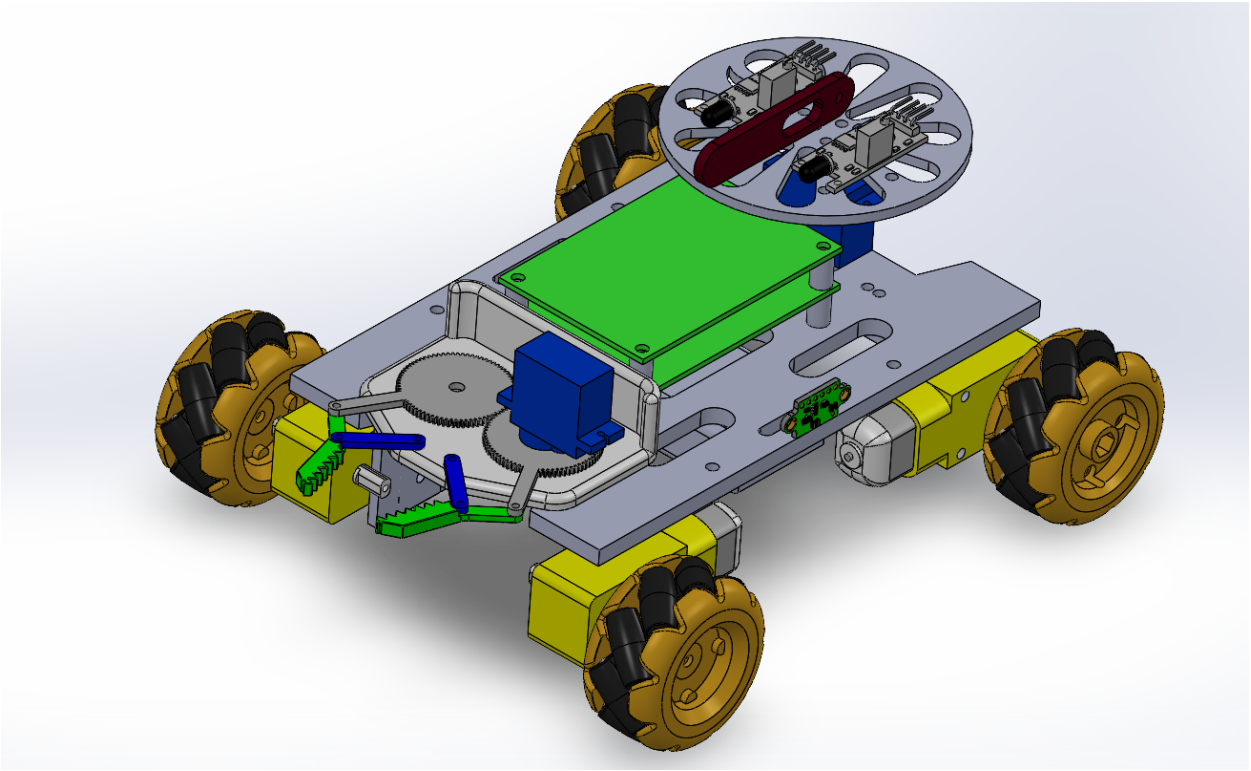
For implementing this functionality in the robot, a flag was added in each callback function of attachHandler. These functions are called whenever a command is sent by the Web page client to ESP 32. Hence when this flag is not enabled and 5 seconds have passed, a green Led is lit. The 5-second interval is checked by using the command ms() which gives us the current timestamp of the program.

Mechanical Design

Our design consisted of two TT 12V DC motors and a caster wheel placed in the front of the vehicle.Our design decisions were also influenced by budget and lead time constraints. Our main criteria were to be bottom heavy to avoid any tilting in case we crashed into any other robot. Second criteria was that we wanted to have a lower center of gravity to be stable. Third consideration was to minimize the weight of our robot to increase maneuverability and decrease power consumption. We wanted to make sure that the robots fit into the spatial constraints of the competition. Weight and dimensions were also criteria to minimize polar moment of inertia.

The TT motors provided enough torque considering our low weighted robots. Initially we had designed for Polulu high torque motors however due to the extremely high torque when compared to weight of our robot, the robot was constantly tilting on accelerating and hence the control was difficult. Also the battery was draining quite fast due to the high current requirement of the motors. Achieving the balance between required PWM duty cycle value to move the motors and the PWM duty cycle value to avoid stalling was a time consuming process and hence we decided to work with the TT motors.

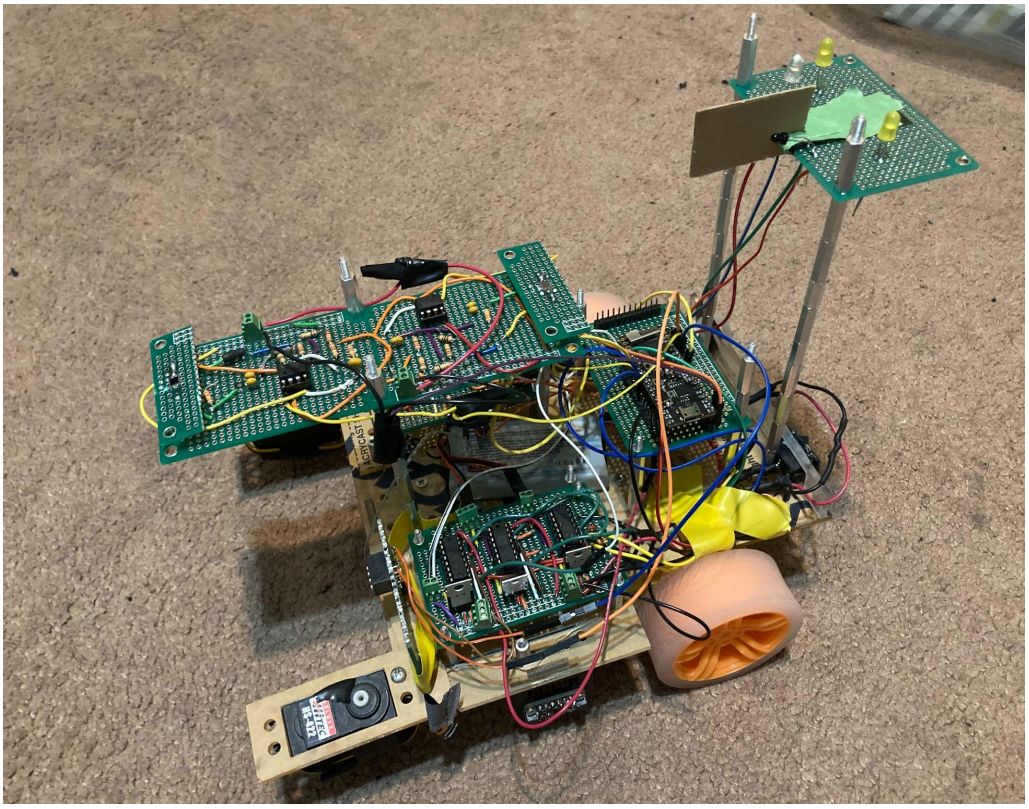
Modularity in design was a major criteria while designing the chassis. This was needed inorder to quickly swap parts and debug if required. We performed almost 5-6 design iterations to get to the final design. Wire routing and wire lengths were also considered during mechanical design to avoid long length wires and wiring and debugging complexity. In the final design we were able to accomplish modularity, lightweight, simplicity and lower center of gravity. The chassis is made of 3mm acrylic along with few sensor mountings made up of 3D printed ABS and PLA. The electronics components are placed on standoffs to isolate any soldering below the performs and minimize the chances of wires touching each other. Placement of perfboards, battery and circuits were also considered before manufacturing the chassis. Provisions were provided to accommodate active and passive arms. Provisions were also provided to protect against collision in case another robot crashes with us.



Initial Design Iteration

We decided to move forward with two motors as opposed to four as they were easy to control. We used rubber tyres to generate enough grip in case other robots started pushing us. We are using differential drive mode to drive the robot where we are controlling the robots by changing the PWM and direction of rotation of the motors on either side. This allows us to keep the most simple control of the robot.

The alignment of sensors was also very carefully chosen and iterated over. The mounting of TOF sensor and ultrasonic sensor was designed in such a way that the sensors are aligned in a uniform and planar manner to reduce spurious readings from objects other than the wall. The IR sensor placement and angle alignment was a critical design choice and was done in a way that gave optimal field of view of the beacon.



Final Design

The photodiodes were required to have a clear view of the vive transmitter at all times in order for the robot to localize itself correctly at all times and for the algorithms to work correctly. For this the photodiodes were placed on top of the vive circuit close to the signal conditioning circuit as instructed in class. Moreover, the photodiodes placement was modular and the circuit was designed so that the photodiodes could be replaced with a new one with ease. This also helped us a lot in debugging if the circuit was wrong or the sensor.

Electrical Design

1. Pin allotment for various sensors and motors

Motors: We used 2 motors and an inverter in our circuit. Hence we needed 2 direction pins from ESP 32 which were connected to an inverter. The GPIO pins chosen for direction were 25 and 26. For controlling the PWM of two motors, the pins selected were 25 and 26.

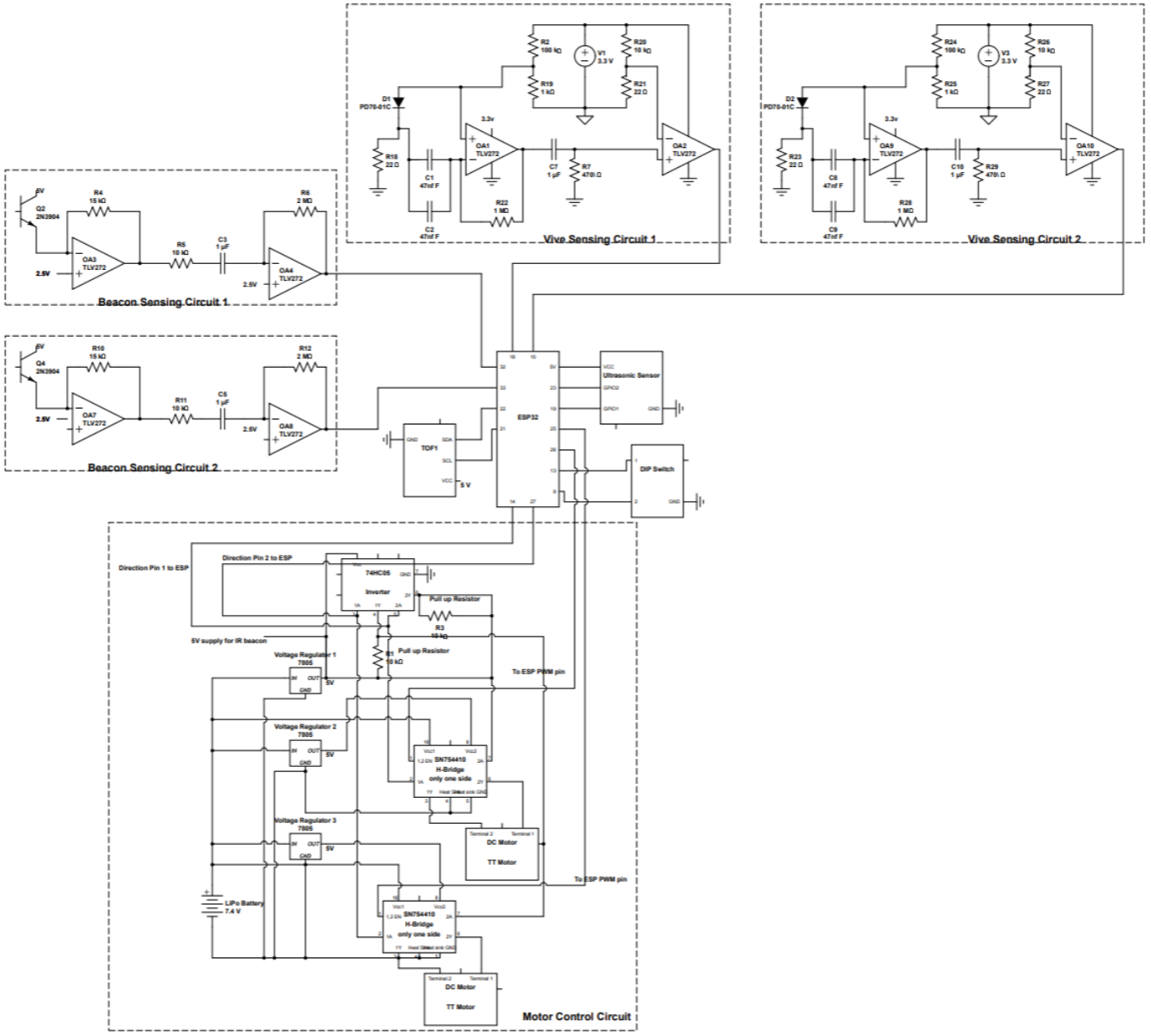
Time of flight sensor : For the time of flight sensor, to enable I2C communication between the ESP 32 and the sensor SCL and SDA pins were chosen as 22 and 21 (default I2C pins).

Ultrasonic sensor: This was the second distance sensor that we used for wall following. The echo and trigger pin of the sensor was attached to GPIO pin 19 and 23 of ESP32.

IR sensor for beacon tracking; We used two IR sensors for tracking the beacon. The output from each sensor after signal conditioning was given to GPIO 32 and 33.

X-Y localisation using HTC Vive: We used two Photodiode sensors for determining both the orientation and position of the robot for better localization. Hence we connected the output of these circuits to GPIO 18 and 10.

DIP switch: We used 2 pins of the DIP switch to create 4 possible id's of the robot. The switch was connected to the pins 13 and 9 on the ESP 32 which both had pullups enabled to avoid floating state error.



2. Things that we observed and learned

1) **Inverter pullup:** To save GPIO pins on our ESP circuit, we used an inverter to control 2 motor directions with only 2 GPIO pins. On initial testing of our circuit, we found that the inverter was only working in one direction and was not able to give the intended voltage in the inverted side when ground was given to the other pin. Upon searching a lot, we found that the inverter pin was open drain output. This meant that we had to add a pullup

resistor to pull up the inverter pin to the intended voltage to give to the motor driver. Hence, we physically added a 10k ohm resistor by using trial and error method for our suitable application.

2) **PWM current issue**: With an intent to save pins for different functionalities, we decided to use a single PWM pin for both our motors as we wanted to give the same speed to both the motors while controlling the robot. During initial tests, everything seemed to work perfectly fine, however, one of the motors would randomly stop working after sometime when the robot was being tested in manual mode. After debugging, we figured out that the PWM current was not equally distributed in the two robots and hence one of our motors suddenly stopped working. To fix this, we decided to use two separate pins for the two motors and our problem was solved.

3) **Improvement of beacon circuit** : To improve the sensitivity of our circuit, we increased the gain of our amplification circuit. Moreover, we also improved the capacitance of our circuit from 100nF to 1 mocroF. This made the circuit sensitive enough to detect the beacon from across the field. The increased capacitance of our circuit made the rise time of the detected frequency smaller which made the sensitivity rise and we were able to sense the beacon from across the field.

4) **TOF initialization problem**: Initializing the TOF sensor without powering it through the usb cable. The unexpected observation was that the TOF sensor was initializing only when the ESP32 MCU was initially powered using a USB cable connected to a computer. On powering the ESP 32 directly from the battery using the 5V pin which allows (5V to 12 V) supply the TOF was not able to initialize consistently. We were not able to solve the first problem and had to connect through USB every single time we had to start the ESP and then switch over to the battery and unplug the USB supply.

Processor architecture and code architecture

We had allocated the pins in such a way that we only required one ESP to execute all the required functionalities for both the competition and the graded evaluation. We didn't want to add any delays by additional communication between two microcontrollers.

Various libraries used in our code:

vive510.h: For calculator of X and Y coordinates using Vive

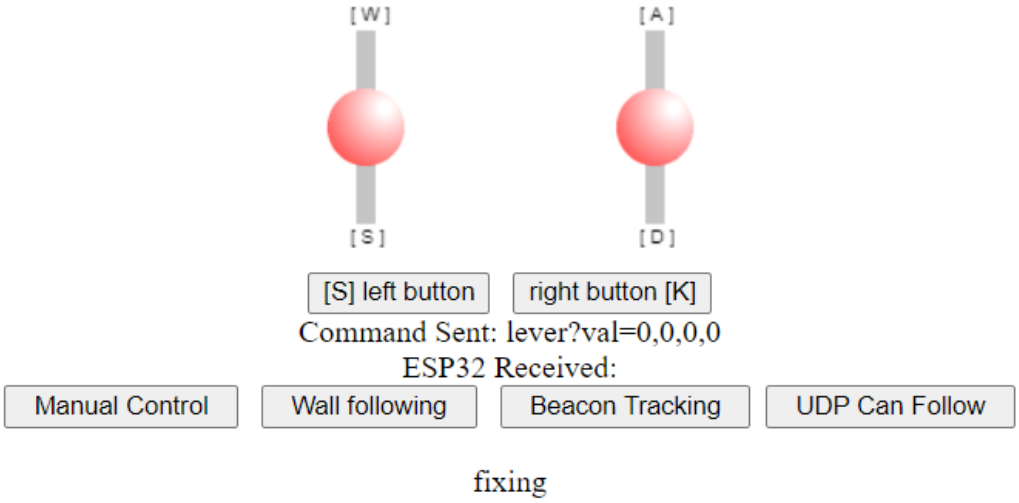
Math.h: For math and angle calculations

Wifi.h: To enable wifi communication between ESP and router

WifiUDP.h: To send and receive UDP packets

Adafruit_VL53L0X.h: For interacting and parsing distance readings from TOF sensor

NewPing.h: For interacting and parsing distance readings from Ultrasonic sensor



Web Interface

Software architecture and approach

In order to make our code modular and include all the above developed functionalities, we decided to keep separated functions for each functionality and the interface to switch between the functionalities smoothly. This approach led to minimum delay in communication and gave modularity to our code allowing fast and efficient changes whenever required.

First, in our main loop we started with the most important information that was needed by each functionality of the robot. This included sensor readings and DIP switch reading for our robot. Moreover, as we had to broadcast our vive location continuously and get the UDP messages as

well, we included this functionality in the main loop itself and used global variables to update these values whenever the required function was called.

Now that we have updated all the necessary information of our robot and sent the required data as well, we then use the different handler functions which are callbacks for when a button is clicked on the web interface to update global variables which decide which functionality will run on the robot. By default, we initially set the functionality to be manual control and then as the loop iterated, we used these handler functions to update a global variable which switched states as per our command. This state-machine type architecture was very useful and modular to add/delete features while developing the final code and also helped reuse several functions again and again. For example, regardless of the logic of different functionalities, the robot control for moving the robot remained the same. Hence, we were able to use the same function to control the robot in different autonomous modes.

We tried to minimize the delay in our code by trying to make it as event-based as possible with application of events and services and removal of blocking elements wherever necessary. We realized from the lag in our manual control that we were printing too many debug statements on our Serial which was causing our code to run slower and was causing a lag on top of the lag from other elements. Removing the print statement removed that lag and helped us control our robot better.

Things that we observed and learned from while developing integrated code:

1) If statement compound condition: Previously we were using a if statement like if(b< variable < a) then do something. But we learnt that due to operator precedence in C++ this is not interpreted as a compound if condition by the compiler, and does not do what we expected to do. To resolve this we had to change the code to if (variable<a && variable>b) do something.
2) Global variable update: In our integration, there were a lot of times when we used global variables to update values from the main loop to the different functionalities. We faced a lot of issues when updating a global variable at one place had an effect on other parts of the code as well. Hence, we took special care of setting the appropriate flags and designed the global variables so that our code doesn't break while switching or using different functionalities.

Retrospective

- What you feel was most important that you learned
    - Iterative Design process
    - Time and effort involved in making robust and reliable systems.
    - Perseverance
    - Time Management
- What was the best parts of the class
    - The professor's and TA's commitment to make our experience as smooth as possible and their responsiveness on piazza
    - Group projects(Lab 4 and Lab 5) and hands on projects, tangible applications of the things we learnt in previous labs and integration
- What you had the most trouble with
    - Making robust and reliable systems
    - Integration of all individual systems
- What you wish was improved
    - More focus(effort and time) on integration
    - Thorough analysis of how to make more robust systems.
- Anything else about class.
    - Class was fun and other students in the class were really helpful.
    - Group projects were the most fun parts.
    - More time for the final competition to make a more level playing field for all teams.

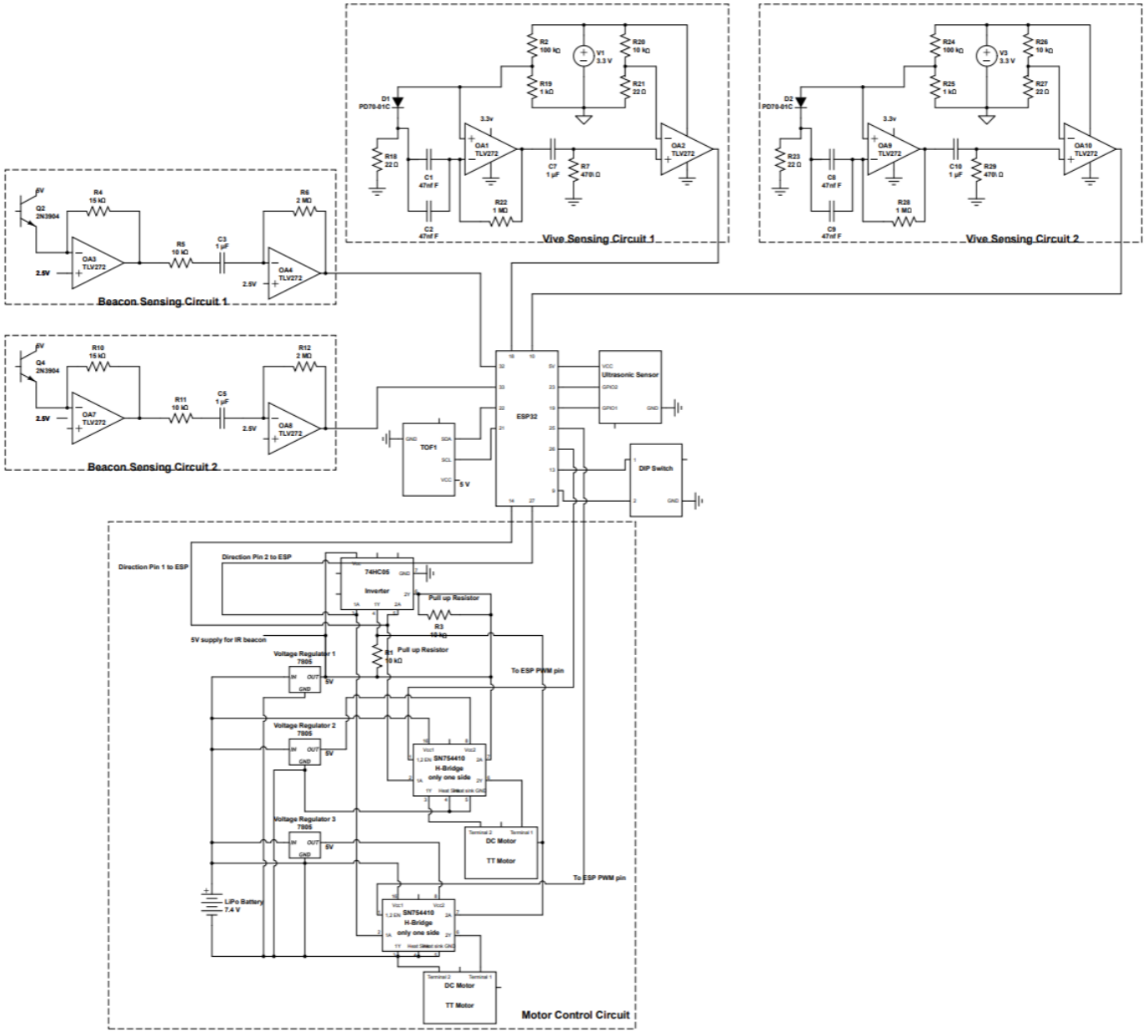
Appendix

1. Bill of materials

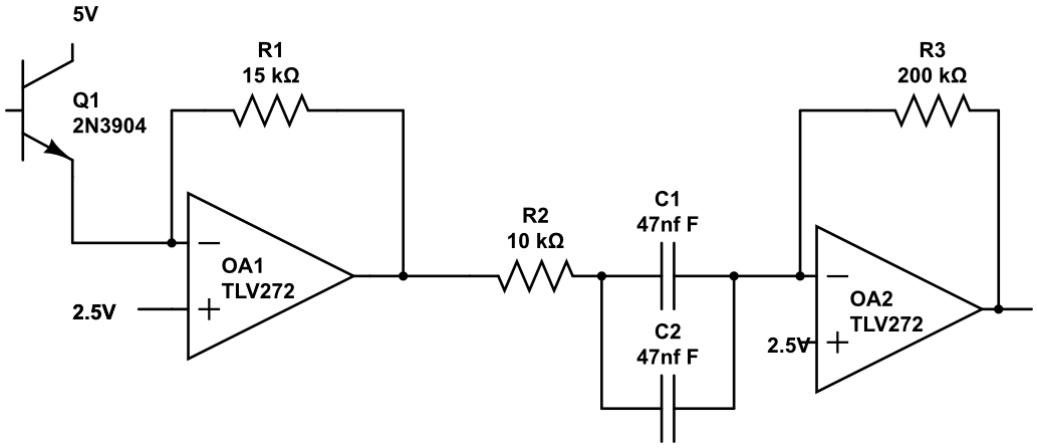
Part	Quantity	Cost ea	Cost	Link to product
Wheels	2	$ 1.5	$ 3	https://www.adafruit.com/product/3766
Motors	2	$ 21.95	$ 43.9	https://www.pololu.com/product/3205
Time of Flight sensor	2	$ 14.95	$ 29.9	https://www.adafruit.com/product/3317
7.4V LiPo battery	1	$ 10	$ 10	Purchased from Professor
Ultrasonic Distance Sensor	1	$ 3.95	3.95$	https://www.adafruit.com/product/4007
5V usb power bank	1	$ 5	$ 5	Purchased from Professor
TT DC motors	2			Supplied by Professor
Chassis Acrylic 1/8th inch thickness	1(24*18)			The Studios at Venture Labs
SN754410 HBridge	2			Ministore
Inverter 74HC05	1			Ministore
LM 7805	3			Ministore
Additional Miscellaneous hardware				Ministore

2. Electronic Circuits

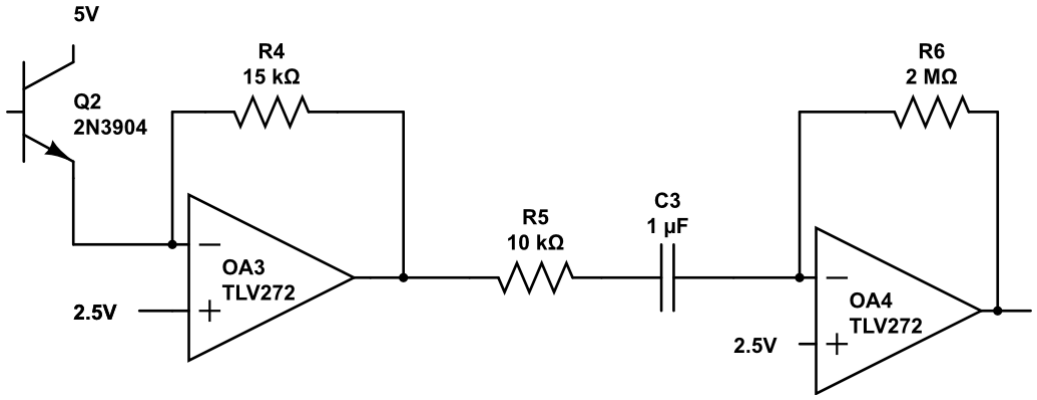
Complete circuit

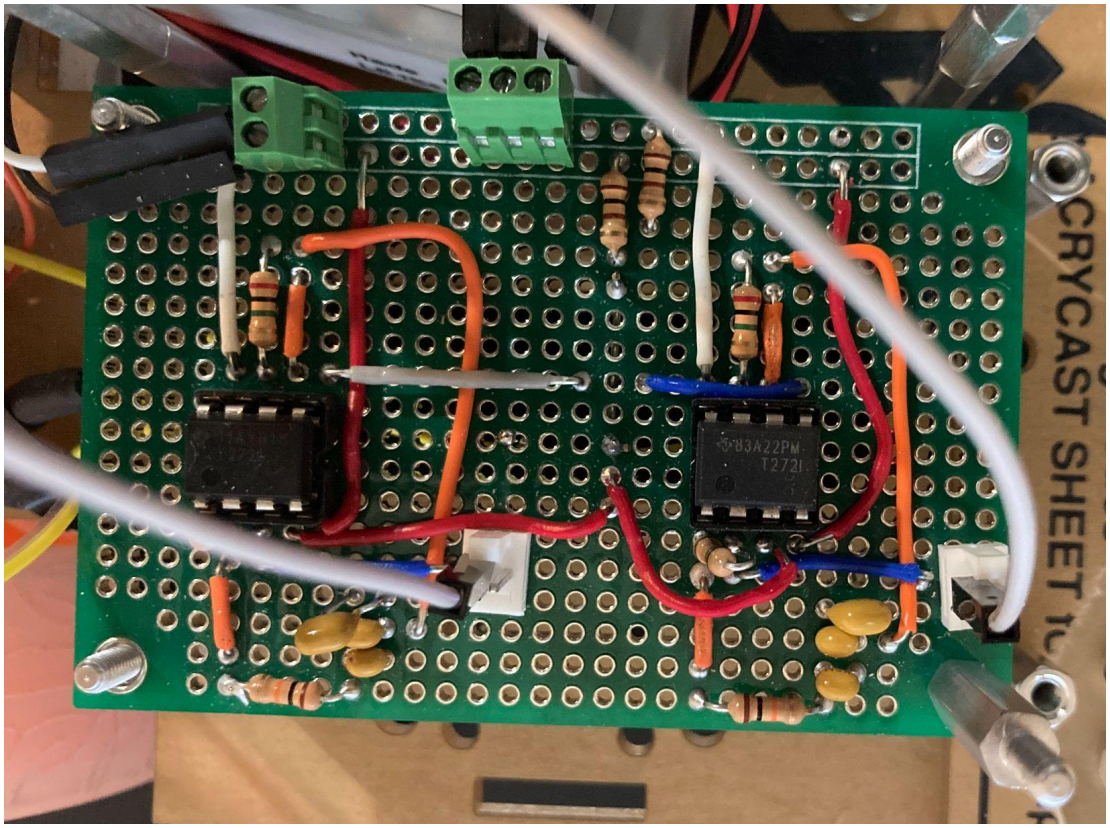


1) Beacon circuit



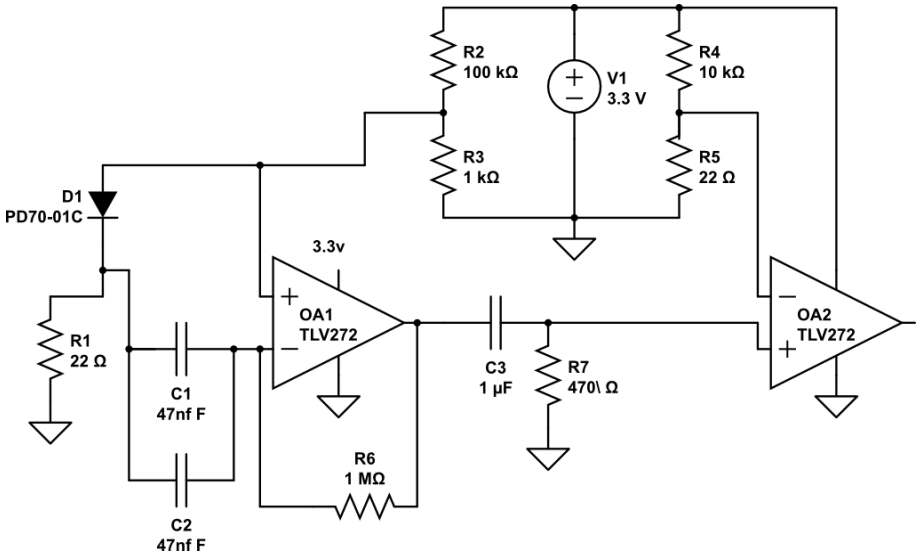
Improved Beacon circuit

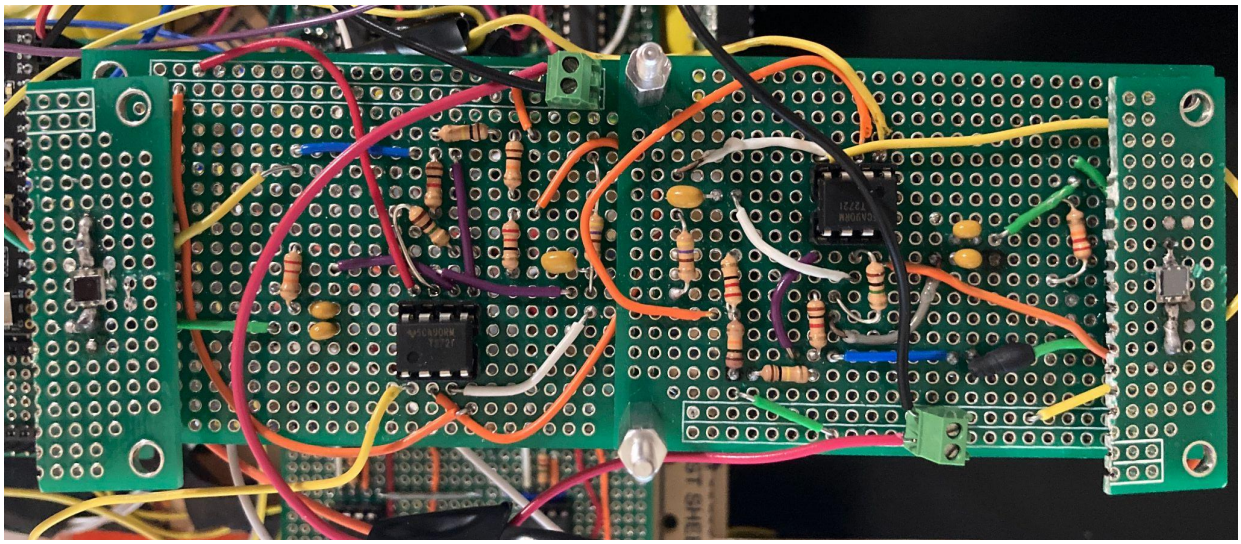




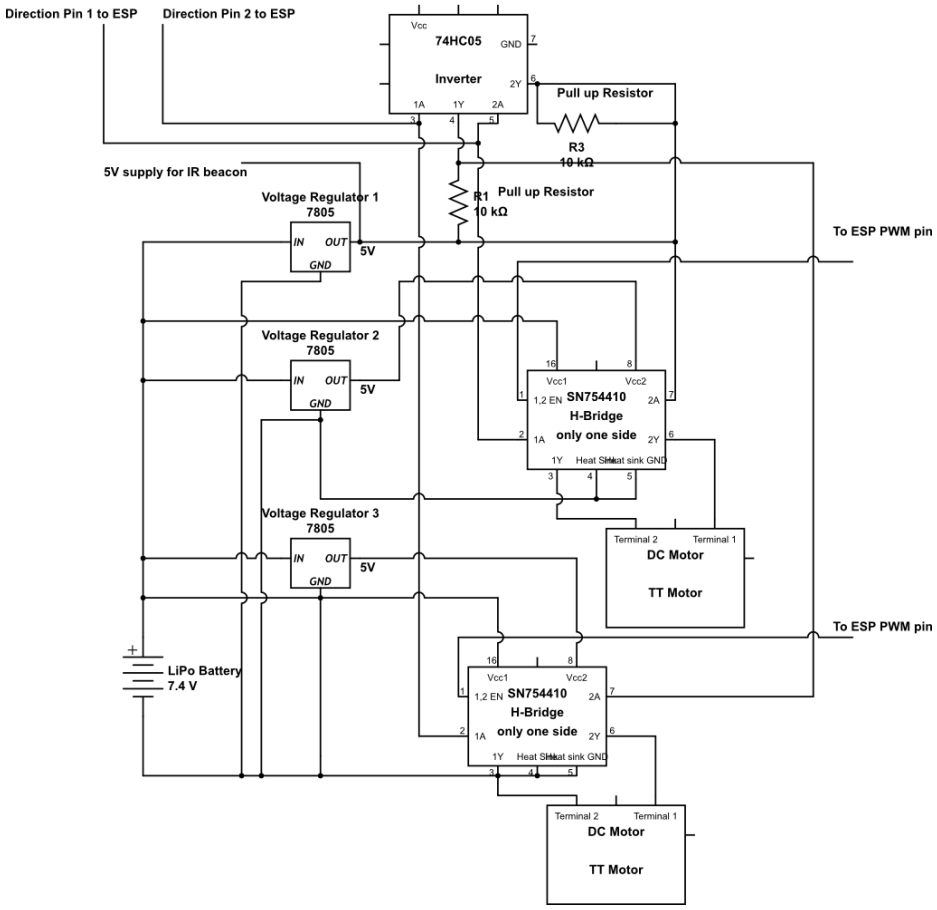
White wire(lines) coming from IR sensor output

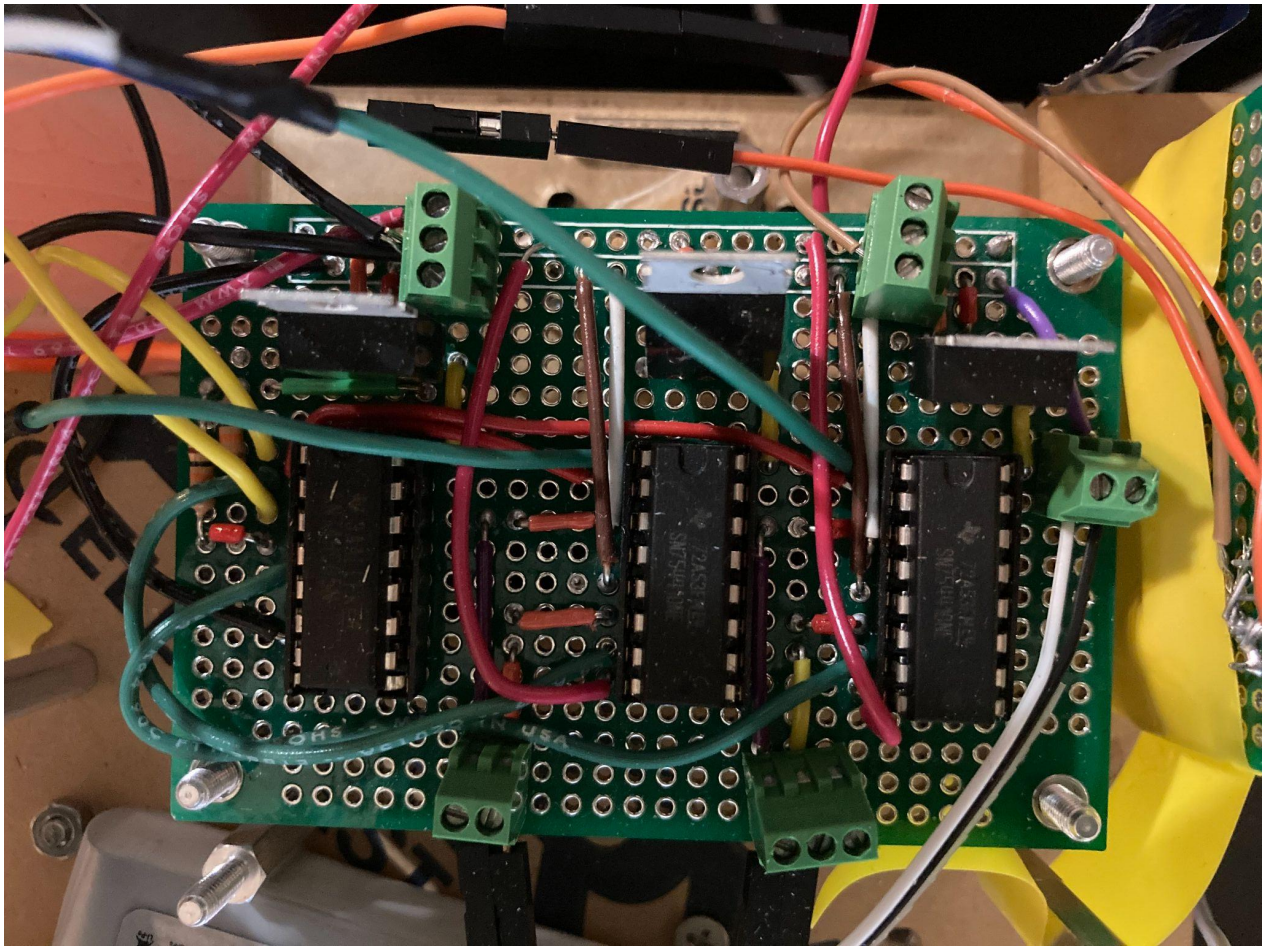
2) Vive circuit





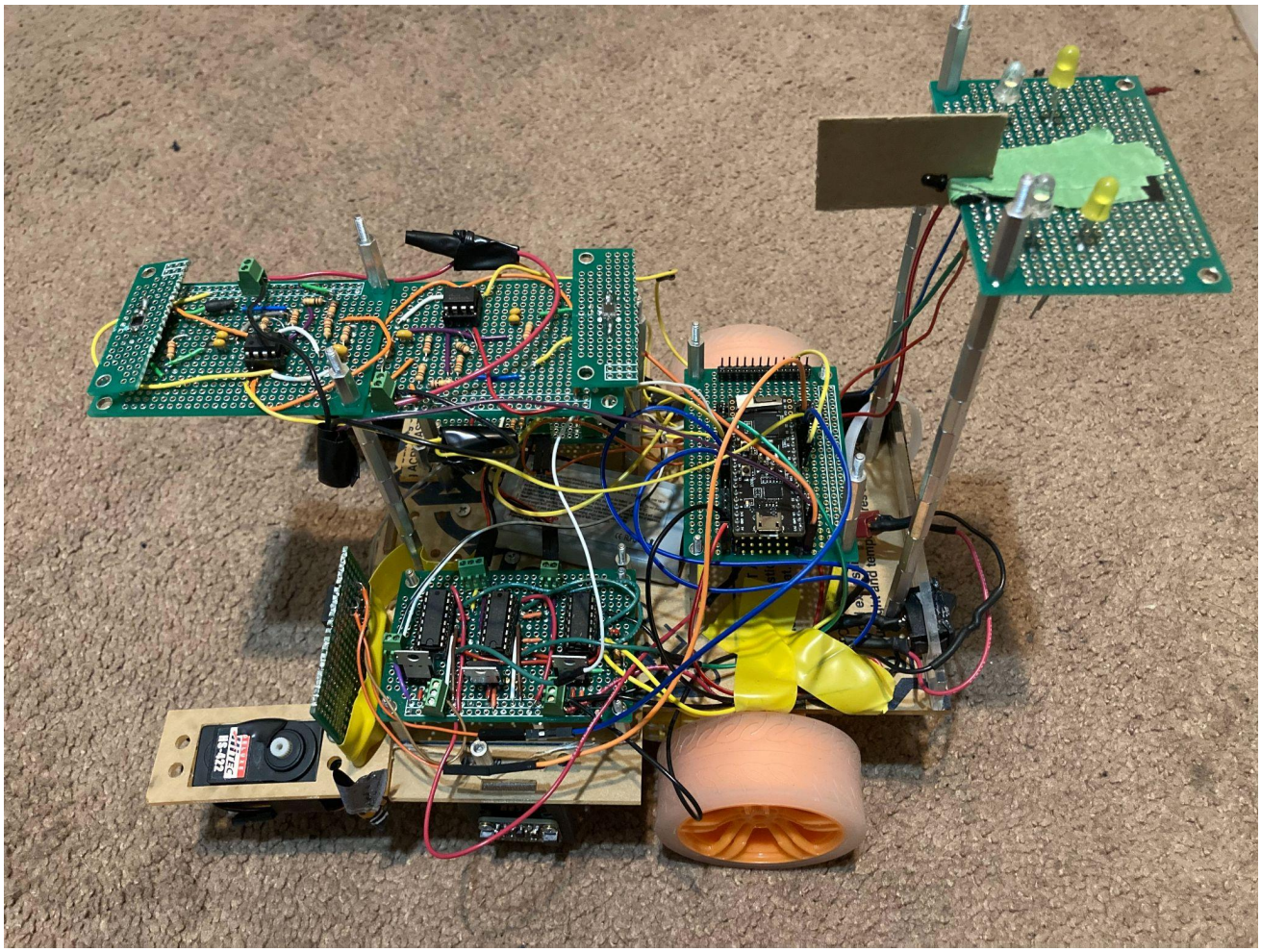
3) Power distribution and motor circuit

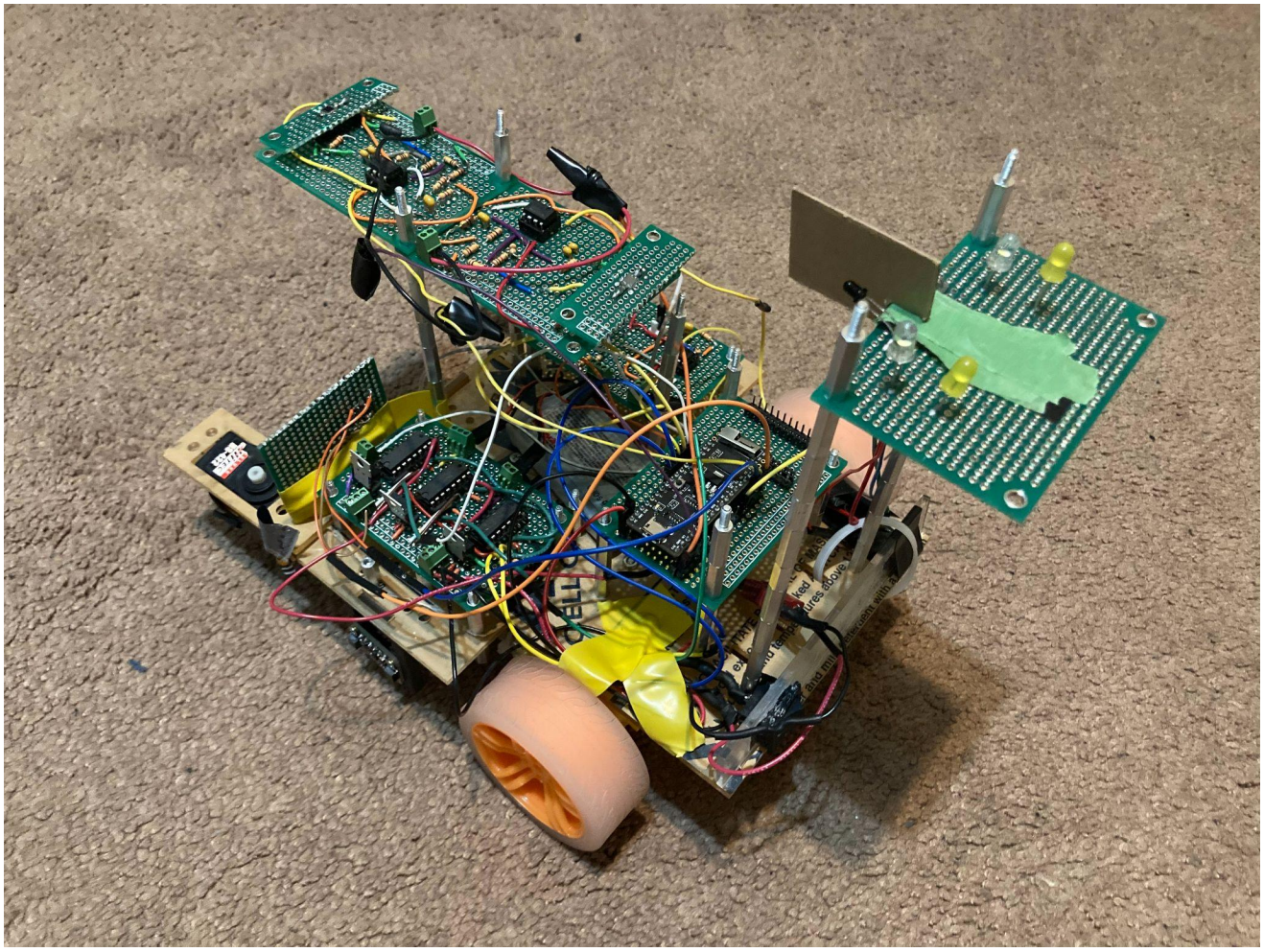




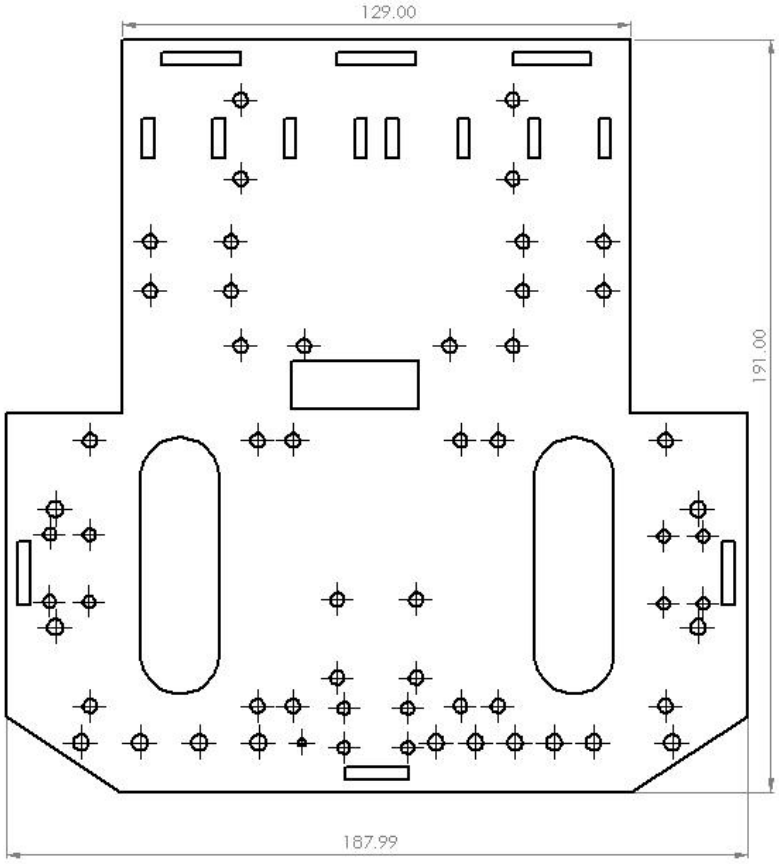
Two ICs on the left are SN754410 Hbridge and the one of right is Inverter 74HC05

3. Photo of the robot

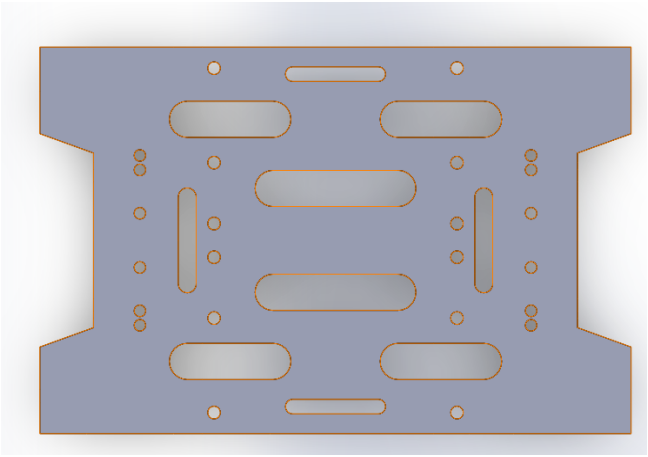




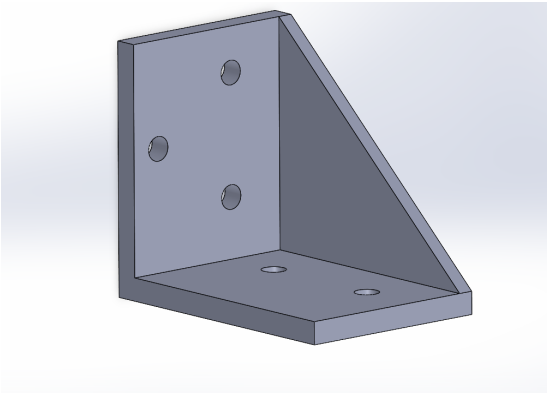
4. CAD drawings



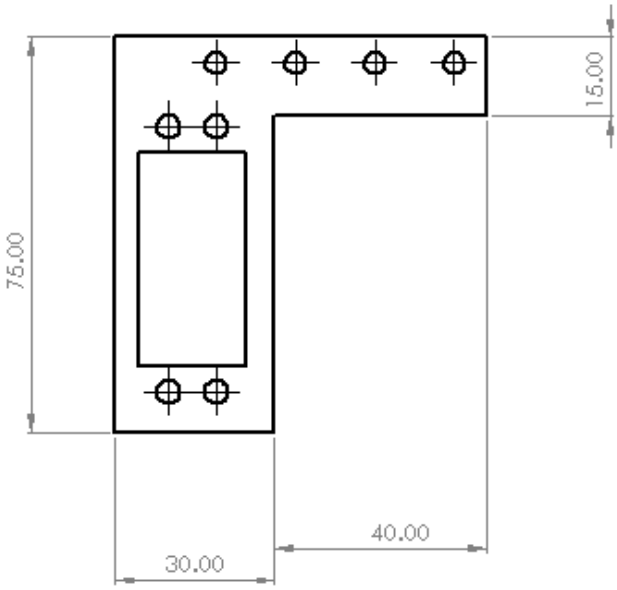
Chassis (all dimensions are in m-m)



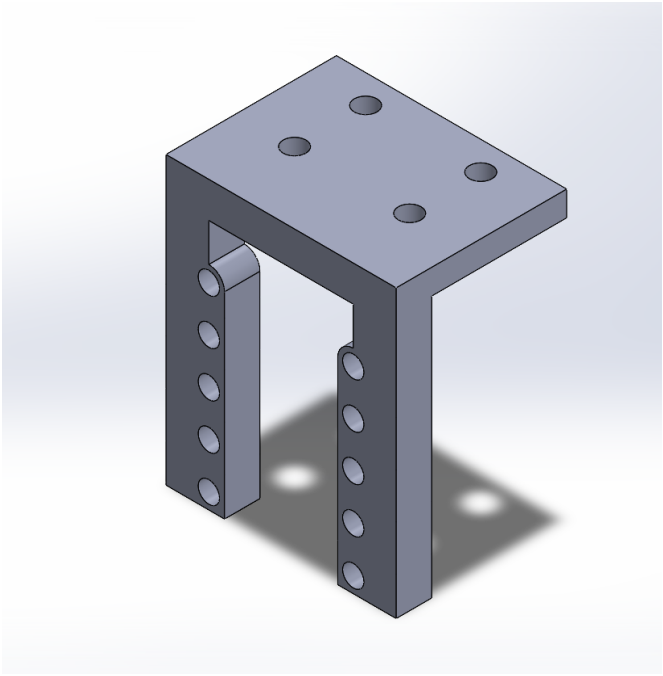
Iteration 1 Chassis Design

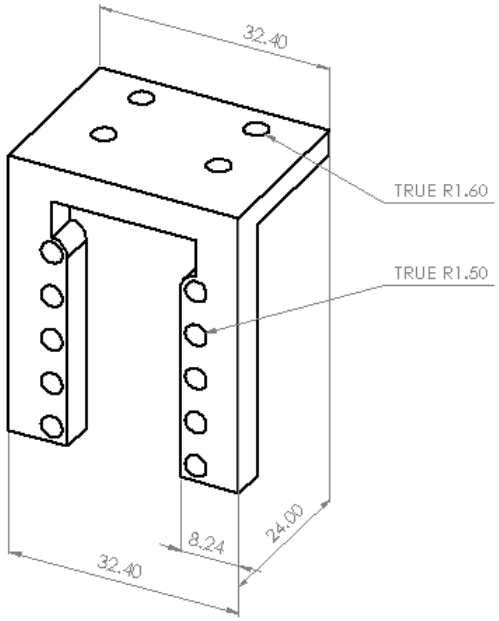


Motor Mounting 3D printed from ABS



Motor mounting for servo MG996R motor for Active Gripper





Time of Flight sensor mounting

5. Datasheets

1. Time of flight sensor: https://www.adafruit.com/product/3317

   Link to datasheet:
   https://learn.adafruit.com/adafruit-vl53l0x-micro-lidar-distance-sensor-breakout

2. Ultrasonic Distance sensor:https://www.adafruit.com/product/4007

   Link to datasheet: https://www.adafruit.com/product/4007#technical-details

6. Videos of functionality

Manual Control mode: https://youtu.be/EvGTSg6jqkc

IR Beacon Tracking: https://youtu.be/aBBqs5QNqqM

Wall Following: https://youtu.be/VHahHvvrw1c

Although we have completed check off for can following, beacon following for full field and extra credit, we were unable to take videos of the behaviour due to logistical reasons.