Autonomous Object Moving Vehicle

ECE 445 Design Document -- FALL 2018

Team 19: Kefan Tu, Kewei Sui

Team 32: Chenliang Li, Honglu He, Siping Meng

TA: Amr Martini

1 Introduction

1.1 Objective

Nowadays, we utilize many domestic robots to help our life become easier. Robot cleaner, kitchen robot and even robot for entertaining cat gradually appear in our houses. However, those robots are specified for certain problems and there is a lack of ability to deal with problems in various situations. There are existing techniques [1] for specific indoor tasks, but people believe that various combined algorithms can eventually be made reliable enough to create a general-purpose domestic robot [2]. One generalized task for an indoor robot is shifting objects from the start to the target destination offline without hitting any obstacles. Here are two real scenarios: 1. Stan wants to feed his cat when he is at school but he forgets to put the cat food can in the chat room. 2. Gwen wants to put her succulents on the balcony for just one hour because they are vulnerable to direct sunlight. However, she is on her business trip and she lives alone. In both of these situations, an agent is required to shift the object from one position to somewhere else.

Thus, we decided to create a small automated robot which could grab and shift objects after receiving the command given by the user. Our robot will have a robot arm to pick objects and the lower part is a chassis vehicle for moving. A camera and a LiDAR sensor will be placed in the front of the robot to help it detect objects and obstacles. Various algorithms will be implemented in the microcontroller to give this robot capabilities to find the moving path automatically and to place the object in the right place under an acceptable tolerance. Besides, our robot is designed to accomplish the task offline and automatically so that user don’t need to wait for the robot to finish its job.

What’s more, we will try to use a new interacting method, VR, to help users visualize and produce the result they want the robot to accomplish first. We think this method could enhance users’ understanding of what will the room look like finally because users can directly “see” the final state in the virtual world. The VR user interface will be a stretch goal for our project.

1.2 Background

For a long time, the idea of robots doing chores around the house has long captured people’s imaginations [3]. However, a generalized servant robot is still a rare sight in the home today. Besides, although the smart house is able to support remote control to turn the air conditioner on or off through host’s mobile phone, there are still numerous detailed situations that are hard to accomplish by simply using programmed command and smart appliances nowadays in the market. We still need a robot to help us arrange the indoor objects and shift them for various needs. Until recently, a smart domestic robot dog, SpotMini [4], created by Boston Dynamics, shows up and demonstrates its powerful abilities to open the door and shift a cup of water just like our project goal. The task’s complexity which SpotMini can do is nearly reach the hardness level of human daily tasks.

However, there is still a big deficiency for SpotMini. The size of SpotMini is too big to recognize relative small obstacles underneath. It still has a high possibility to collide with obstacles (like wall or furniture) when it turns around and even falls down in a real indoor environment. Thus, we limit the size of our robot and try to increase the movement precision as much as possible in order not to deal damage to the objects in the house because some of them are pretty fragile.

1.3 High-level Requirements

- The robot can go to the location specified by the user (either directed given from PC or VR)
- The robot can recognize the object user specified.
- The robot can grip, hold, and place the object.

1.4 Experiment Settings

The experiment area is a 4m×4m flat tile surface. We will divide it into 100 grids and thus the unit length of the coordinates is 0.4m. Here is an example of the experimental setting.

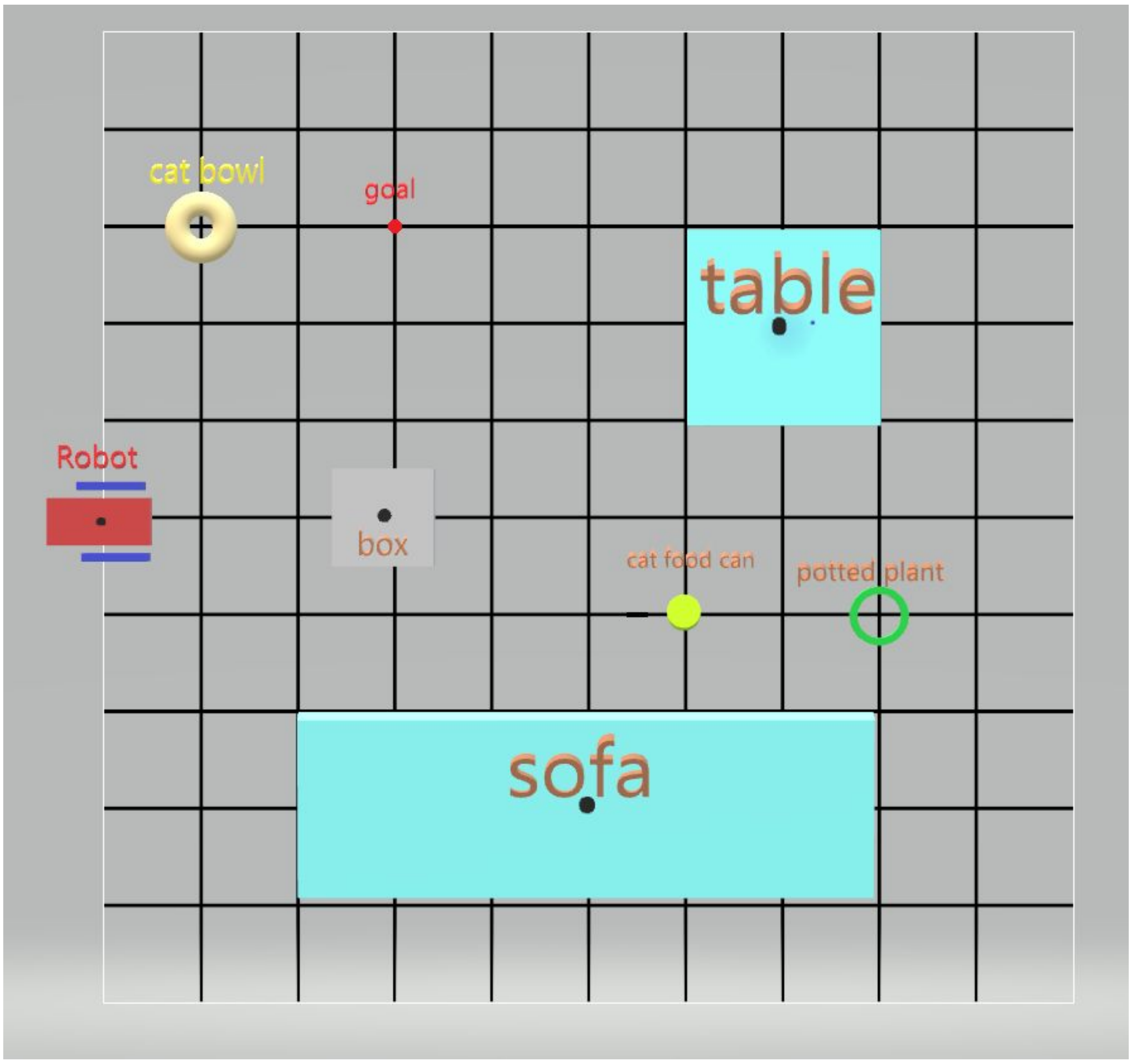


Figure 1. Experiment setting map (above)

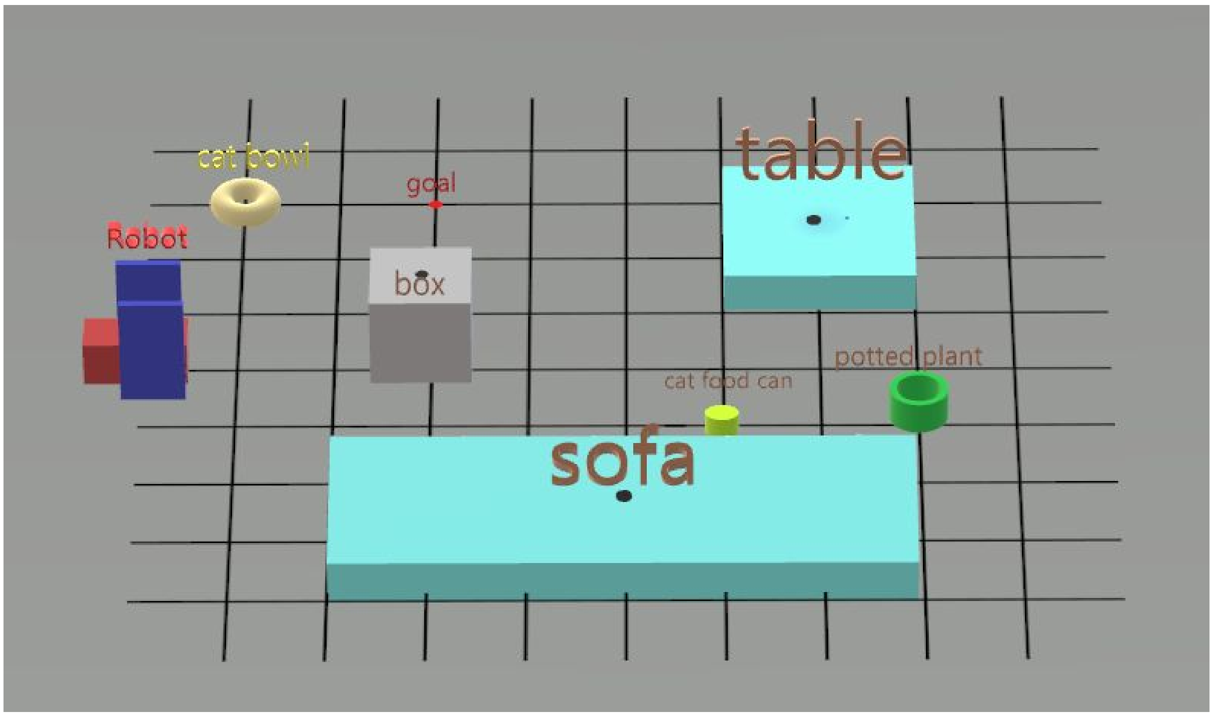


Figure 2. Experiment setting map (aside)

- "robot" represents the robot vehicle and it always starts at the initial position.
- "goal" represents the goal position for the target object which should be moved.
- "cat food can" and "potted plant" represent the target object (like cat food can in real life) with high-saturated color wristbands around it. A color wristband is used to provide a distinct feature for each object as well as enhance the efficiency and accuracy of the object detection algorithm. If the number of the wristband colors is smaller than the that of the objects, multiple color wristbands will be placed around the object in different combination ways. The object can be placed on any known place in the experiment area.

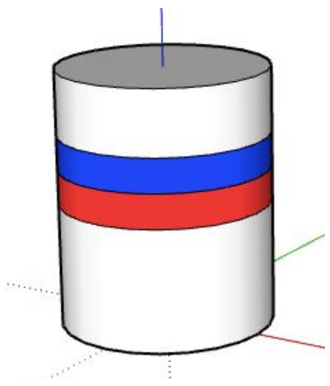


Figure 3. Example of experimental object with color wristbands

- "sofa", "table" and "box" represent obstacles with unknown size and position (like furniture or shoes on the ground in real life). The size of the obstacles has to be large enough to be detected by the LiDAR sensor. A new obstacle can be placed anywhere in the experiment area during the experiment and all obstacles are movable. It's used for testing the path planning and obstacle detection function. We are going to use artificial rectangular-shaped obstacles for the demo in order to mitigate the complexity of the experimental environment.

1.5 Solution Description

In this system, we will first provide a 2-D map in the computer to visualize the current room size with a starting point. The user will also see movable objects on the screen and a move button. All of these data will be stored and transmitted to an autonomous robot through Wi-Fi. Moreover, this robot can automatically find the object and search a path to move it to the target position with the correct orientation. In this case, the user will never be worried about the procedure of completing the task and they can even save this task into the database for the next time.

This project can be separated into two big parts, the computer-based frontend, and the autonomous robot. For the computer part, we need to create a simulated 2-D map that correctly reflects current room size, a starting point, and movable objects. For the autonomous robot part, we need to create our own robot based on a pre-built robot vehicle. The robot should have the ability to move objects and find path automatically. We will use a LiDAR, which is more accurate than an ultrasonic sensor, a gyroscope, and a camera(for object recognition) to achieve this goal and we will use a microcontroller (Raspberry Pi) and ROS system to control the vehicle. By using these sensors and localization, mapping and path planning algorithms, this robot will be able to avoid obstacles.

2 Design

2.1 Block Diagram

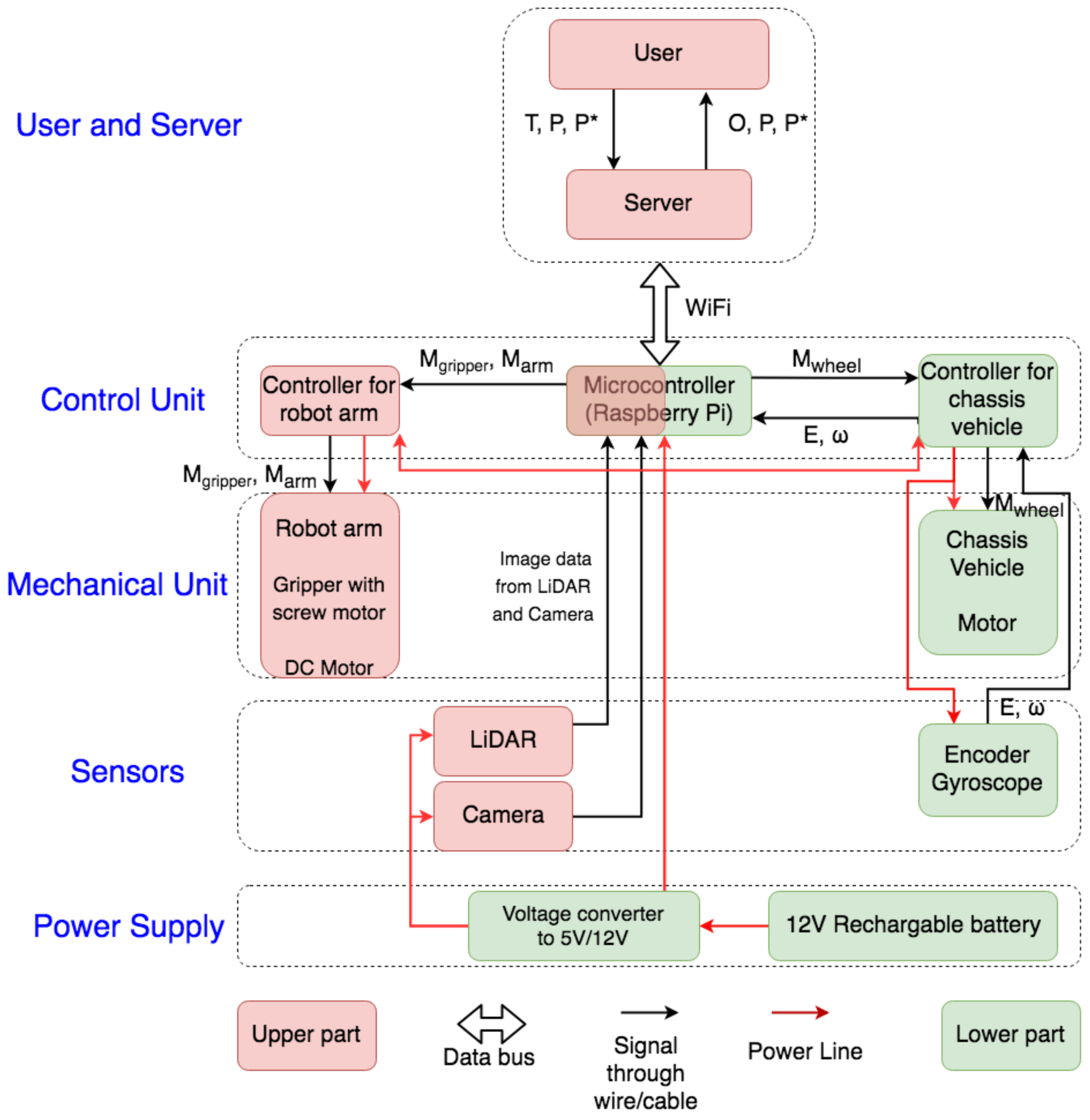


Figure 4. Block Diagram

T	Target object	**P**	Old position	**P***	New position
O	Obstacles	**E**	Data from encoder	**ω**	Data from gyroscope
M_{wheel}	Signal to control wheel motor	**M_{arm}**	Signal to control robot arm motor	**$M_{gripper}$**	Signal to control gripper screw motor

The block diagram is consist of five main units: user and server, control unit, mechanical unit, sensors, and power supply. They are balanced separated into two parts and each group of our big team is in charge of one part. User and server unit is the block which allows the user to give commands to the robot and obtain the new-generated 2D map from the microcontroller and then show it to the user. Control unit is the brain of the robot which contains one microcontroller (Raspberry Pi) and two control circuits (two PCBs for both upper and lower parts). Then, a Mechanical unit is the physical structure unit for the entire robot. The robot arm is placed on the upper part and the lower part is mainly a chassis vehicle. What's more, various sensors are used in this project in order to implement multiple algorithms and accomplish the task. Finally, the power supply unit provides the power with different voltages to the entire robot.

The group in charge of the upper part of the robot majorly contributes the user and server unit and the blocks arranged on the upper part of the robot in control, mechanical and sensors unit. Likewise, except dealing with the blocks placed at the bottom, the lower part group also handles the power supply unit. Both groups will work on the microcontroller together because it is the most crucial component of the entire system.

2.2 Physical Design

The physical design of this robot is chosen to allow flexible moving with a relatively small size and to place all sensors in appropriate positions to work. Our robot basically consists of a two-layer chassis vehicle with two wheels. The dimensions of the chassis vehicle are 250mm x 205mm x 103mm. We choose only two wheels rather than four in attempts to reduce the cost and to allow the robot to do pivot turns more easily. The encoders will be placed on the motors in order to calculate distance traveled by detecting the speed of wheels.

A robotic arm placed on the top of the chassis (on the second layer) will be actuated by two motors to grip items. One big DC motor is used to lift the arm and the other screw motor is used to open and close the gripper. The physical design of the arm will be elaborated in the robotic arm block. On the second layer, we will place a LiDAR in the front of the vehicle to detect obstacles when the robot is moving. However, the arm and camera will be put on the back, because we don't want other physical components to be detected by LiDAR otherwise the obstacle detection will be undermined. When the robot arrives its destination and is going to pick up an item, it needs to turn around first so that its robotic arm and camera will face to the item. We will put the remaining hardware components on the first layer of chassis, including the microcontroller, PCBs and a gyroscope.

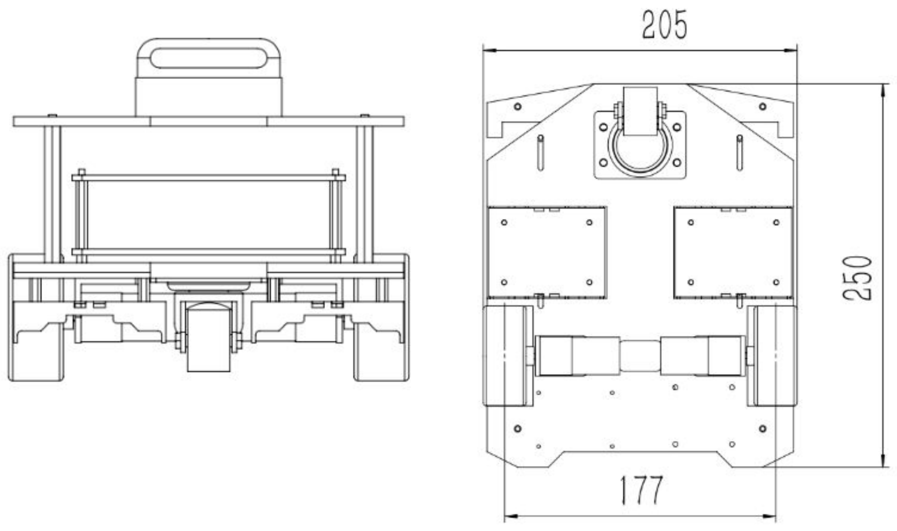


Figure 5. Physical overview of chassis vehicle (unit: mm) [5]

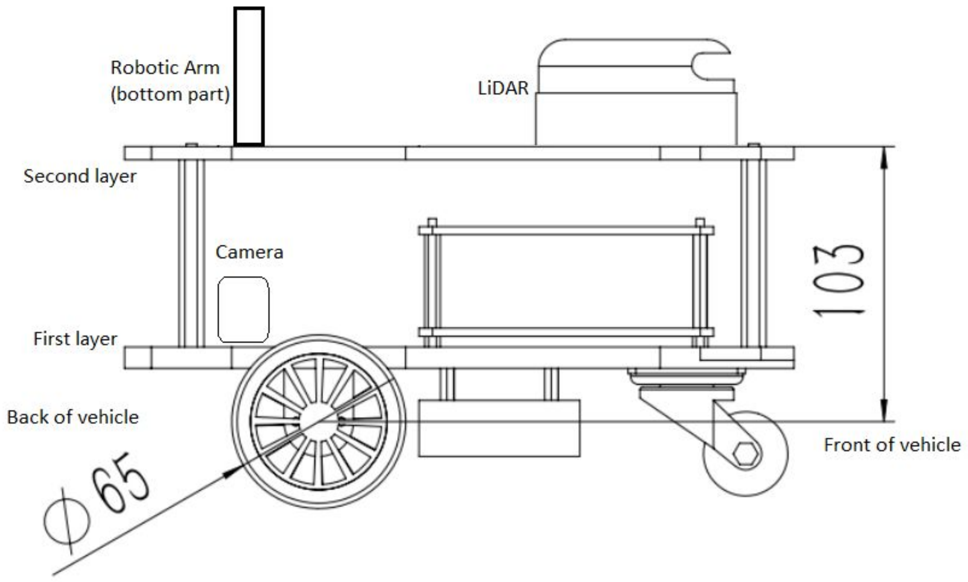


Figure 6. Relative positions of arm, LiDAR, and camera (unit: mm) [5]

2.3 Design Document 1 (Upper)

2.3.1 Robotic Arm

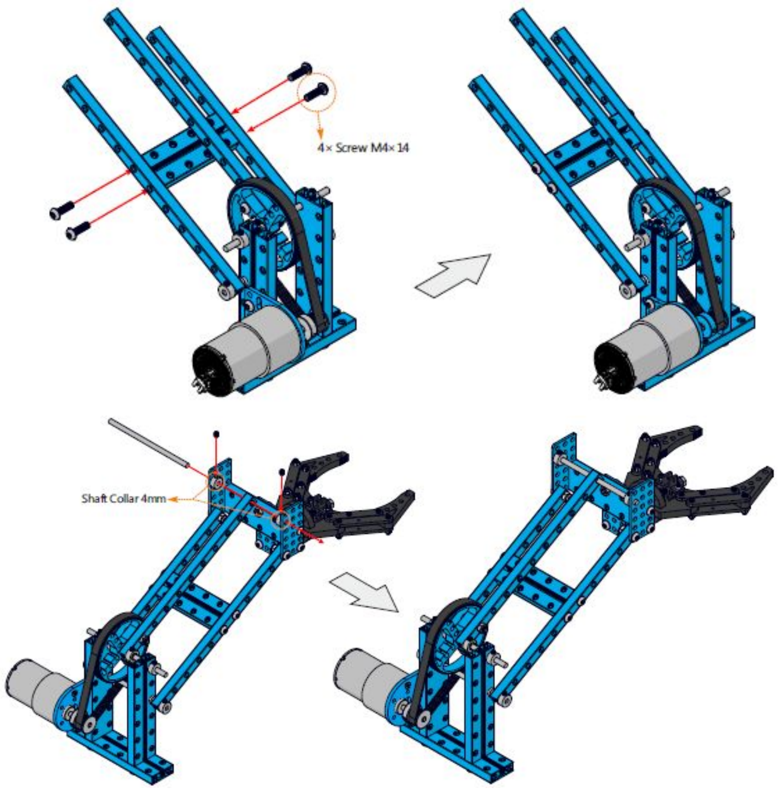


Figure 7. Robotic Arm Overview [6]

The physical design of the robotic arm is divided into two parts: the upper part has a gripper at the end of the beam while the other end is connected by a rotatable joint with the lower part, which is fixed to the chassis vehicle. The robotic arm needs to adjust the height of gripper. This is achieved by using a timing belt pulley that is driven by a motor; the pulley rotates with the upper beam so that we can adjust the height of the gripper.

2.3.1.1 Robot Gripper

We will put a gripper at the very front of our robotic arm to grip items. The gripper is controlled by an N20 screw motor. When the screw spins, the gripper will be pushed to open or pulled to close due to the mechanical design. The operating voltage range of the gripper is 5-12V DC.

Gripper (with motor)	
Requirement	Verifications
Able to provide enough torque to grasp a can, whose diameter is usually 65mm and height is usually about 125mm. If the can is full of water, the weight is about 360mg.	A. Run arm control program B. Ensure the distance between two ends of the gripper is at least 65mm C. We will keep increase the pressure exert on the can until we can ensure the can is

	grasped and lifted without slippery and then record the data every time during the test in order to get the best result.

2.3.1.2 DC Motors

We will use a 37mm DC motor to control the timing belt pulley in order to adjust the height of gripper and use an N20 screw motor to control the operation of the gripper. The operating voltage range of both motors is 5-12V DC. They will interface with the motor controller circuit to be able to run in opposite directions.

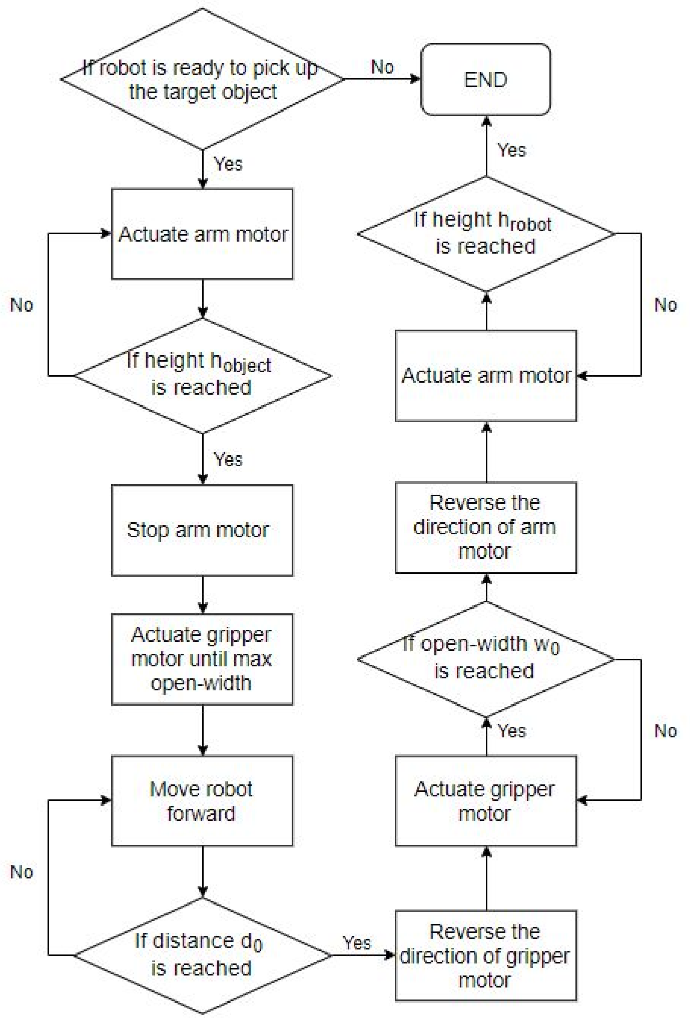
DC Motor	
Requirement	Verifications
Able to provide enough torque to lift the robotic arm	A. Give 12V input to the motor B. Ensure that timing belt pulley rotates as motor rotates and that the angle between the upper part and lower part of the arm is adjustable

2.3.1.3 Arm Control Algorithm

This algorithm will be called after robot is adjusted to a proper location to pick up or place the target item based on its calibration algorithm. The predetermined values, including height h_{object} and h_{robot}, distance d_0 and d_1, and open-width w_0, need to be determined based on experiments before we run this program.

To grip the item the robot just confirms, the pseudocode is the following:

```
If robot is ready to pick up the target object:
        Actuate arm DC motor to lower the height of gripper
        Stop it if predetermined height $h_{object}$ is reached
        Actuate screw motor to open the gripper to its maximum width
        Move the robot forward by predetermined distance $d_0$ so that the item is in the gripper
        Reverse the direction of screw motor and actuate it
        Stop it if predetermined open-width $w_0$ is reached
        Reverse the direction of arm DC motor and actuate it
        Stop it if predetermined height $h_{robot}$ is reached
```



If robot is ready to pick up the target object
No
END
Yes
Actuate arm motor
No
If height h_{object} is reached
Yes
Stop arm motor
Actuate gripper motor until max open-width
Move robot forward
No
If distance d_0 is reached
Yes
Reverse the direction of gripper motor
Actuate gripper motor
Yes
If open-width w_0 is reached
No
Reverse the direction of arm motor
Actuate arm motor
If height h_{robot} is reached
No
Yes

Figure 8. Flowchart 1 of arm control algorithm

To place an item on the floor, the pseudocode is very similar to the pickup one:

```
If robot is ready to place the target object:
        Actuate arm DC motor to lower the height of gripper
        Stop it if predetermined height $h_{object}$ is reached
        Actuate screw motor to open the gripper to its maximum width
        Move the robot backward by predetermined distance $d_1$ so that the item is out of gripper
        Reverse the direction of arm DC motor and actuate it
        Stop it if predetermined height $h_{robot}$ is reached
```

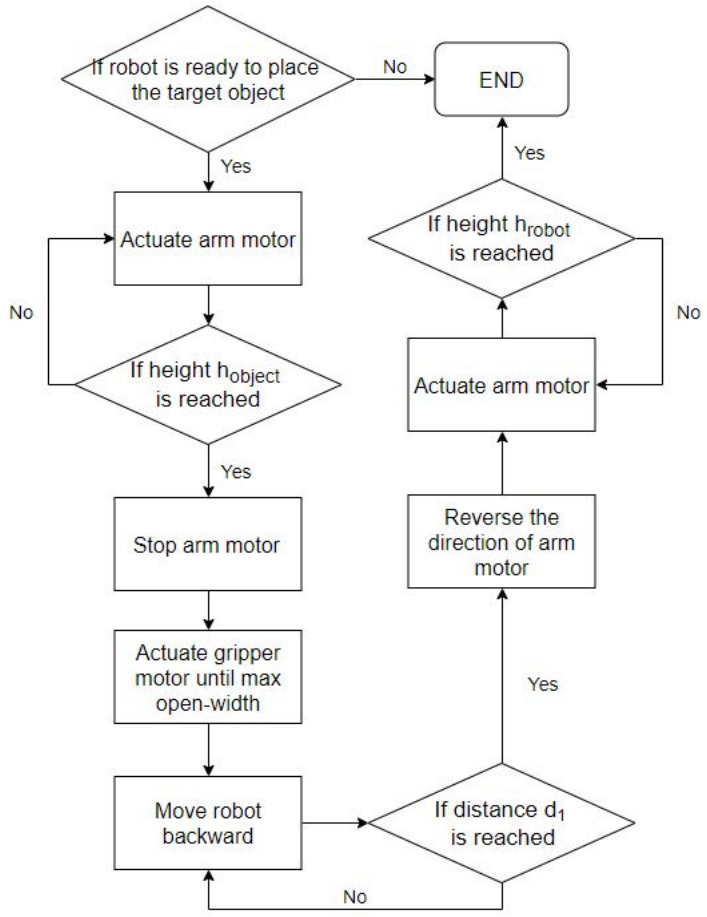

If robot is ready to place the target object
No
END
Yes
Actuate arm motor
No
If height h_{object} is reached
Yes
Stop arm motor
Actuate gripper motor until max open-width
Move robot backward
If distance d_1 is reached
No
Yes
Reverse the direction of arm motor
Actuate arm motor
If height h_{robot} is reached
No
Yes

Figure 9. Flowchart 2 of arm control algorithm

2.3.2 PCB design

2.3.2.1 Motor Controller

A motor driver will be implemented in our PCB as a control circuit to drive motors. Basically, we use an H-bridge design to drive the motors, using two pairs of transistors to control the direction of current and thus the direction of motor rotation. The controller will receive signals from the microcontroller and control the current direction correspondingly. We will use the chip TB6612FNG, which is a high-efficiency MOSFET driver with low heat dissipation.

Motor Controller	
Requirement	Verifications
Able to change the direction of motor running given input signal from microcontroller	A. Give proper signals to H-bridge to change the current direction B. Ensure the direction of motor is changed corresponding to signals from microcontroller

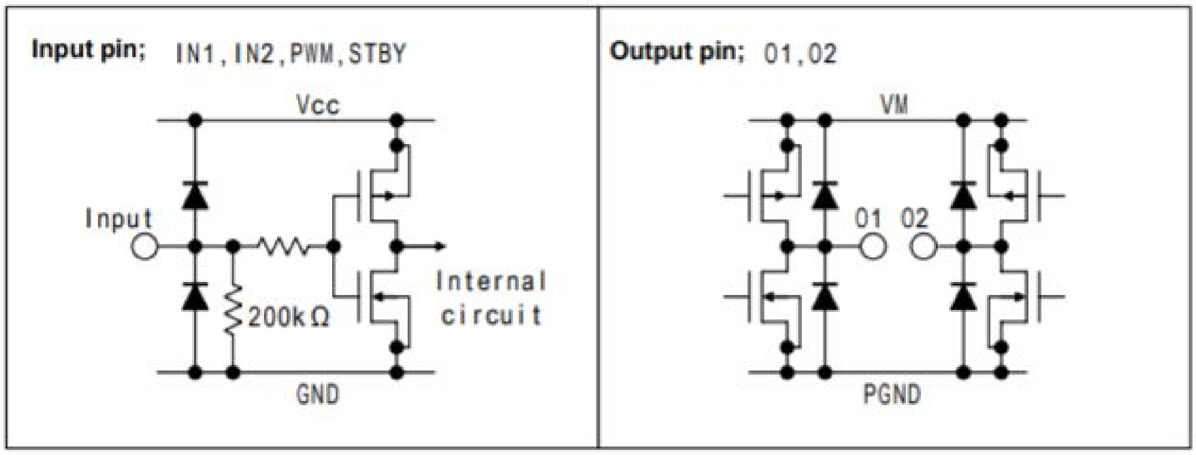
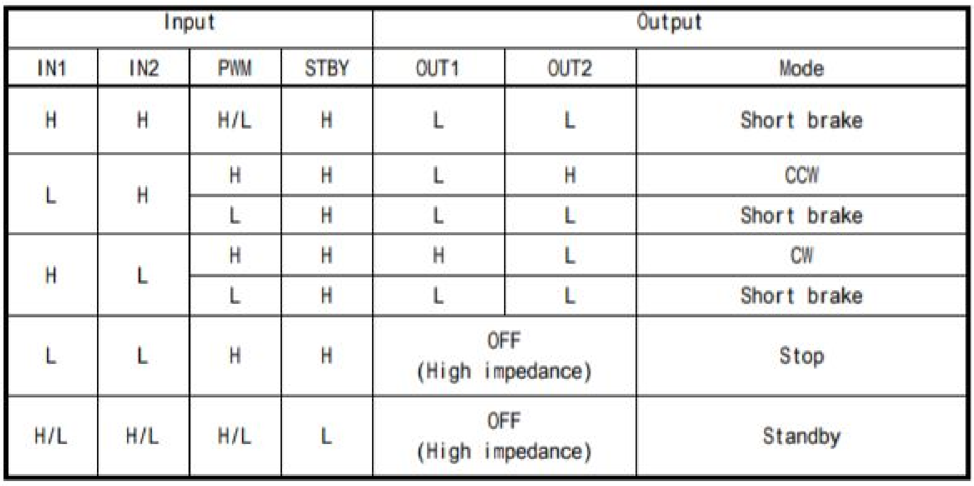


Figure 10. Schematics of H-bridge circuit [7]



Input				Output		
IN1	IN2	PWM	STBY	OUT1	OUT2	Mode
H	H	H/L	H	L	L	Short brake
L	H	H	H	L	H	CCW
		L	H	L	L	Short brake
H	L	H	H	H	L	CW
		L	H	L	L	Short brake
L	L	H	H	OFF (High impedance)		Stop
H/L	H/L	H/L	L	OFF (High impedance)		Standby

Figure 11. Control Function Table [7]

2.3.2.2 Circuit Schematic

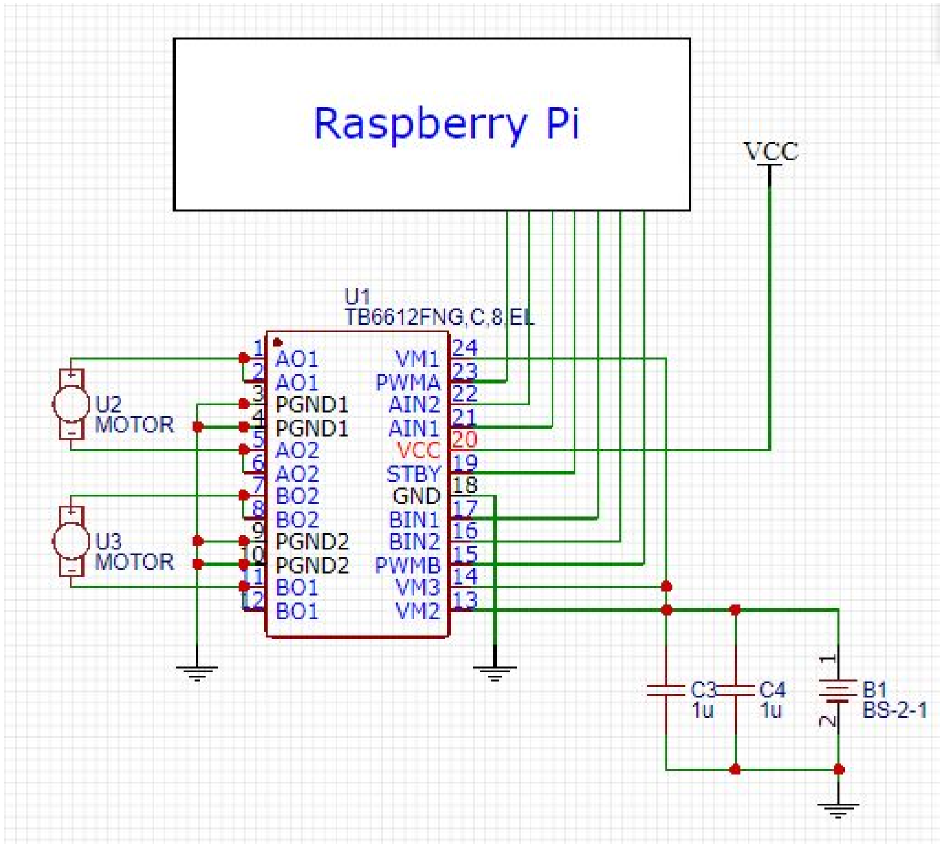


Figure 12. Circuit Schematic

2.3.3 User

User (or Client) is the front-end for the entire system. By directly inputting the target object (T) and the new position for that object (P*) into this front-end, the command will be generated and transmitted to the server and finally reach the microcontroller on the robot. We will create a visualized 2D map on the computer in order to visualize the positions of the robot, objects and detected obstacles and error message sent by the robot. User is able to use keyboard to input T and P*. What's more, stop command will be implement in order to stop the robot immediately to prevent it from hurting others.

User	
Requirement	Verifications
1. An efficient user interface should be designed to show positions of the robot, objects and detected obstacles and error message as well as	1. A. Print out the received data from the Raspberry Pi and make sure they are exactly the

transmit data to the server and microcontroller. 2. Stop command can successfully stop the entire robot in less than 3 seconds.	same with the input from the user B. Print out the positions of the robot, objects and detected obstacles and any error message and make sure they are exactly the same with the positions calculate from the Raspberry Pi 2. After send stop signal, the robot should shut down and stop moving immediately.

2.3.4 Server: Connection between Microcontroller and User

Since we will connect the Raspberry Pi and our computer/VR device wirelessly, a server-user architecture will satisfy our needs. This is because of the Raspberry Pi, holding a lot of information including location data, LiDAR data and error messages, need to be shared with a user so that the user could visualize these data on a 2-D map:

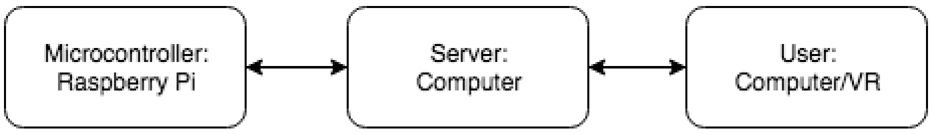


Figure 13. Server, Microcontroller and User relationship

Meanwhile, the user could also send a service request, such as feeding the cat and moving plants, to the server and then from server to the microcontroller. We will use Python <socket> package to implement this feature. More specifically, on the server, we will open a socket and listen for incoming connections from user and microcontroller and build up a database to log the data. On the user, we will connect to the socket on the server and record the incoming information that arrives. On the microcontroller, we will keep writing data into the socket and send it to the server every second.

Server-User architecture	
Requirement	Verification
Successfully connect microcontroller, server and computer	Print out everything in the terminal and ensure data has been transmitted successfully

2.3.5 Sensors

2.3.5.1 Camera

The Camera we used is the Raspberry Pi 8-megapixel camera module V2 which will be directly connected to the microcontroller as data receiver and power supplier. We will use this camera to detect the object from the sampled pictures by utilizing an object detection algorithm to find the mean point of the object on the image. Then our microcontroller will calculate the relative position based on the location of the object shown in the image. By correcting its direction toward the target object, the robot is able to move close enough to the target object and pick it up finally. The horizontal Field of View (FoV) is 62.2 degrees and the vertical FoV is 48.8 degrees which are sufficient for our project [8]. The best stream mode is 1080p with 30fps [8] but we only need shoot five to ten 480p pictures per second because high-resolution pictures are not necessary for just finding the relative position the object and we need to cut down the memory usage of this task. We will discuss more the object detection algorithm in 2.3.6.2.1.

Camera	
Requirement	Verification
Camera must shoot at least five 640x480 RGB pictures per second	Write a program to output the png file of the captured pics and then manually check the created time of those files to ensure at least five pics are created in a second

2.3.5.2 LiDAR

The model we use is the SLAMTEC RPLIDAR A2m8. The frequency of the LiDAR is set to 10 Hz, and 400 data points for a 360-degree scan. We'll use as many data points as possible, depending on the dimension of the robot arm.

LiDAR	
Requirement	Verifications

On the microprocessor will process the LiDAR data and output the obstacle location to the map. The map will show at least 90% of the obstacle that the current robot can detect	A. Put the robot at home position, and turn the robot on but without running. Wait for 5 seconds and then check the map generated by the robot B. Compare the obstacles location with the real ones, and people stay at home position and count if those obstacles can be seen by human eye

2.3.6 Control Unit

2.3.6.1 Microcontroller (Introduce Raspberry Pi)

Raspberry Pi 3B+ will receive data from all sensors, include LiDAR (USB), Camera (Pi module), Encoder (GPIO), Gyroscope (ADC to GPIO). It will send data to robot arm (GPIO/PWM), motors (wheel) (GPIO). It will also send location data to the user's laptop through TCP/IP communication.

2.3.6.2 Algorithms

2.3.6.2.1 Object Detection

We will use OpenCV to help us implement this algorithm. Each object will have a color wristband on it, and after the robot reaches the approximate location, the camera will play a role as a feedback loop and keep running this algorithm to correctly making sure the robot is directly facing toward the object.

The light intensity might fluctuate widely in a room because of the artificial light. Thus we need to apply gamma correction [9] to the image in order to get rid of the problem of overexposure and underexposure.

We also need to change the color model of the picture from the RGB model to the HSV model because we need it to apply the color mask in OpenCV [10]. Color masks we will use are actually ranges of HSV color. Those ranges will be defined and saved in the code first. This algorithm will check all pixels and if the color of one pixel is located in one range, then this pixel will be labeled as the color which this range represented. By checking the vertical order of colors in the image with the map which stores all orders as keys and the relative object as values, the robot could understand what object it is.

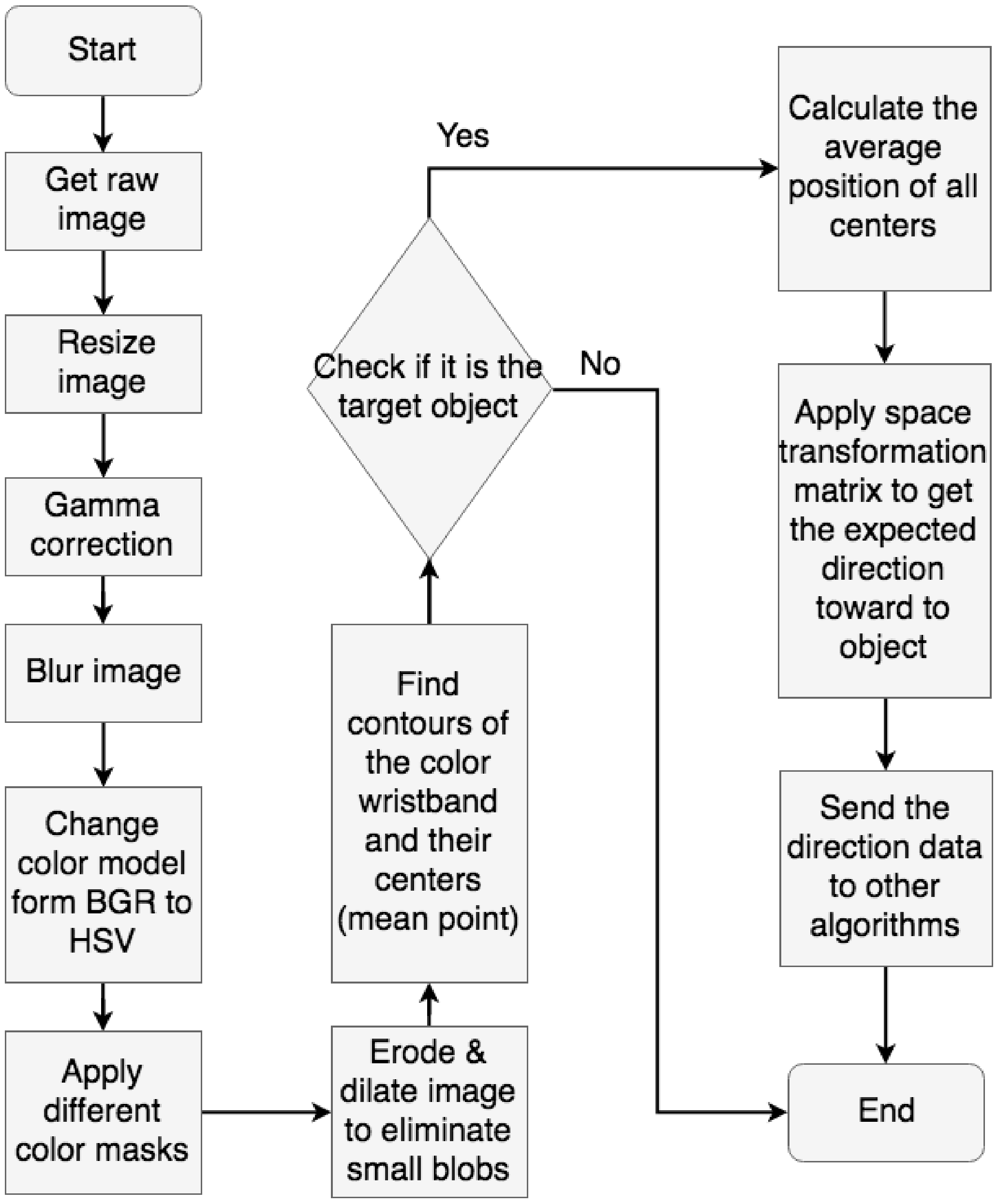

Start
Get raw image
Resize image
Gamma correction
Blur image
Change color model form BGR to HSV
Apply different color masks
Erode & dilate image to eliminate small blobs
Find contours of the color wristband and their centers (mean point)
Check if it is the target object
Yes
No
Calculate the average position of all centers
Apply space transformation matrix to get the expected direction toward to object
Send the direction data to other algorithms
End

Figure 14. The flowchart of object detection algorithm

Object Detection	
Requirement	Verification
1. The detection range should be at least 0.5m. 2. The robot can adjust it self facing object, with an angle of +/- 5°	1. A. The robot will run this algorithm and detect the object once it get close to the target position. The minimum required detection range should be at least 0.5

	meters which is slightly bigger than the unit length of the coordinates. B. We will test the performance of the object detection in the situation that the distance between the object and the camera is larger than 0.5m and adjust algorithm for eliminating small blobs. 2. A. We put the robot 0.5 meters away from object and turn on camera detection, and then check the final angle between the robot and the object. B. If the final angle is larger than +/- 5°, we could smaller the tolerance about the center of the detected object and the center of the whole image in order to move the image of the object closer to the center.

2.3.6.2.2 Localization Calibration

We'll calibrate the robot position by letting it go home. Upon the completion of every task, the robot will go home to calibrate itself. The robot will push itself into the space between two walls. The distance between two walls is slightly longer than the width of the robot. At this time, the rotation angle, θ, will be calibrated by resetting to 0°. We'll use object detection to adjust angle of the robot in order to let it move straightly into the space between walls. We'll also use the LiDAR data to get the distance to the left/right side wall in order to calibrate the position.

Calibration	
Requirements	Verifications
1. User inputs a calibration signal, and then the robot should go home immediately or after finishing up current task	1. At any position in the map, we give the robot a calibration signal, the the robot will start to go home in one second.

2. After it gets home, the calibrated location and angle should be within 1% difference from the actual location and angle compared to the unit length and angle of the coordinates.	2. A. We will check the location after calibration stored by robot and compare it with the actual location during the test. B. We will use object detection algorithm and the data from LiDAR at the same time in order to make sure the robot will successful move into the space. The object detection algorithm will rotate the robot to face straightly to the space and the data from LiDAR will calculate the position of the robot relative to two walls and keep the robot on the center line of the space.

2.3.6.2.3 Obstacle Avoidance

The LiDAR will pass in a raw value (distance in each data point) to the microprocessor, and we'll process them into world frame position and mark the grid near them an obstacle. Once an obstacle position is determined, it can't be cleared unless the robot re-start with a fresh new map. A* algorithm will avoid any obstacle.

Obstacle Avoidance	
Requirement	Verification
While robot is moving, it avoids collision by stopping moving toward obstacles or just moving along them but does not collide with an obstacle or get stuck	A. Starting at home position, input a object location and let the robot run itself B. Ensure the robot will get not stuck or collide with obstacles before successfully going to target location

2.4 Design Document 2 (Lower)

2.4.1 Mechanical Unit

2.4.1.1 Chassis

We are using an acrylic waffle board robot car chassis with the length of 250mm, the width of 200mm and the height 140mm. This car has two layers, two wheels with motors and an Omni wheel.

2.4.1.2 Motor

The DC brushed motor we'll use to drive our robot is CHR-GM25-370, DC 12V, 220 RPM. It comes with a 6-PIN Magnetic Holzer Encoder.

We choose brushed motor rather than the brushless motor because it is inexpensive, steady, and easy to control: we can simply use voltage to control the speed of the motor and we can also change the voltage direction to change the rotation direction of the motor. The maximum speed is 160-300rpm and the working voltage is 3-12V DC and the max torque is 1.8kg·cm.

Motor	
Requirements	Verifications
The motor can drive the robot with all the components on it and the speed should be no less than 4.7cm/s i.e. robot can move along the diagonal line in 2mins.	We test the time of the robot with all the components on crossing the diagonal line and check if the time is less than 2mins.

2.4.2 Sensors

2.4.2.1 Encoder

The motor comes with 6 -PIN Holzer encoder, which counts to 224.2 pulses per revolution. With rotation rate information, we can read it to the microprocessor and use the setting diameter of our wheels to calculate the linear distance that the robot has traveled.

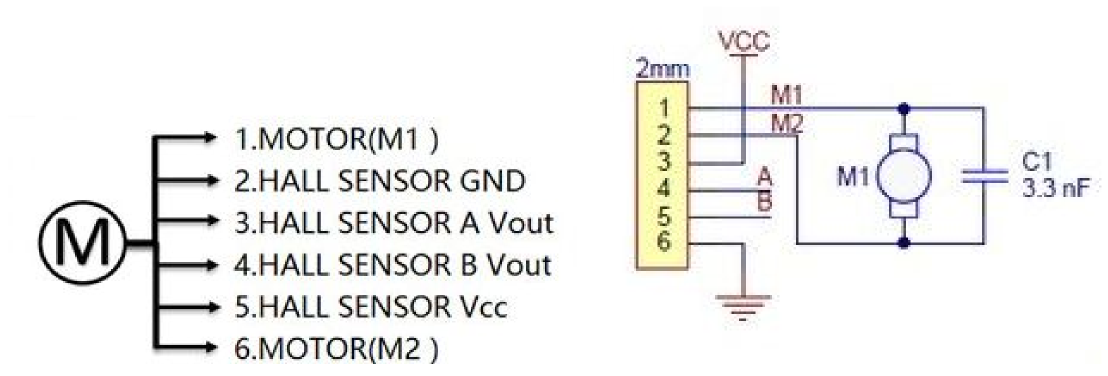


Figure 15. Encoder & motor schematic

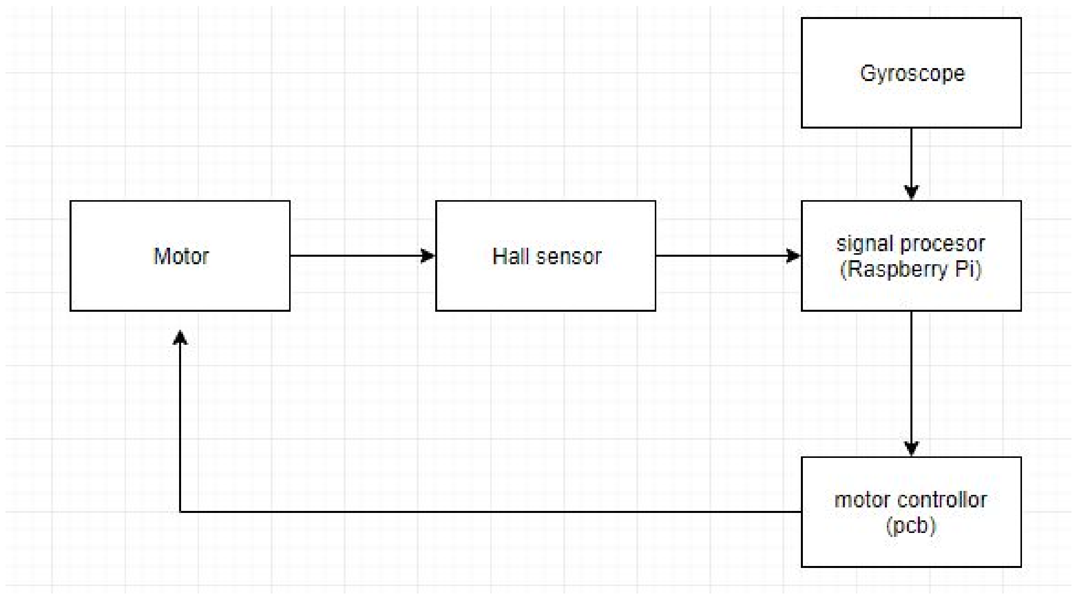


Figure 16. Motor control loop

We have two methods to implement the wheel speed, one is based on frequency and another is based on a period cycle. Since our wheel speed is relatively low, we decide to use the measuring method based on a period cycle.

$$\mathrm{n} = \frac{60f}{p \times m}$$

In the formula above, n is the rad/s, f is the base frequency we produce by the controller, p is the pulse number in one rotation that the encoder produces. And m is the number of the base pulse in one rotation.

As long as we get the rotation rate we can calculate the distance we traveled and we can determine our moving direction from the information we get from the gyroscope.

Since we can control our wheels independently, we can change the moving direction by giving different speed to different wheels.

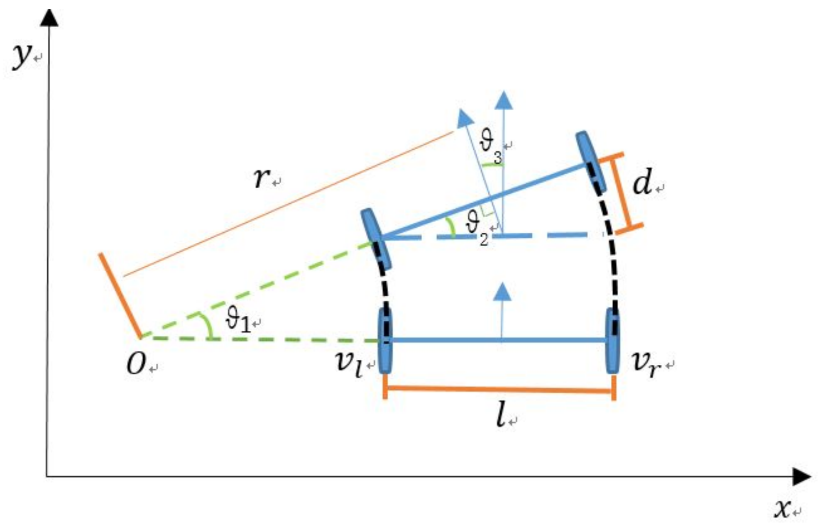


Figure 17. Example of robot moving path

We will have:

$$v = \frac{v_r + v_l}{2} \qquad \theta_3 = \theta_2 = \theta_1 \qquad w = \frac{\theta_1}{\Delta t} = \frac{v_r - v_l}{l} \qquad r = \frac{v}{w} = \frac{l \cdot (v_r + v_l)}{2\,(v_r - v_l)}$$

Where *v* is total velocity, ω is angular velocity, θ is the angle we rotate, r is the rotation radius.

From the formulas above we can calculate the current velocity, angular velocity and rotation radius by the feedback from our encoder and gyroscope, then we can regulate our wheel speed individually to control the motion of the robot.

Encoder	
Requirements	Verifications
1. The pulses generated by the encoder to the controller can be used to calculate the current angular velocity of the wheel encoder. 2. The error of the velocity calculate from the encoder information should be within 0.05m/s	1. Connect encoder to the microcontroller, it should display current average linear velocity of both wheels. 2. We manually push the robot for 1m, and integrate the velocity to check the actual difference.

2.4.2.2 Gyroscope

The single Axis gyroscope （MPU-6050) will transmit data by I2C to raspberry pi, with a RMS noise of 0.05°/s.

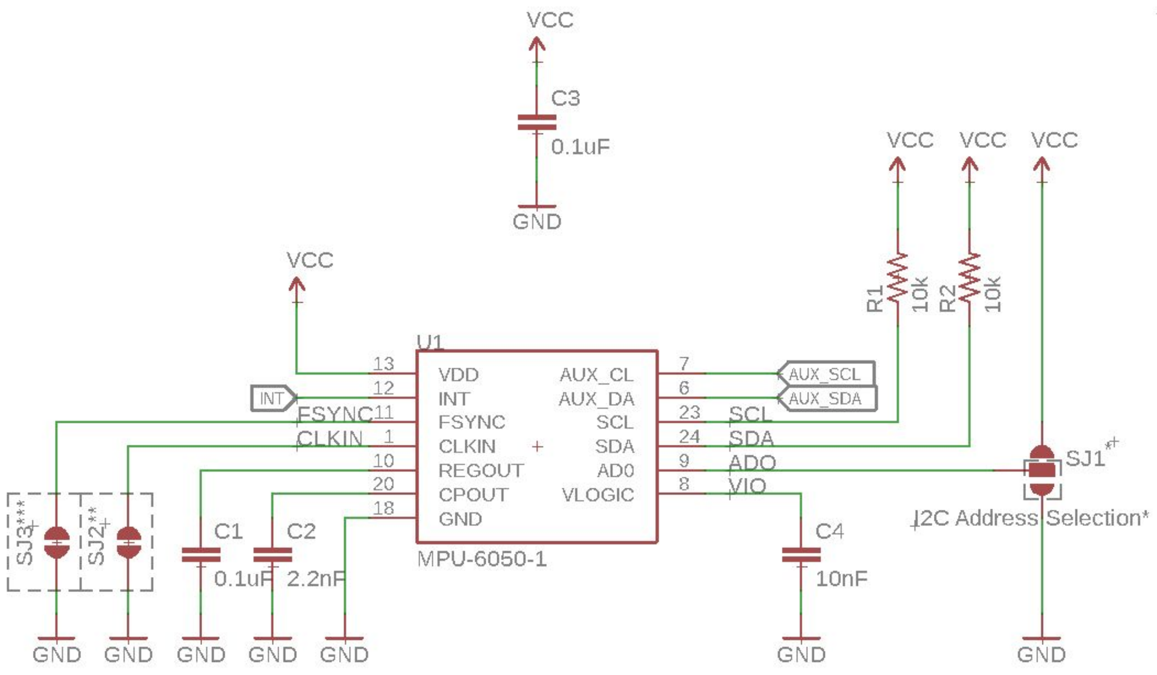


Figure 18. Gyro I2C communication connection[11]

Gyroscope	
Requirements	Verifications
1. The microprocessor will be able to read in and convert the input voltage to angular velocity in rad/s. 2. The angle report by integrating the angular velocity should be within +/- 5%.	1. Connect the gyro, raspberry pi, the raspberry pi will display the angular velocity in terminal. 2. Start with angle 0, and gradually turn the robot 360°, and then check the current angle.

2.4.3 PCB design (for motor, gyro,encoder)

2.4.3.1 Motor Controller

We are using 3-12V DC brushed motor to drive our robot. To control the DC brushed motor we need to control the voltage across the motor. So we need to build a voltage control circuit.

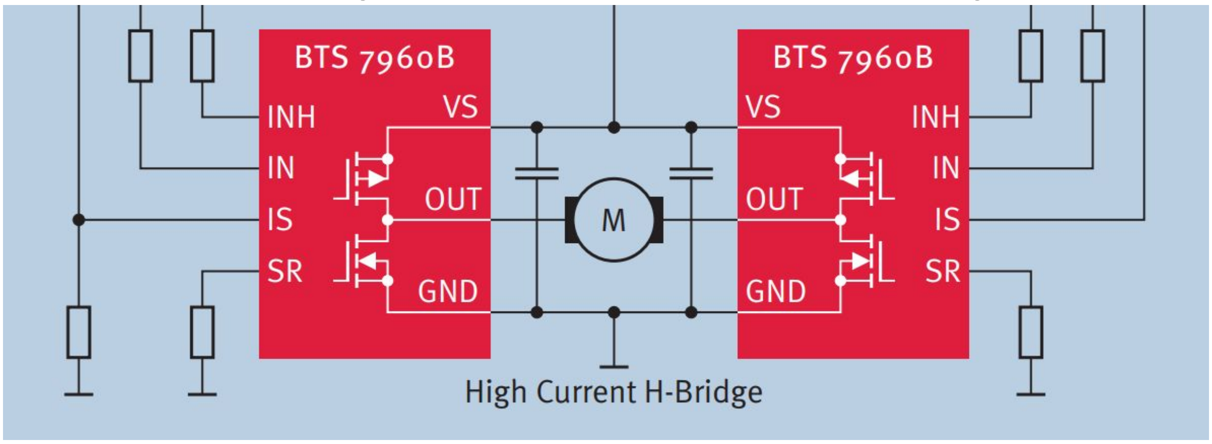


Figure 19. Circuit schematic[12]

For brushed 130-motor, the speed control mostly depends on the voltage provided to the motor, and the direction of rotation depends on the current direction.

We use two BTS7960B chips to build an H bridge to control since we need to control the current direction to make the motors run in two directions. We choose to use BTS7960B is because that chip can provide precise voltage control to the motor at the high current condition(43A).

Sensor circuit	
Requirements	Verification
1. The motor controller can receive the signal from Raspberry Pi and control the voltage and current direction across the motors. 2. All the parts on PCB can work together and the robot can make a turn by the degree we set.	1. We can use the microcontroller to send the signal to the motor controller and control the motor rotation. 2. We can compare the parameters we set and the result we get when measuring the robot's movement. The difference between our expectation and the real movement should less than 5%.

2.4.3.2 Protection Circuit

A protection circuit in the PCB to prevent our circuit from any misconnection of the power or overload.

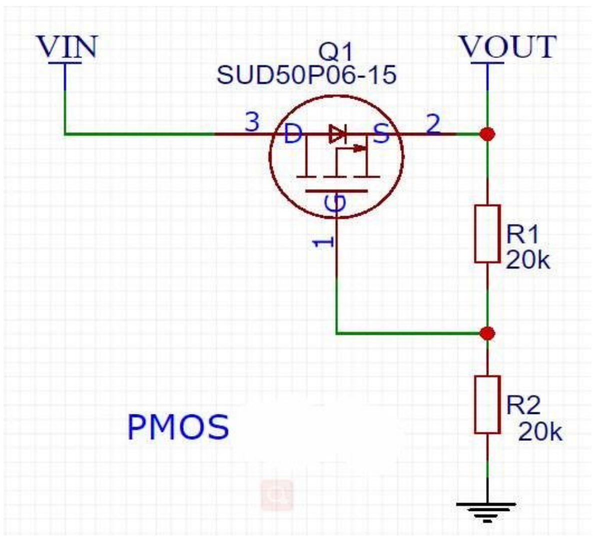


Figure 20. PMOS

The PMOS we choose may change eventually depend on the circumstance. It will restrict the direction of the current.

We will also use the resettable fuse in the circuit to constrain the current and make sure our circuit will not burn due to excessive current.

2.4.3.3 Circuit Schematic

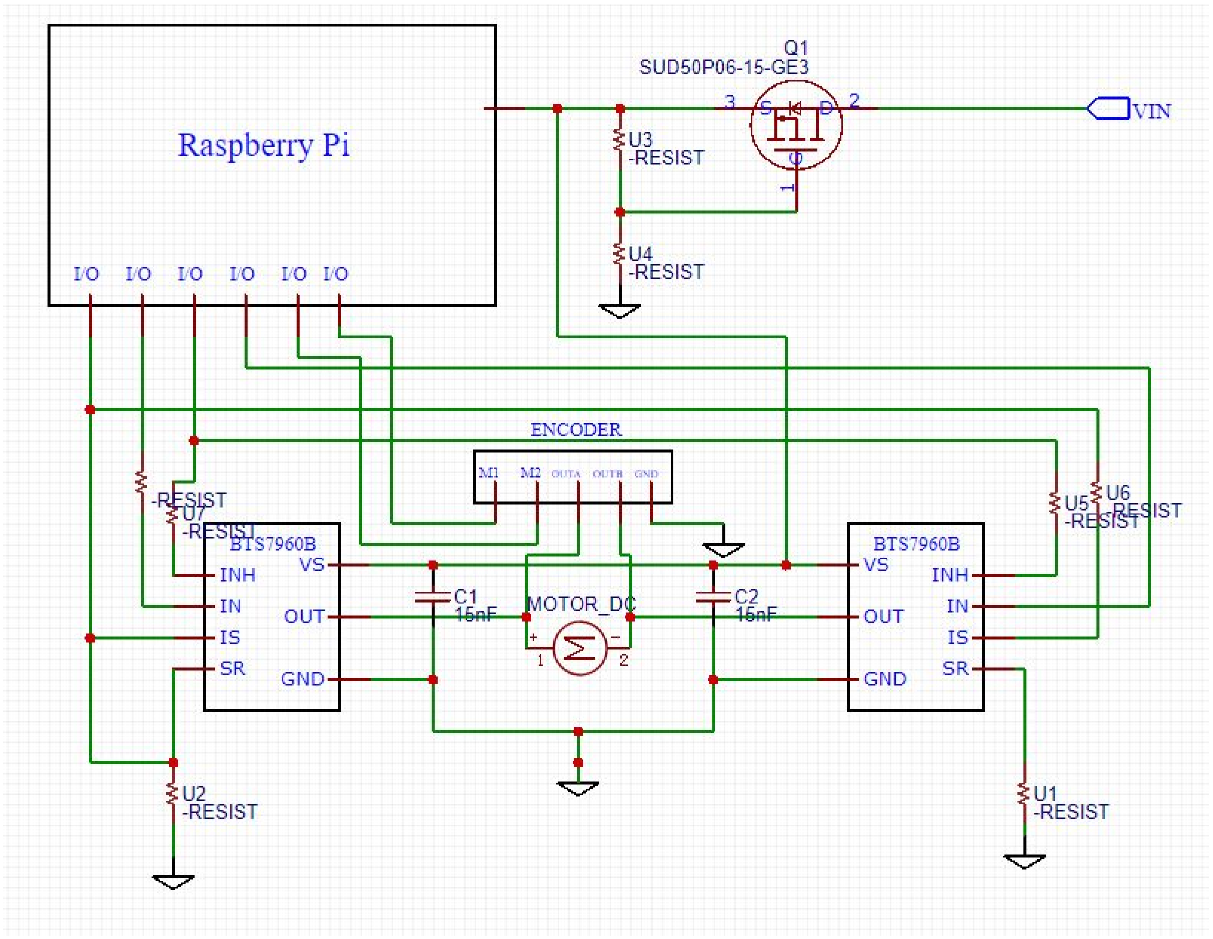


Figure 21. Circuit Schematic

2.4.4 Algorithms

2.4.4.1 Localization

Since the microprocessor will process the raw information from the gyro and both encoders, we can calculate the robot position relative to world frame as follow:

$$\begin{pmatrix} x \\ y \\ \theta \end{pmatrix} = \begin{pmatrix} \cos\theta_{prev} & 0 \\ \sin\theta_{prev} & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} v \\ w \end{pmatrix} t + \begin{pmatrix} x_{prev} \\ y_{prev} \\ \theta_{prev} \end{pmatrix}$$

Figure 22. Localization formula

Where (x,y) is the current location, (x_{prev}, y_{prev}) is the previous location at time t ago. θ and θ_{prev} are the angles the robot faces for current and previous respectively. Time t is determined by the output frequency of the gyro and the encoder, whichever is faster.

Localization	
Requirement	Verification
The robot location should be +/-0.05m from actual location	We set a object location and let the robot drive itself, check its final location and the given object location

2.4.4.2 2D Mapping

Our map will be a 10*10 gridded map, corresponding to a 4m×4m environment. All obstacles will be a 1*1 square box. With LiDAR passing length information in different angles, we can calculate its relative position and transform into world position and then check which obstacle it's close to, and update that obstacle.

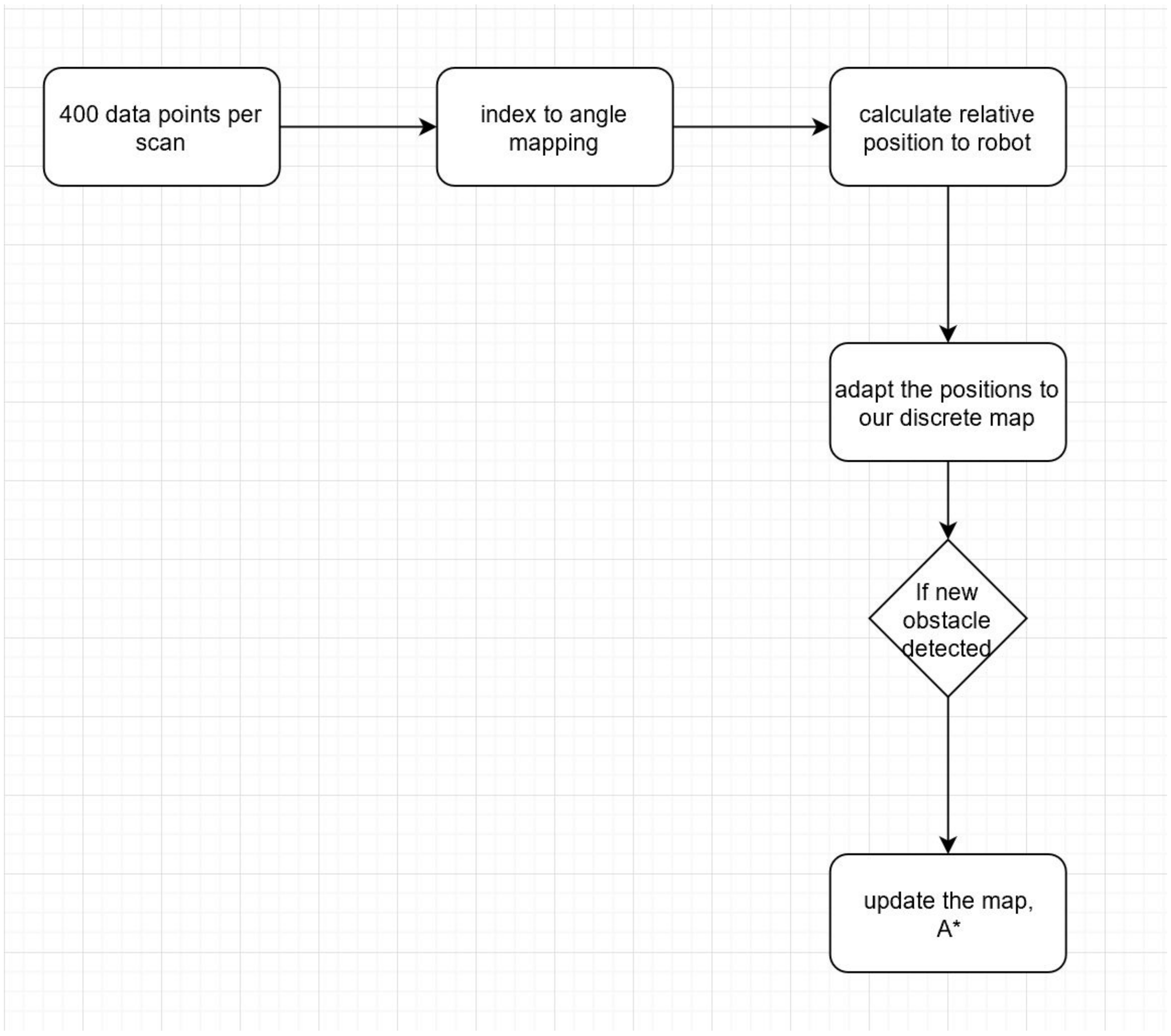


Figure 23. Mapping flowchart

2D Mapping	
Requirement	Verification
The robot should mark any obstacle it can see from its prospective	A. Putting the robot at home position for 5 seconds, then check the updated map, and compare it with actual environment B. Start the robot at home position and give an object location, check if the map is updated during the run

2.4.4.3 Path Planning

Based on the up-to-date map and the predefined location of the object, the robot will just generate a path using A*, with each step at the center of the gridded block on the map. While moving, the LiDAR will also collect surrounding information, and if a new obstacle is detected, the map will be updated and the path will be recalculated by A* based on the up-to-date map. Moreover, if there is no available path for the robot and target, the robot will go back to its starting position and standby.

Algorithm	
Requirements	Verifications
1. The robot will not hit obstacles during the whole process, including walls. 2. The robot will walk down the optimal path.	1. By observation, the robot should not hit any objects. 2. Robot will send back its current path to show this on 2d map.

2.4.5 Power Supply

2.4.5.1 Battery

We will use a 4S Li-Po rechargeable battery(14.8V, 7000mA) as the power source for the whole vehicle including the motors, controller, the signal receiver, and the sensor system. To power the different systems we need to connect the battery to the PCB board which will convert the voltage of the battery 14.8V for different components.

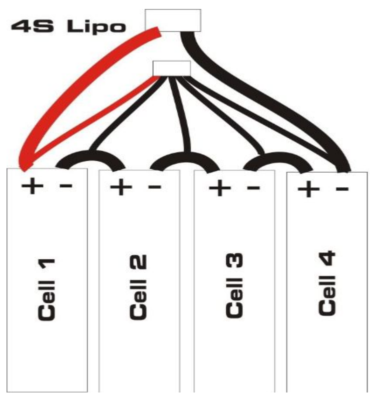


Figure 24. Battery overview

We are going to use the 4S Li-Po battery whose voltage is about 14.8V. Since the microcontroller works at 5V and the motors work at 3-12V, we will need converters to output different voltage for the different components in our robot.

Due to the working voltage range of components is lower than the voltage output from the battery, we will use the Buck DC/DC converter, which can step down the dc voltage. The Buck converter will need the TPS62130A chip as the central component. The circuit layout shows below is an example for 12V input to 3.3V output.

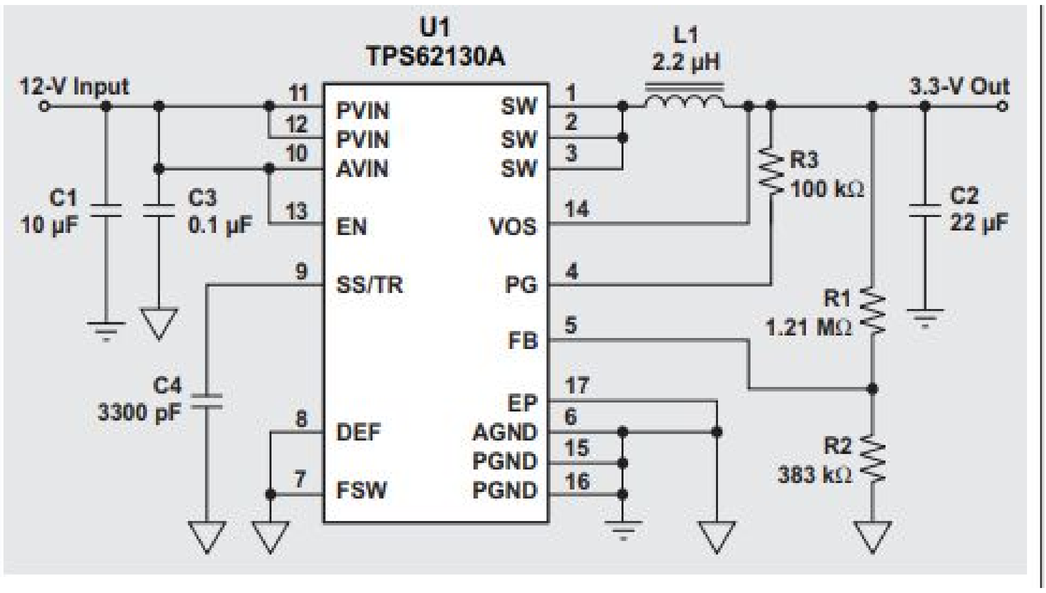


Figure 25. Circuit layout of TPS62130A [13]

Power Supply	
Requirement	Verification
The whole power set need to provide steady and plentiful power. The power converter, which is the DC-DC converter, need to	When we test the voltage out from the DC-DC converter, we should get the voltage with the difference between the expectation less than 5%

2.5 Tolerance Analysis

Our design contains two critical features for the successful completion of tasks: vehicle movement control and robot arm mechanism.

Vehicle movement control basically involves direction control and speed control, and the error caused by the gyroscope, wheel encoder, the control system will add up to total error. The error caused by gyroscope is 0.05°/sec specified by datasheet. Taking the worst scenario when there's +0.05/sec noise for the whole run, and for the robot to cover all grid starting from home position will take 200s, assuming the robot velocity at 0.3m/s.

$$\int_0^{200} (\omega + 0.05)dt = \int_0^{200} \omega\, dt + \int_0^{200} 0.05\, dt = \int_0^{200} \omega\, dt + 10^{\circ}$$

Equation 1: Final angle error after reaching objective

The the integrated error is around 10°. The error caused by wheel encoder would be that frequency is too low (1Hz), so the robot wouldn't be able to precisely calculate current position in between each 1s. Suppose our robot is set at 0.3m/s, then the max error the location could get while running in a straight line is 0.15m. This happens when the robot decelerates from 0.3m/s to 0m/s in 1 second.

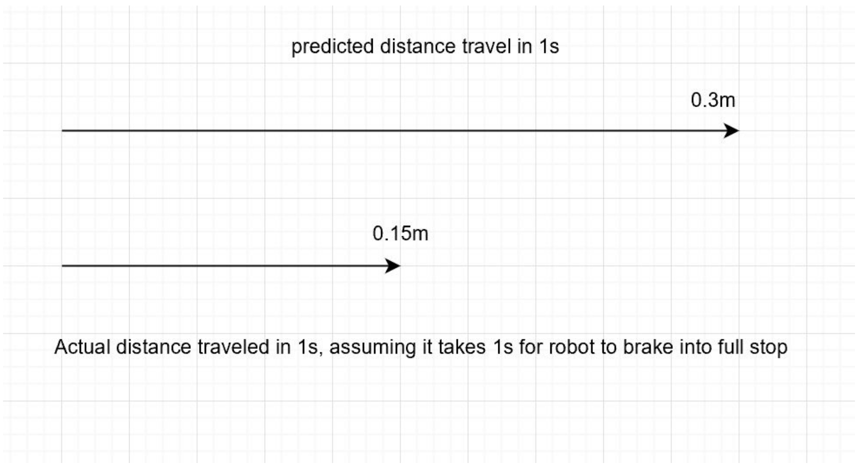

predicted distance travel in 1s
0.3m
0.15m
Actual distance traveled in 1s, assuming it takes 1s for robot to brake into full stop

Figure 26. Slow frequency error by encoder

The control system like speed control may have a tolerant overshoot (0.02m away from designated location), but will also add up to final error.

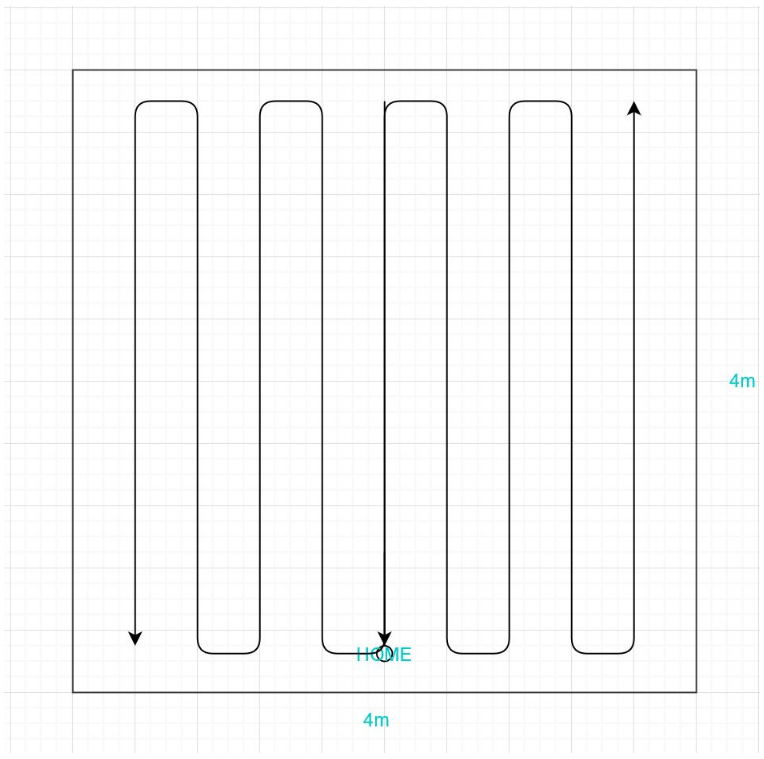


Figure 27. Worst scenario for the path of the robot

Let's say there's no obstacle in the environment, the robot is at home position (0,0), and the target location is the furthest corner (-4,9). Then as long as the robot final location is within 1 unit (0.4m) around the location, and within 31° facing the object location (half of the camera's range), the object detection algorithm will play the role for grabbing the object. And according to the previous error estimate, the max error after reaching the goal position is 0.18m and 10°, so the object detection algorithm will still be triggered to adjust robot position for object grabbing. The error of encoder could be reduced by slowing down, and the likelihood for the integrated angle to get up to 0.05°/sec error is diminutive. Therefore overall our vehicle movement control is feasible.

The critical requirement of the robotic arm mechanism is that the robot successfully recognizes the object user specified and picks it up. It basically relies on the high accuracy of relative position and orientation between the robot and the object and a reasonably high resolution of image data. Thus the critical components involved in the requirement are the camera and robot pose calibration algorithm.

When the robot arrives its destination given the 2D map with a location of the object, it needs to turn around and look for the object using the camera. If robot motion and localization is accurate enough, the object will show up in pictures taken by the camera. However, the object may not be at the exact position where it can be picked up by the arm, so we still need to adjust the relative position between robot and object based on image data from the camera.

Suppose we know the distance between the object and the robot if we count the number of pixels of the wristband image because a greater number of pixels should indicate a greater distance. Our assumption will be that if the centroid of an image of a wristband, which is around the object, is at the centerline of the picture (condition 1) and the number of pixels is in some predetermined range (condition 2), the robotic arm is able to pick up the item. Then we only need to adjust the position and orientation of the robot completely until these conditions are satisfied. Checking satisfaction of these conditions depends on the resolution of camera images because the accuracy of image position coordinates relies on the resolution.

Since we can only control the robot by small steps, say turning right by 1°, chances are the robot repeats turning left and right endlessly while adjusting the orientation, because the centroid is always at left or right of but never exactly at the centerline of the picture. Thus we must set up a tolerance in our position and orientation calibration algorithm. Suppose we allow condition 1 to be satisfied if the $(x_{centroid} - x_{center}) <= n_{tolerance}$, where $x_{centroid}$ represents the x coordinate of the centroid of a wristband, x_{center} represents the x coordinate of center of the picture, and $n_{centroid}$ is the tolerance value we set. Then this value should be chosen based on a large number of experiments. It needs to be as large as possible but still allows the robotic arm to pick up items.

3 Cost and Schedule

3.1 Cost Analysis

PARTS				
Index	**Part Name**	**Qt.**	**Price Per Item**	**Total Price**
1	**RPLIDAR A2**	1	319.95	319.95
2	**CAMERA**	1	25	25.00
3	**ROBOT ARM**	1	89.99	89.99
4	**VEHICLE**	1	40.00	40.00
5	**RASPBERRY PI**	5	39.95	199.75

6	**GYRO**	1	15.61	15.61
7	**MICROSD**	5	9.8	49.00
TOTAL COST				739.3

LABOR			
Name	**Salary ($/hour)**	**Hours**	**Total(*2.5)**
Kewei Sui	40.00	130	13000
Kefan Tu	40.00	130	13000
Honglu He	40.00	130	13000
Chengliang Li	40.00	130	13000
Siping Meng	40.00	130	13000
TOTAL COST			65000

3.2 Schedule

3.2.1 Schedule (upper part group)

	Kefan Tu	**Kewei Sui**
9/24/18	Physical design of the robot vehicle and the robotic arm	Arrange work schedule and host weekly meeting
10/1/18	Assemble the robotic arm; Research on motor driver circuit	Connect camera module to Raspberry Pi and sample pics from the video stream; Experiment OpenCV in Raspberry Pi
10/8/18	Implement a motor control circuit on breadboard; Successfully and separately control direction of two motors	Implement object detection algorithm in OpenCV and successfully recognize object with color wristband
10/15/18	Implement PCB design of motor controller; Test and analyze the torque the motor of arm can provide	Finish implementing object detection algorithm in order to calculate the direction toward the object and send this result to other algorithms

10/22/18	Implement arm control algorithm; Give correct and proper inputs to motor controller via microcontroller	Conduct simple test on object detection algorithms without vehicle and work on error reduction
10/29/18	Optimize the control algorithm of robot arm and conduct simple tests without vehicle and fix any problem shown during the test	Start to Implement localization calibration algorithm on camera without vehicle
11/5/18	Revise PCB design; Conduct more tests about robot arm with vehicle and fix any problem shown during the test	Continue Implement localization calibration algorithm and test object detection algorithm with vehicle
11/12/18	Test new PCB and optimize physical design of the entire robot.	Conduct tests on localization calibration algorithm with vehicle and fix any problem shown during the test
11/19/18	Conduct basic tasks and make sure the robot fulfill the high-level requirement about robot arm.	Work on stretch goal (VR) and make sure the robot fulfill the high-level requirement about the object detection
11/26/18	Conduct more real scenarios testing and fix any problem shown during the test Start to work on final paper and present mock demo	
12/3/18	Finish final paper Present mock presentation and demonstration	
12/10/18	Presentation	
12/12/18	Final paper due	

3.2.2 Schedule (lower part group)

	Chenliang Li	**Honglu He**	**Siping Meng**
9/24/18	Research on chassis car and suitable motors and power unit	Overall Design and research on various sensors (LiDAR, camera, encoder and gyroscope)	Research on microcontroller and algorithms about path planning
10/1/18	Purchase required hardware equipments	Localization and calibration design	Research on server setup and communication between Raspberry Pi and computer

10/8/18	PCB board design	LiDAR and Pi Serial Communication	Connection between Raspberry Pi and computer
10/15/18	Wheel function test	Map and coding project skeleton setup	Work on path planning algorithm and start to work on main function that controls all functions
10/22/18	PCB board made	2D Mapping of 4m×4m environment	Continue work on main function
10/29/18	Test all sensors in lower part and start to work on power supply unit	TCP/IP communication to send obstacle location to PC	Combining built algorithm code to main function and debug the error
11/5/18	refine PCB design and combine the entire robot together	Specify target location for the robot and drive itself without any path planning; Implement location calibration algorithm	Continue combine built algorithm code to main function and work on basic user interface.
11/12/18	Test new PCB and Start to prepare the objects and obstacles for experimental environment	Optimize path planning algorithm and test on location calibration algorithm	Optimize all algorithms in the microcontroller in order cut down the memory usage and speed them up
11/19/18	Conduct basic tests and make sure the robot fulfill all high-level requirements based on current experiment settings	Continue optimize path planning algorithm with LiDAR updated map on robot; conduct basic tests and make sure the robot fulfill the high-level requirement about the movement	Work on stretch goal (VR) and this new client is able to communicate with server
11/26/18	Conduct more real scenarios testing and fix any problem shown during the test Start to work on final paper and present mock demo		
12/3/18	Finish final paper Present mock presentation and demonstration		
12/10/18	Presentation		
12/12/18	Final paper due		

4 Stretch goals

4.1 Virtual Reality User Interface

More and more domestic robots appear in our houses. But they almost all use traditional UI to interact with the user which means that user can only send pre-programmed commands to these robots and the nonvisualized result is actually unclear to the user. For instance, when we click the “clean” button on the vacuuming robot, we, in fact, don’t know which part of the room will be cleaned and how clean it will be. We want to try some new interaction ways that can enlighten the smart appliance industry. With the VR experience, we designed a unique interaction way with a future smart appliance that we can manually produce the result we want in the virtual world and then a programmed robot could accomplish this task automatically in the real world and offline. High level of immersion in a virtual environment could improve the engagement of the user and facilitate he or she has a better understanding of the complex indoor environment space [14].

We will first ensure that by directly inputting P* and T into the microcontroller of the robot, it could work functionally. VR part is a kind of a stretch goal for our project. We decide to use Oculus Go with the controller as our front-end in this project, because of the portability of the helmet and good maneuverability of the controller. In VR, a user could emerge in a virtual environment that is exactly the same as the real experimental environment (the design of the map is shown in the introduction section). The user is able to see the target objects and the obstacles stored. Because we only save the 2D position for every objects and obstacle, models for them will be placed in the same x,y coordinates in VR but the height for those models may differ from that in the real world. By using the controller, we can grab the object freely and move it to somewhere else. Then, the new position P* of the target object T will be transmitted to the server and then sent to the robot in order to finish the task in the real world.

5 Ethics and Safety

We have several safety concerns about our project. The Li-po battery requires the highest attention to deal with due to the explosibility, and we will make sure the temperature of the battery stay in the safe range of the industrial standard for all time. Our charger is an industry made IC device and will shut charge controller off if charging input is beyond required voltage range. In this way, it can reduce the likelihood of hazards while charging. We will also design a protect circuit for sensor and motor components. To protect the circuit and PCB, we will use flyback diode and Transient Voltage Suppressor since the motor will also be connected to the PCB. What’s more, in order to prevent a short circuit which may lead to electric shock, we will follow the electricity using manual during the experiment, and will also check our power and circuit before connecting to the battery.

Another safety concern is about the vehicle. Since we will use four motors (2 for wheels and 2 for robot arm), we have to put malfunction of these parts into serious thoughts. If any software or circuit drive these motors mistakenly, the whole robot will be out of control and may hurt people standing nearby. This is definitely what we wish to avoid during the whole designing and demo period. In order to prevent such a condition, we will first accomplish a circuit test for each part separately. Then we will test the searching algorithm and robot control code in a safe laboratory as many times as possible. To increase safety, we will also make sure to build an emergency stop command on the computer so that we could make sure to stop the robot immediately and remotely.

For the VR headset (if applicable), we will operate it following the product safety manual and make sure while testing VR device, other teammates will be around to check if the user is experiencing discomfort. [15]

For the ethical issues, we will follow IEEE and ACM code of ethics. We may encounter many problems in the project. But when problems occur, we will not try to disguise the problems and move on recklessly. Based on #5 of the IEEE Code of Ethics, “to seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors” [16]. Therefore, when problems show up, we will try our best to find a way with teammates to solve them. If we can’t solve the problem by ourselves, we will turn to our TA for help.

6 Citation

[1] C., Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid,and J. Leonard (2016). Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age. *IEEE Transactions on Robotics*, 32(6), pp.1309-1332.

[2] I. Sutskever, G. Brockman, S. altman & E. Musk, “OpenAI Technical Goals”, June, 2016. [Online]. Available: https://blog.openai.com/openai-technical-goals/. [Accessed Oct. 4, 2018].

[3]E. Guizzo, “So, Where Are My Robot Servants?”, May, 2012. [Online]. Available: https://spectrum.ieee.org/robotics/home-robots/so-where-are-my-robot-servants. [Accessed Oct. 3, 2018].

[4] Boston Dynamics, “About Spot Mini”, *bostondynamics.com*, 2018. [Online]. Available: https://www.bostondynamics.com/spot-mini. [Accessed Oct. 3, 2018].

[5] “Introduction of Pibot,” Dec, 2017. [Online]. Available: https://blog.csdn.net/baimei4833953/article/details/78853797. [Accessed Oct. 1, 2018].

[6] “Robotic Arm Add-on Pack for Starter Robot Kit,” github.com, July 23, 2014. [Online]. Available:

https://github.com/Makeblock-official/Robotic-Arm-Add-On-Pack-For-Starter-Robot-Kit/blob/master/Assembly%20Instructions.pdf. [Accessed Sep. 20, 2018].

[7] Toshiba, "Driver IC for Dual DC motor," TB6612FNG datasheet, June, 2007.

[8] J. Hughes, "Camera Module Readme.md", Apr. 24, 2018. [Online]. Available: https://github.com/raspberrypi/documentation/blob/master/hardware/camera/README.md. [Accessed Oct. 4, 2018].

[9] A. Rosebrock, "OpenCV Gamma Correction", Oct. 5, 2015. [Online]. Available: https://www.pyimagesearch.com/2015/10/05/opencv-gamma-correction/. [Accessed Oct. 3, 2018].

[10] A. Rosebrock, "Ball Tracking with OpenCV", Sep. 14, 2015. [Online]. Available: https://www.pyimagesearch.com/2015/09/14/ball-tracking-with-opencv/. [Accessed Oct. 4, 2018].

[11] Instructables. (2018). *ADC MCP3008 (Raspberry Pi)*. [online] Available at: https://www.instructables.com/id/ADC-MCP3008-Raspberry-Pi/ [Accessed 5 Oct. 2018].

[12] "BTS 7960B High Current PN Half Bridge NovalithIC," BTS 7960B. [Online]. Available: https://www.infineon.com/dgdl/bts7960b-pb-final.pdf?folderId=db3a3043156fd5730116144c5d101c30&fileId=db3a30431ed1d7b2011efe782ebd6b60. [Accessed: 01-Oct-2018].

[13] C. Glaser, "Five steps to a great PCB layout for a step-down converter," Analog Applications Journal. [Online]. Available: http://www.ti.com/lit/an/slyt614/slyt614.pdf. [Accessed: 28-Sep-2018].

[14] H.L. Miller, N.Bugnariu "Level of immersion impacts the effectiveness of virtual environments used to assess or teach social skills in Autism Spectrum Disorder," *Cyberpsychology, Behavior, and Social Networking*, vol. 19, no. 8, p.246, Apr, 2016. [Online serial]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4827274/. [Accessed Oct. 2, 2018].

[15] Oculus VR stuff, Oculus Best Practice, Oculus VR, 2017.

[16] Ieee.org. (2018). IEEE IEEE Code of Ethics. [online] Available at: https://www.ieee.org/about/corporate/governance/p7-8.html [Accessed 19 Sep. 2018].