MEAM-510 Final Report

Group 29

Sunan Sun, Jayadev Chevireddi, Sebastian Peralta

University of Pennsylvania

MEAM-510: Design of Mechatronic Systems

12/22/2021

Content

Functionality

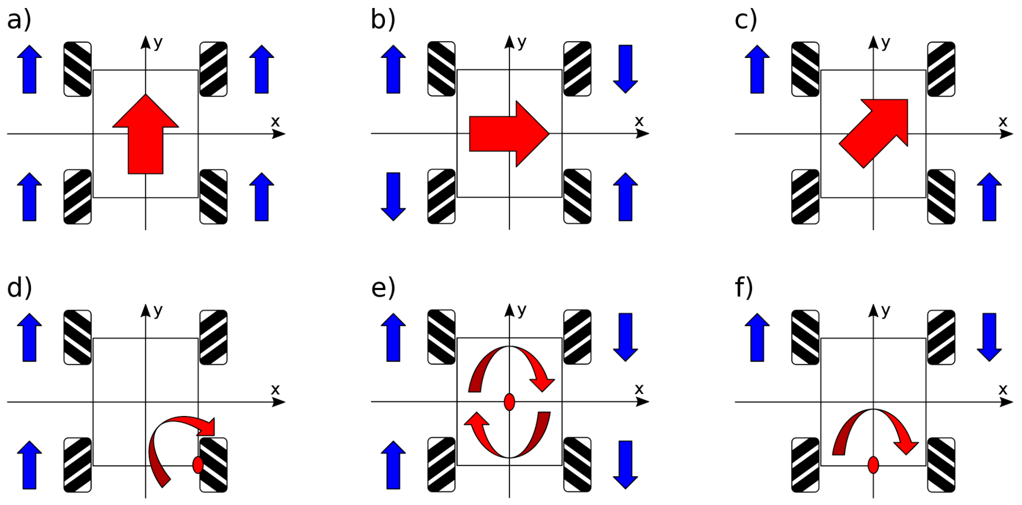
Overview

In the early stage of the project development, our team adopted the strategy to maximize the robot's performance by choosing a different mechanical design from the last lab. Previously, the robot had a differential drive system, which was able to maneuver itself over the course; however, the intrinsic difficulty in controlling the robot led our team to seek for a new solution. Our team immediately reached a consensus that the holonomic drive would be the new drive system for our robot, due to its high mobility and ease of control. Considering the fact that the final competition can be essentially divided into manual phase and autonomous phase, we believed that the holonomic drive would give us an edge over other teams in both scenarios. More specifically, the ease of manual control would not only allow us to move the robot freely, place the objects at desired locations, and block the enemy from stealing objects, but also facilitate the programming of the autonomous part, which would be discussed more in detail in the section of code architecture.

Once the control of the robot laid a solid foundation, we could then build any additional functionality on top of its high mobility and agility. We would like our robot to be competitive in the game. That being said, our team was intended to integrate all the required functionality into the robot, including the ability to move in all directions, to detect both 23Hz and 700Hz beacons across the entire field, and to move autonomously towards a given location or a stationary beacon. The omnidirectional motion of the robot can be easily achieved by differentiating the mecanum wheels of the robot. The beacon detection was designed to consist of only one IR sensor wrapped around by the insulation tape. Such design allows the IR sensor to only detect the signal incident on its tip, rather than diffuse signals from other angles. The autonomous navigation was achieved by directly using the coordinates from the Vive lighthouse. We believed that constructing another detection system specifically for cans would complicate the overall design. Hence, rather than actively detecting the cans, we stick to only using the coordinates of the objects to achieve autonomous navigation. The gripper mechanism was conceived initially, but later truncated due to time constraints.

Manual Control

One perk of employing a holonomic drive system is its ability to move in all directions. Its omnidirectional motion was achieved through different combinations of wheel directions, as shown in the diagram below.



[1]Figure 1: Movements to any directions for mecanum wheels

The change in direction of the wheels was dictated by pulse width modules(PWM) with different duty cycles. More specifically, PWMs were generated by the ESP32, using the LED control(LEC) and the signals were then directly fed into the motor driver. We used the Adafruit DRV8833 motor driver which is essentially a dual H-bridge, so that each motor driver can independently drive two motors. We powered the motor driver with a 9V alkaline battery. Because this breakout board has a voltage regulator integrated, the 9V battery can be used to supply power directly to both the motors and H-bridge. Unlike SN754410NE H-bridge, which takes one PWM to control the speed, and two logic inputs to control the direction, DRV8833 only requires two PWMs. For example, if we would like the wheel to spin forward at a certain speed, we can PWM both input pins, but one with a specified non-zero duty cycle and the other with zero duty cycle, which is essentially a low logic level. If we would like the motor to spin backward, all we need to do is just to swap the PWMs. To stop the wheel, we just PWM both input pins with zero duty cycle. In a nutshell, the duty cycle of two PWMs can completely

[1] https://en.wikipedia.org/wiki/Mecanum_wheel

determine the speed and direction of a wheel. Therefore, with eight PWM signals, we can completely control the motion of the robot.

As shown in the part c of Figure 1., the holonomic drive system is able to move in diagonal directions. However, any diagonal motion can be decomposed into a horizontal translation and vertical translation. We, therefore, only developed the algorithm to move the robot forward, backward, left and right. In terms of turning, the holonomic drive provides three distinct ways of turning: turn about the wheel, turn about the center and turn about the midpoint between two wheels. After careful consideration, our team decided to only integrate the turning about the center, which is the configuration of the part e in Figure 1. Such turning is advantageous because the robot can turn at a stationary point. That being said, when the robot makes a turn, it does not have to travel any distance in a circle like how a car would normally make a turn. In addition to four translational motions, the robot has two angular motions: left turn and right turn, and that comes to totally six allowable motions, which completely define the locomotion of the robot.

Each motion of the robot was linked to a button we created on the web interface. Hence, the user can directly press the button on the web, the command would then be passed to the robot via Wifi, and the robot would then execute the code of a specific motion and move accordingly. The greater detail of the programming, web interface, and TCP protocol would be discussed in the section of code architecture. At this point, the manual control of the robot is fully achieved.

Wall Following

Four distinct motions are required for the implementation of the wall following: forward, left turn about the center, left up and right up. The corresponding wheel combinations are also shown in figure 1. Two ultrasonic ping sensors were used to detect the distance between the wall and the robot by sending and receiving ultrasonic waves and calculating the distance based on the time it takes at the speed of sound. One pin sensor is located on the right side of the robot to determine its distance from the side wall and the other pin sensor is located at the front of the robot to determine its distance from the front wall.

The sequence of operations is as follows:

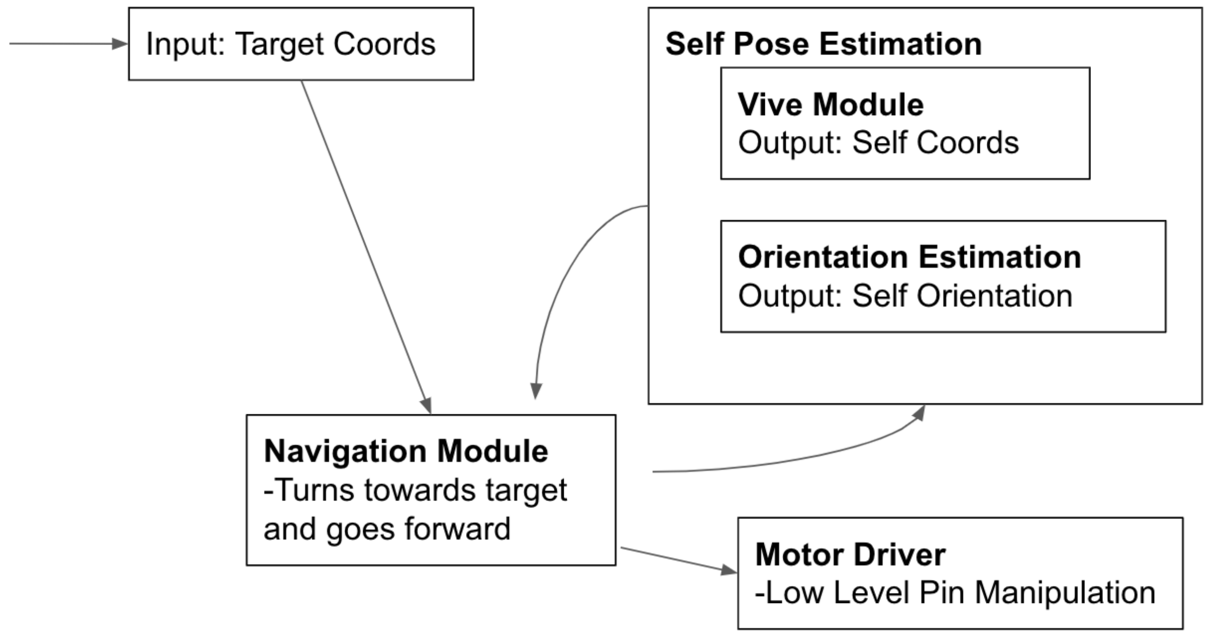
1. Prior to moving forward from its starting point, the robot would intentionally move right and hit the wall to force adjust its position so the robot aligns with the wall. This process is a simple implementation to make sure the robot is parallel to the wall as we are not using any rotations to adjust the pose explicitly. The robot will then move left away from the wall until reaching a certain distance.

2. Then the robot starts moving in the forward direction while simultaneously getting the values from the front and side walls. If there is any deviation from the right wall either towards or away from it, the robot will translate accordingly to compensate for the deviation.

3. When the forward ping sensor measures a distance of less than a certain threshold value, the robot will stop moving forward and start making a 90 degree left turn. The time it takes to make an exact 90 degree turn was found from trial and error. Afterwards, the robot again performs the first operation until it aligns with the wall and gets ready to move forward.

The detail of the wall following code was discussed in the section of the software approach.

Autonomous Navigation



The autonomous navigation module includes a navigation module, self pose estimation module, and motor driver module. It takes a set of target coordinates and continuously estimates its pose, and navigates towards the target coordinates. The output of the navigation submodule is a command to either turn left, right, or go forward which is then translated into low-level pin manipulation and PWM signals by the motor driver. The self pose estimation module consists of a vive submodule which receives coordinates of the robots current position from the vive tracker using a vive circuit and arduino library. The orientation estimation module calculates a vector of the robot's direction by first receiving a position coordinate from vive, driving forward, then receiving a second position coordinate from the vive. A benefit of this scheme is that only one vive circuit is required. However, as expected, a single sensor created a fair-amount of uncertainty in the orientation estimation and so we believe faster autonomous navigation could be achieved with a dual vive circuit setup.

Beacon Tracking

For the beacon tracking a single IR sensor was designed to be capable of sensing both 23 Hz and 700 Hz. The beacon tracking functionality was integrated and linked to a button on the web interface that was created to execute the code.

The main goal for this functionality was to track the beacon that could be at most 13ft from the robot. This required our IR sensors to track the beacons at a large distance and orient the robot to face towards the beacon anywhere in the field. The orientation of the sensors was done by actually rotating the robot until any of the sensors found the beacon.

The process goes as follows:

1. The robot has a beacon tracking circuit that is fixed on the robot and its height is equivalent to that of the beacon height. The robot first rotates (left) around its center of mass which automatically rotates the beacon tracking circuit that is placed on the robot.

2. The robot will keep turning until the single IR sensor sees the beacon, and the detail of such a detection circuit will be discussed in greater detail in the section of the electrical design.

3. Once the IR sensor locks on the beacon, the robot will then move towards the beacon. We also integrated the front pin sensor, so the robot will keep moving forward until the beacon is detected in the vicinity of the robot.

Single IR sensor reduces the complexity in circuit and makes the sequence of programming compact.

Mechanical Design

Mechanical Performance

As discussed in the section of the manual control, we assigned overall six allowable motions to the robot: right, left, forward, backward, right turn and left turn. Ideally, for any translational motion, the robot was expected to move in a straight line. While in practice, the wheels tended to slip and the robot would slightly drift in lateral direction while moving forward. The problem was then examined, and it quickly became apparent that the low friction between the wheels and the ground was the culprit. First of all, the surface of the field is quite smooth. But most importantly, the rollers on the mecanum wheels, though rubberized, are texture-less, as shown in the figure below. It means that the grip force the wheels exert on the ground is limited, and such force can be deteriorated even more after the wheels pick up dust on them, which can further decrease the friction.

[2]Figure 2. Textureless rollers on mecanum wheel

Such drifting is clearly undesired, since the robot can eventually end up at an entirely different location from the specified one. Our team immediately came up with two solutions to tackle this problem; one is to somehow increase the friction between the wheels and the ground, and another is to introduce a feedback system to constantly adjust the motion of the robot. Eventually, we chose to develop a feedback algorithm that can constantly calibrate the pose of

[2] https://store.dji.com/product/robomaster-s1-mecanum-wheel

the robot, so the robot can quickly adjust itself back to the intended path if any deviation is detected. The programming of this calibration algorithm will be discussed more in the section of the code architecture, but the takeaway is that the mechanical performance of the robot was then greatly improved with the assistance of the software programming. Despite the existence of the inevitable drifting, the robot was now able to constantly adjust itself and move in any translational direction as expected.

Ideally, our robot would turn about its center; i.e. after any turn is made, the robot should still remain at its original location. While in practice, turning is also subject to the low friction, and the robot would inevitably deviate from its original locations while turning. However, such deviation turns out to be less than a problem for our robot. Mainly due to the fact that turning is nearly exclusively used for beacon tracking. In searching for the beacon, the robot would keep turning until the sensor sees the beacon, and move towards the beacon afterwards. Now because of the low friction, the robot may slip to some other location, but the beacon detection system guarantees that when the robot senses the beacon and stops turning, it will always face towards the beacon. That being said, regardless of where this new position of the robot is, as long as the orientation is correct, the robot can always move towards and reach the object. Regarding autonomous navigation, any path from the robot to the target location can be decomposed into one translation in x-direction followed by another translation in y-direction. Hence, no turning is explicitly involved, except some minute turns made to adjust the robot's pose during the feedback control. Such small-scale turns are less subject to the low friction, and did not exhibit any odd or undesired behaviour in practice.

Initially, we were worried about the performance of the motors, because all four motors would be powered by a single 9V alkaline battery, any excessive voltage drop along the circuit could potentially cause the motor to underperform. It was conceived that if the 9V battery failed to bring enough power to the motors, we might add a boost converter to step up the voltage. Fortunately, in practice, all four motors were able to operate normally. As a matter of fact, the motors would spin too fast if run at full duty cycle, and we had to PWM the motor at a lower duty cycle.

Electrical Design

Motor Driver

The specific motor driver we used is DRV8833 from Adafruit. As shown in the figure below, this motor driver contains a dual H-bridge, which can power two motors independently. Since our robot has a total of four motors to run, we procured two of these motor drivers to completely control the robot. The 9V power supply was directly connected to the VMotor pins, which would not only power the motor, but also power the H-bridge itself through an integrated voltage regulator. Even though the entire breakout board is powered, the chip is still inactive by default. To enable the driver, we directly bridged VM and SLP pins, where the VM pin would supply a logic high signal to the SLP pin, which would then turn off the sleep mode. AOUT and BOUT pins are connected to the motors. AIN and BIN pins are connected to the ESP32, which would PWM to those input pins to control the speed and direction of the motor. The detail of the approach was already discussed in the section of functionality.

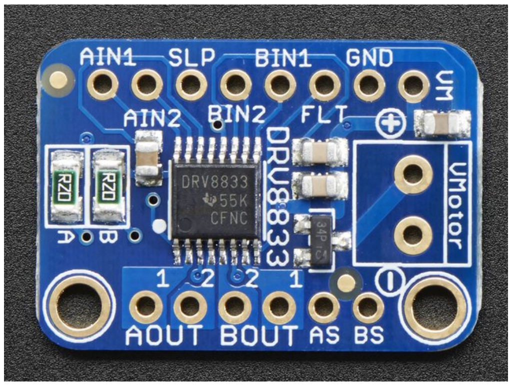


Figure 3. Adafruit DRV8833 DC/Stepper Motor Driver Breakout Board[3]

The intended performance of the motor driver was to control all four motors independently at different speeds and directions. As the results turned out, the motor driver was fully able to achieve the desired results. No apparent noise or any odd behaviour was spotted. With two breakout boards and eight PWM output pins, the motor driver laid a good foundation, and indeed promised both the manual control and autonomous navigation of the robot.

[3] https://learn.adafruit.com/adafruit-drv8833-dc-stepper-motor-driver-breakout-board?view=all

Vive lighthouse

The vive circuit turned out to be one of the modules that was most prone to the noise. The intended performance of the vive is simple and straightforward – receiving x-coordinate and y-coordinate of the robot anywhere in the field. As mentioned in the section of functionality, we did not construct any circuit to actively detect cans, Hence, our approach to the autonomous navigation was to utilize the coordinates of the objects and the robot. The coordinates of cans were given. That being said, the success rate of the autonomous navigation would fully rely on the accuracy of the coordinates of the robot. If the vive circuit fails to retrieve the proper coordinates of the robot, the robot may use that wrong location to orient itself towards a wrong target.

To avoid the potential noise in any sensing circuit, we used two separate power supplies; one 9V alkaline battery for motors, and one portable battery for the rest of the modules. Nevertheless, when the vive circuit was first constructed, the circuit was unable to lock on the vive lighthouse. However, just by wiggling the wire a bit, the circuit would magically start working again. We believed that it was the inductive noise that kicked in due to the large loop area in the vive detection circuit. While we intended to add shielding to reduce noise, we were able to have the vive work consistently just by manually moving the wires on the circuit further away from each other. Though not the most robust solution, our team decided to move along due to time constraints.

While the circuit was able to receive data consistently, the accuracy of the data still posed a huge challenge. From time to time, the coordinates retrieved from the vive were off. For example, when the robot is directly underneath the lighthouse, the vive circuit would sometimes output (8000,8000), which was clearly wrong. We eventually discovered that the distance between the photodiode and the vive lighthouse dictates the data accuracy. Initially, the photodiode was placed five inches above the ground. Later on, after we placed the photodiode one foot above the ground, the accuracy was greatly improved and the robot hardly received any wrong data. The accuracy of the vive circuit was further manifested in the autonomous navigation, where the robot was able to retrieve its proper location and move towards the target.

Range sensors

The specific range sensors that were used was RCWL-1601. The range of these sensors is 2cm to 450 cm. But it was recommended to use it for a range between 2 and 250 cm for the best results. The width of the playing field is way less than the range of these sensors making them desirable to use. These were picked instead of IR or TOF sensors as they were most easy to work with. The sole purpose of these sensors was in wall following and object collision detection.

Figure 4. Adafruit RCWL-1601

While these sensors were easy to use they were prone to more noise and the outliers from the sensor data were quite frequent which made them undesirable at first. A simple moving average and median filter was able to mitigate the noise to an outstanding degree.

Integrating these sensors into the robot for the wall following required the placement of the sensors in an appropriate position with respect to the robot and also each other as these sensors might interfere with each other giving undesirable readings. The most difficult part in the wall following was to make the left turn when the front sensor faces the wall. We first tried a number of different time intervals of rotating the robot to turn exactly 90 degrees. This was not possible for most of the time due to the slip between the tyres and the random bumps in the field which made such a precise turning movement challenging and almost impossible.

We were able to overcome this by rotating the angle so that it approximately turns 90 degrees and then it hits the wall by moving sideways, to make itself align parallel to the wall

IR sensors

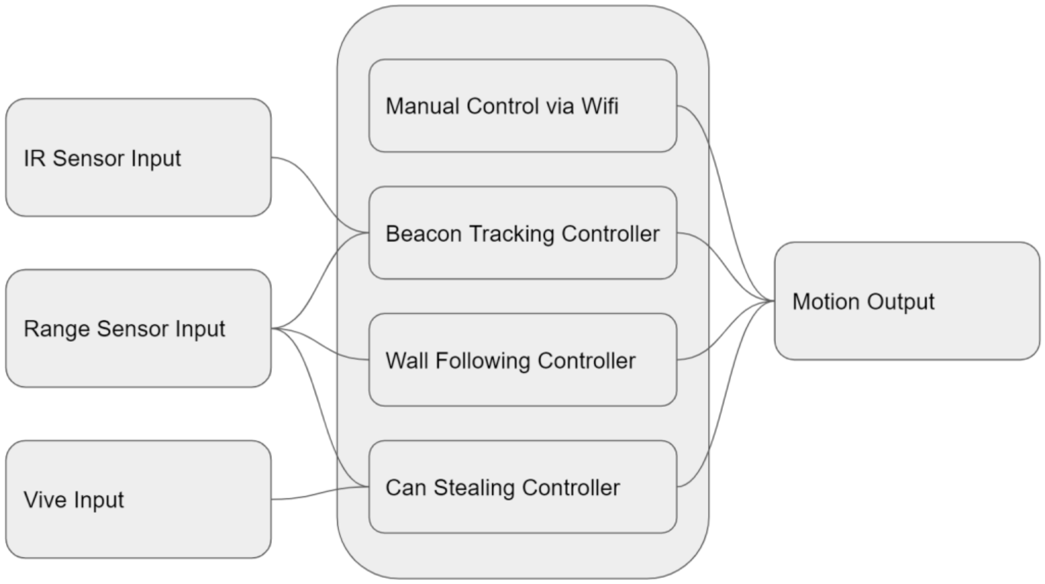
The beacon detection circuit can be essentially divided into two distinct criterions: range and orientation. In other words, the IR sensors were expected to sense the beacon across the entire field and precisely identify the orientation of the beacon with respect to the robot's location. Problems were encountered in meeting the required range and orientation. In terms of the range, the extreme distance was 13 feet which is the diagonal distance across the field. Initially, the prototype circuit was able to detect the beacons only about 3 feet away. And no signal would be sensed any further. To improve the performance of ranging, different resistors and capacitors in the circuit were attempted until the best combination was found. The detailed resistor and capacitor value used were shown in the schematics in the appendix. Eventually, a single IR sensor was able to detect signals in both 23Hz and 700Hz from 13 feet away.

The initial design was to use two IR sensors with a separator in between, and the robot is facing towards the beacon only when both sensors detect signals. However, in practice, the beam width of the IR sensor was too wide. In other words, the IR phototransistor can still detect the beacon even when the signal is coming from, for example, 50 degrees away from the tip of the phototransistor. With such a wide beam width, the robot may easily fail to face towards the beacons even if both IR sensors are detecting the signal. Rather than adding a separator, we later came up with an easy but effective solution to narrow down the acceptable beam width, which is to wrap around the phototransistor with duct tape, but leave its tip exposed. The duct tape was intended to block all the diffuse signals from any other directions, and the exposed tip was supposed to only allow the incident signal. The test result turned out to be quite appealing. The duct tape indeed successfully blocked all the diffuse signals coming from other angles, and only the signal incident on the tip of the phototransistor was detected. Even at the extreme distance of 13 feet, the IR sensor was still able to precisely identify the correct orientation of the beacon. Such design did not only increase the accuracy in orientation, but also decreased complexity in design, reducing two IR sensors to just one.

Processor Architecture and Code Architecture

Block diagram and Port Allocation

The middle block represents the ESP32 which essentially contains four distinct modules, each representing the required functionality of the robot. On the left side, three blocks indicate the inputs that the robot receives, which are respectively IR signal, range sensor signal and vive signal. Each signal will be input into different modules on the ESP32, which would then make corresponding output, which can be essentially categorized to one single output – motion output as shown on the right side of the block diagram.



Below is a brief overview of port allocation. We were able to make the use of pins of a single ESP32 without introducing a second ESP32:

- 8 PWM for 4 DC motors: 19/23/18/5/25/26/32/33
- 4 pins for pin sensors: 9/10/5/8
- 1 Vive Input: 27
- DIP Switches: 37/38/34/35
- 1 Beacon tracking: 13

Manual Control

As discussed many times in previous sections, the control of the speed and direction of the wheels are dictated by the duty cycle of the PWMs fed into the motor driver. The PWMs were generated by ESP32 using the LED control. The detailed procedure was shown in the pseudo code below. Once the resolution bit, the resolution in decimal and frequency were defined, we can set up an LEDC channel and assign that channel to any specific output pin. This output pin on ESP32 would then be directly connected to the input pins on the motor driver. After the setup was done, the signal was generated executing the command ledcWrite which takes the LEDC channel and duty cycle as two arguments. For example, if we would like the robot to move forward at a certain speed, all the wheels were supposed to move in the same direction. As shown below, we assigned a non-zero value to the variable dutyCycle, and a zero to the variable dutyCycle0, and executed the command ledcWrtie to PWM the input pins on the motor driver to achieve the desired motion.

```
#define ledcResolutionBit 15
#define ledcResolution ((1<<ledcResolutionBit)-1)
#define frequency 5

Void move forward(){
  ledcWrite(ledcChFrontRight1, dutyCycle);
  ledcWrite(ledcChFrontRight2, dutyCycle0);
  ledcWrite(ledcChRearLeft1, dutyCycle);
  ledcWrite(ledcChRearLeft2, dutyCycle0);
  ledcWrite(ledcChFrontLeft1, dutyCycle);
  ledcWrite(ledcChFrontLeft2, dutyCycle0);
  ledcWrite(ledcChRearRight1, dutyCycle);
  ledcWrite(ledcChRearRight2, dutyCycle0);
}

Void setup(){
  ledcSetup(ledcChFrontRight1, frequency, ledcResolutionBit);
  ledcAttachPin(ledcPinFrontRight2, ledcChFrontRight2);
}
```

Autonomous Navigation

The autonomous navigation consists of two infinity loops; one for translational motion in x-direction and one for translational motion in y-direction. When the program enters the first loop, the robot will constantly compare its current location to the target location. If the difference is greater than 100, it means the target is on the left of the robot, and the robot would then move to the left, and vice versa if the difference is greater than -100. Anything in between will indicate that the x-coordinates of the target and robot are matched, and the program will exit the first while loop and enter the next one. Similar to the translation in x-direction, we would like the robot to keep moving forward until the y-coordinates of the target and robot are matched.

```
Void autonomous_navigtaion(){
  while(1){
    If (x_robot-x_target) > 100 {
      moveleft();
    }
    Else if (x_robot-x_target) < -100{
      moveright();
    }
    Else {
      break;
    }
  while(1){
    If (y_robot-y_target) > 100 {
      calibrate();
      moveforward();
    }
    Else {
      break;
    }
}
```

One worth noting point from the pseudo code above is the calibration function following the moveforward. This function was the feedback control algorithm mentioned in the section of the mechanical performance. Recalling that our team did not construct any detection circuit to actively search for cans, the robot can start deviating from its original path due to the low friction between the wheels and the ground. Hence, a feedback algorithm called calibration was introduced while moving forward. Apparently, such calibration can be easily achieved if two vive circuits were constructed, which would generate an orientation vector for adjustment. However, the downside for building two vive circuits is time-consuming and more prone to potential mistakes. Our team eventually settled down with one single vive circuit, and programmed the software to compensate. As shown in the pseudo code below, we retrieved one x-coordinate, let the robot move forward for a small distance, and then retrieved a new x-coordinate. We then computed the difference between two x-coordinates. Ideally, if no deviation occurs while the robot moves in true y-direction, the difference between x-coordinates should be zero. If any difference is detected, the robot will then turn accordingly and only exit the loop when x-coordinates are matched.

```
Void calibrate(){
  Do{
    x_coord_prev = vive1.xCoord();
    moveforward();
    x_coord = vive1.xCoord();
    x_diff = x_coord - x_coord_prev;
    If (x_diff > 50){
      turnleft();
    }
    If (x_diff < -50){
      turnright();
    }
    Else{
      break;
    }
  }while(1);
}
```

Beacon Tracking

Although the electrical part of the beacon tracking was a huge challenge over the course of the project, the software approach of the beacon tracking was surprisingly easy and straightforward to implement. As shown in the pseudo code below, the code structure consists of a single infinite loop which would only exit when the robot reaches the beacon. The first condition in the if statement says that when the IR sensor detects any signal at about 23Hz, the robot should start moving forward. The second if condition states that the robot would keep moving forward until the front pin sensor senses the beacon in its vicinity. The third if condition states that if at any moment when the IR sensor loses the track of the beacon, it would then start turning and would only stop when the IR sensor locks on the beacon again.

It is worth noting that the calibration function used in the autonomous navigation part was not implemented here, since the calibration function would be an overkill in this scenario. Say the robot actually deviates while moving forwards without an explicit feedback control. The robot would surely lose the signal from the beacon, but the third if condition would be triggered and the robot will then start turning and actively look for the signal until the beacon is locked on again. Hence, the beacon tracking does not need any extra feedback control, and the robot is able to adjust its pose to always move towards the object.

```
Void beacon_23hz(){
  while(1){
    if(freq <= 24 && freq >= 22){
      moveforward();
    }
    Else if(front_sensor < 100){
      break;
    }
    Else if(freq > 24 || freq <22){
      leftturn();
    }
  }
}
```

Wall Following

The wall following algorithm consists of one single while loop, which is essentially a for loop with a counter. Obviously from the pseudo code below, the counter will increment every time the robot makes a turn at the corner. And the whole program will stop after the robot makes four turns as required.

As seen in the below pseudocode there are four conditional statements that determine the movement of the robot based on the distances the front and the right ultrasonic sensor measures. When the wall following is started first the robot moves right, hits the wall and moves left to self align. If the robot is at a distance between 17 and 23 as seen from the right sensor, it moves forward. If the robot is greater than 23 as seen from the right sensor the robot moves front and right diagonally. Similarly for the left deviation. If the front sensor measures a distance of less than 10 cm the robot turns, moves right, hits the wall causing self alignment and moves left to a distance between 17 and 23.

```
counter = 0;
Void wall_following(){
  while (counter < 4){
    if(right_sensor <= 17){
      moveupleft();
    }
    Else if(right_sensor >= 23){
      moveupright();
    }
    Else if(right_sensor > 17 && right_sensor < 23){
      moveforward();
    }
    Else if(front_sensor <= 10){
      turnleft();
      delay();
      moveright();
      delay();
      moveleft();
      delay();
      counter++;
    }
}
```

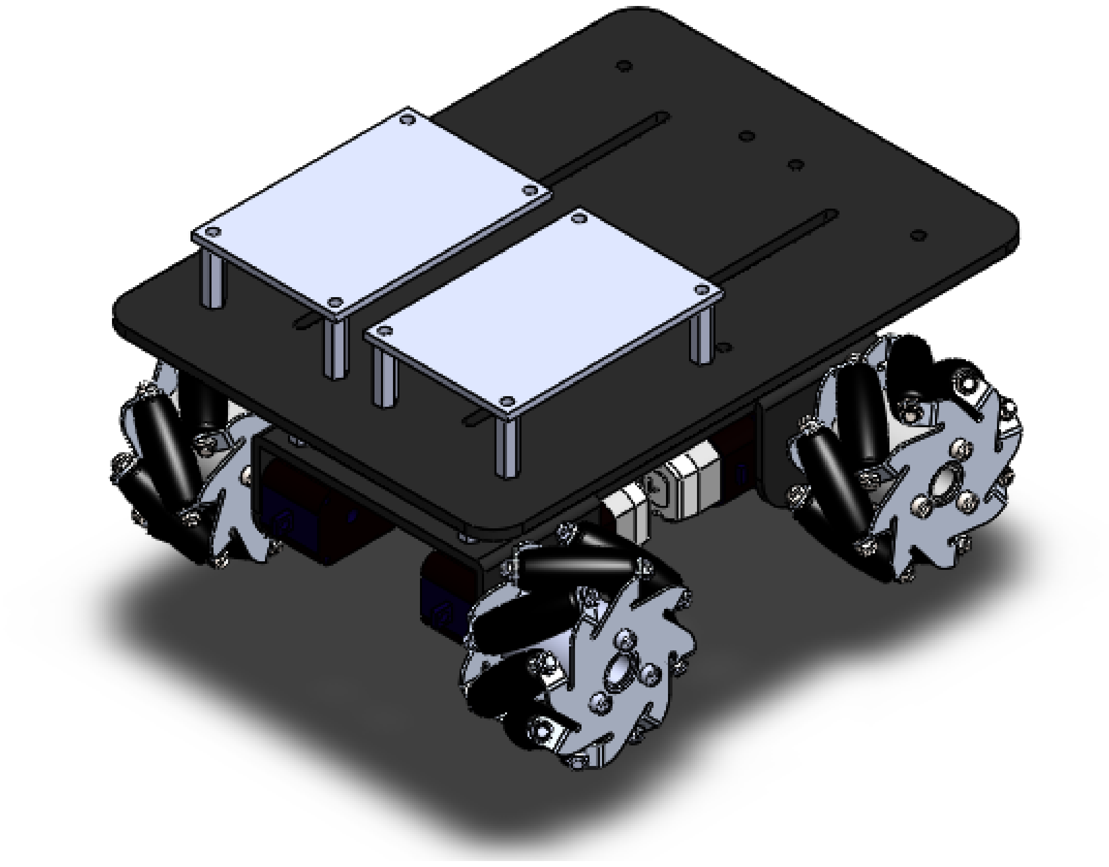
Retrospective

Of all the things we learned in this class, we feel that the biggest takeaway is a general skill for how to implement robotic solutions. Whether it is programming microcontrollers, selecting sensors that meet our specifications, or implementing communication protocols, this class has given us the necessary exposure to feel confident and secure to design and execute new solutions to future problems.

Our favorite things about the course have to be integrating all the skills and microsystems we learned over the semester to synthesize something brand new in labs 4 and 5. It is not lost on us, the universality of designing an autonomous system to compete in a game with established rules. Unsurprisingly, these two labs were the most difficult which we appreciate given that any real-world integrated solution is likely to be even more involved.

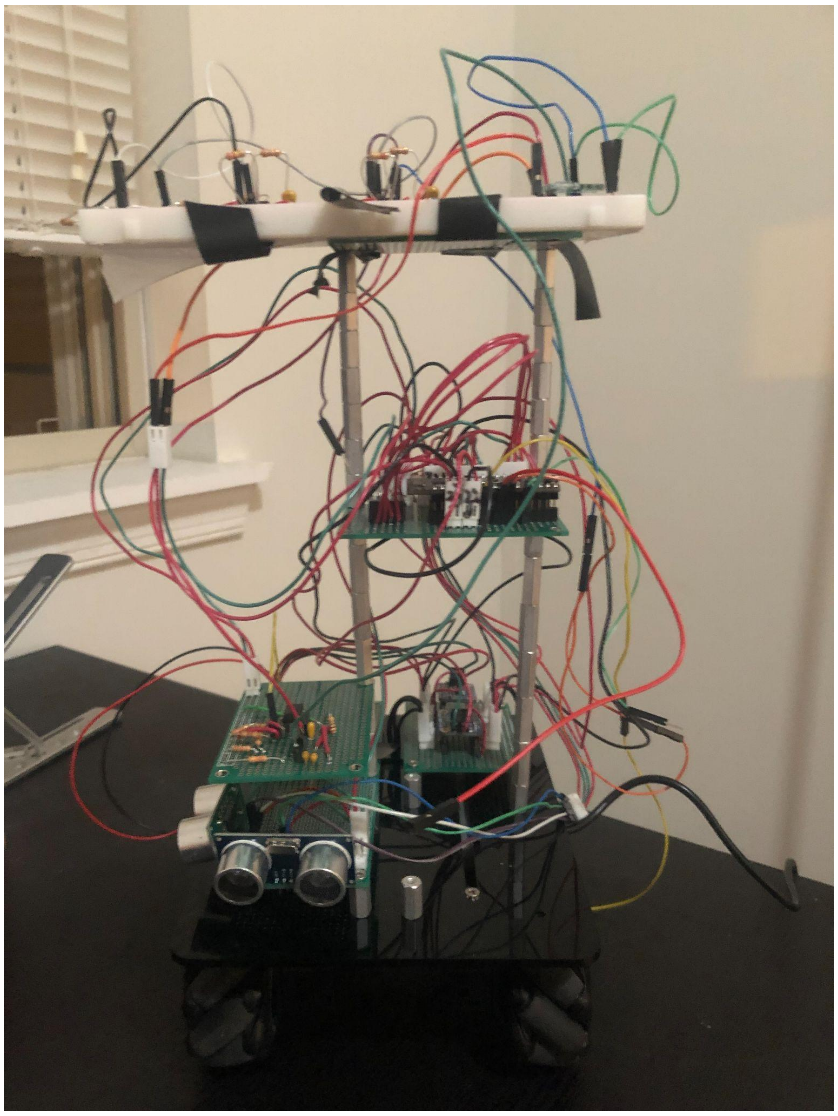
Appendix

CAD Model



Four motors are attached to the chassis underneath the top platform. The perfboards are connected to the top platform. The top platform was laser cut to fit the perfboard and any other electrical and mechanical components.

Photos



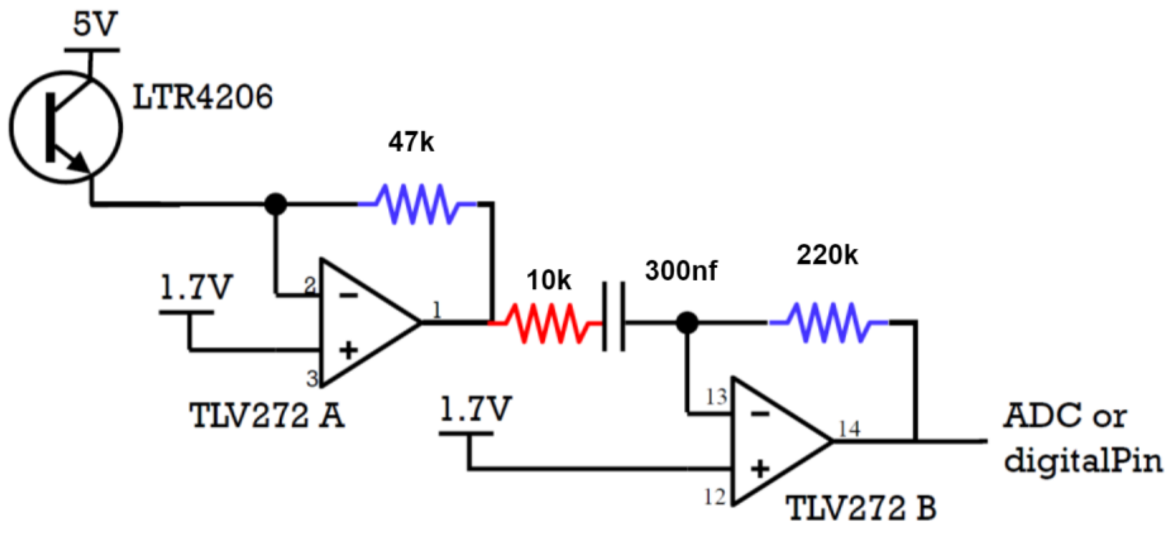
This is the front view of our robot. Two ping sensors are clearly spotted. The motor drivers sit adjacent to the ping sensor circuits. Above the spin sensor is the vive circuit, where the photodiode was later elevated at 1 foot above the ground. The ESP 32 sits above the motor driver and below the beacon tracking circuit. The beacon tracking circuit sits on the top of the platform to match the height of the beacon.

Bill of Material

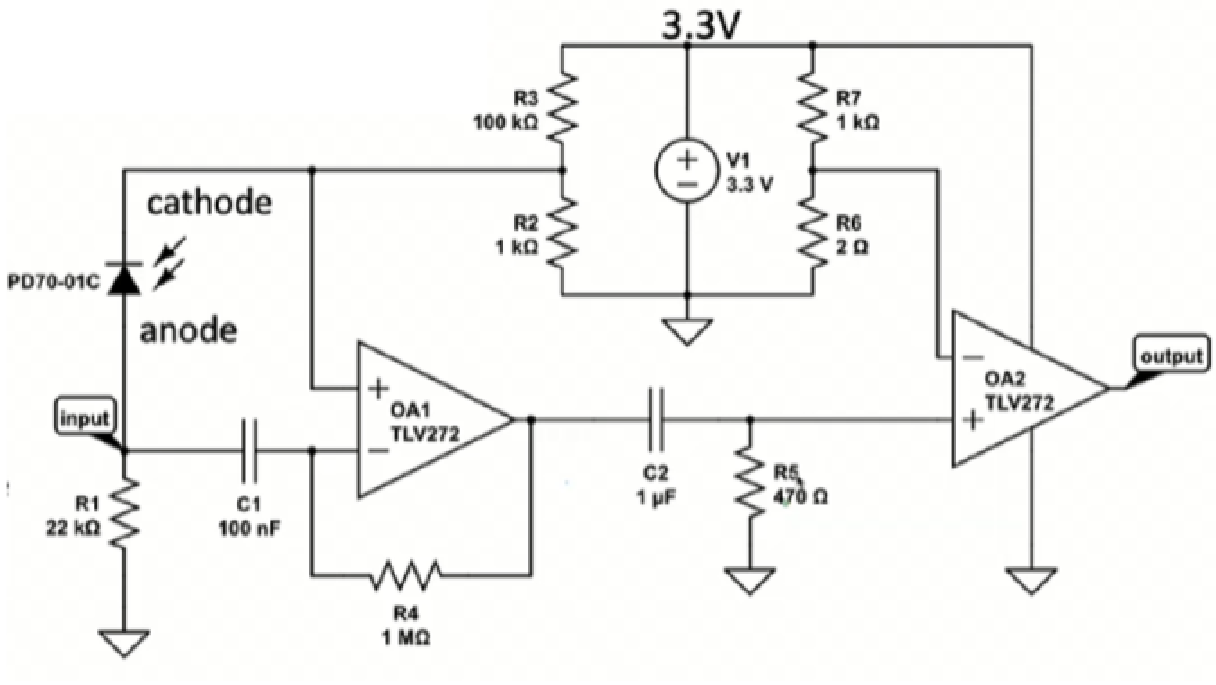
Item #	**Name**	**Quantity**	**Description**	**Cost**
1	TT DC 6V motors	4	Providing the rotational motion to the wheels	
2	Mecanum wheels	4	Special wheels that allow unidirectional motion	
3	Chassis	1	Mechanical structure of the robot to support wheels and motors	$39(The bundle includes chassis, wheels and motors)
5	9v Alkaline battery	1	Power source	$5
6	USB Portable battery	1	Portable battery to Power Esp32	$8
7	Adafruit VL53L0X Time of Flight Distance Sensor	2	Ranging sensor(30 to 1000mm)	2*$15=$30
8	Adafruit TB6612 1.2A DC/Stepper Motor Driver Breakout Board	2	Bidirectional control of four motors	2*$7.5 = $15
9	Photodiode	1	Specifically used for vive circuit	
10	Phototransistor	1	Sensor used in beacon detection circuit	
			Total Cost	**$97**

Schematics

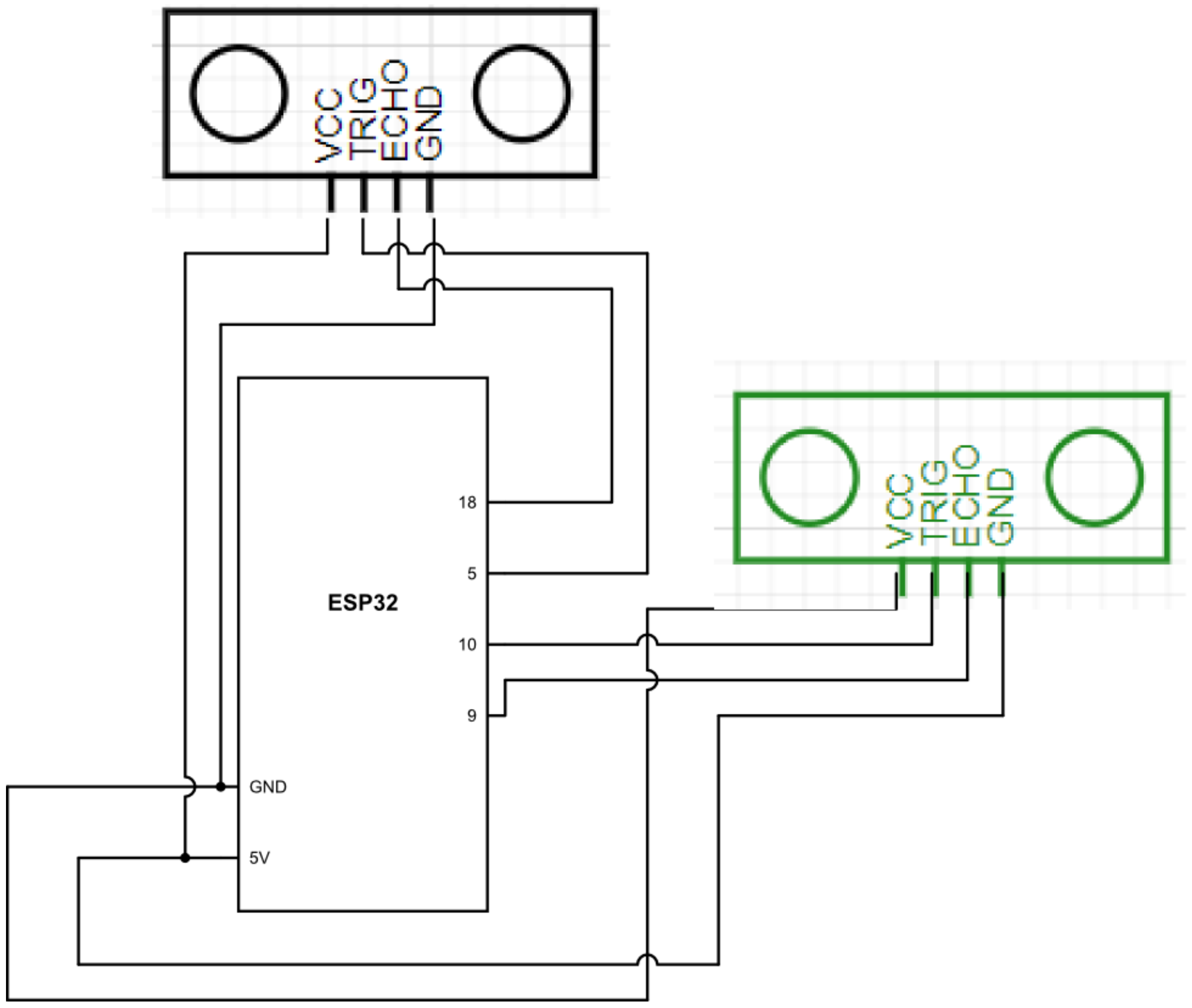
Beacon Detection Circuit



Vive Detection Circuit



Wall Following Circuit



Video Links

Wall following:

https://www.youtube.com/watch?time_continue=87&v=QeZ9K1XqW-I&feature=emb_title&ab_channel=JayadevReddy

Manual Control:

https://www.youtube.com/watch?v=3CDc7eETFe4&ab_channel=SS

The videos of other functionality was not available.