ECE 445
SENIOR DESIGN LABORATORY
FINAL REPORT

# REMOTE ROBOT CAR CONTROL SYSTEM WITH RGBD CAMERA FOR 3D RECONSTRUCTION

**Team Members**:
Yuhao Ge, yuhaoge2
Hao Chen, haoc8
Junyan Li, junyanl3
Han Yang, hany6

**Teaching Assistant**: Yiqun Niu
**Sponsor**: Prof. Pavel Loskot

Tuesday 23rd May, 2023

## Abstract

This report presents a remote car control system utilizing an RGBD camera, a low-power platform (Raspberry Pi), a 3D reconstruction algorithm, and an image compression transfer protocol. The system addresses two main challenges in current car control systems. Firstly, by employing 3D reconstruction, it provides comprehensive information and real-time modeling of the car's surroundings, eliminating the need for frequent manual inspection. Secondly, the system enables small, lightweight, and low-power-consuming car platforms by offloading computation-intensive tasks to a remote server. The network reception capability is enhanced, and an image compression transfer protocol is designed to optimize bandwidth usage. Besides, the system uses McNamee wheels and PID control to achieve omnidirectional motion and allows remote car control via a joystick. Users can also get a third-person view of the car and its surroundings. The findings demonstrate the effectiveness of the system in achieving safe, fast, and robust car control in complex environments.

**Keywords:** remote car control system, 3D reconstruction, image compression, McNamee wheels, PID control.

# Contents

# 1. Introduction

## 1.1 Problem and Solution Overview

Nowadays, there is an intensive need for remote vehicle control systems such as robotic control, car control, and drone control. And the requirements for those systems are no longer limited to "being able to work" but extend to a safe, fast, low-latency, and robust control system in complex environments. When exploring the danger or dedicated environments, robots have their own advantages like standing in the extreme environment. Their experience will be useful for humans. However, the current design of such systems poses the following challenges:

- **The lack of comprehensive information about the surroundings:** For those vehicles that only have one camera showing the image, they lack information and memory about the surroundings in other directions, and even there are some solutions utilizing multiple cameras to sense the environment, they have drawbacks like dead zones, high costs, and large space usage for installment. Also, without the memory, their pre-exploration will be difficult for humans to reuse.
- **Excessive size and high-power consumption:** for some use cases such as disaster area detection, and historical sites detection, the vehicle platform should usually be small and with low power consumption to meet the endurance requirements, while any Image processing modules usually require a large amount of computation and thus need a large on-vehicle energy and computation module.

The existing remote-control systems utilized in applications such as robotics and drones encounter difficulties when operating in complex environments, resulting in inadequate control over the machine. Moreover, for remote control, operators usually lack a comprehensive view of their surroundings, thereby increasing the likelihood of accidents. Therefore, it is essential to design a remote-control system that enables the user to effectively maneuver the car in intricate surroundings, while providing a comprehensive view of the environment, even when the vehicle is beyond the range of visual perception.

## 1.2 Solution

In our project, we aim to address the challenges mentioned above using an RGBD camera, a low-power platform (Raspberry Pi), and a 3D reconstruction algorithm [3] running on a remote server with the support for an image compression transfer protocol.

We consider 3D reconstruction a solution for the first difficulty mentioned above, as the 3D reconstruction module record and build a model for the surrounding in real time [2], which means the vehicle and the user could have memory for every scene it saw and an integrated 3D model instead of only one scene. Users can have a sense of what's around the vehicle at any time and don't have to turn around the vehicle to check the environment frequently. Also, information on the surroundings would be accumulated as the vehicle keeps digesting the images of the environment.

As for the second difficulty, we decided to only use a Raspberry Pi on the vehicle and not handle those calculation-intensive jobs on the vehicle but on a remote server so the vehicle can be small, light, and low power-consuming, however, network bandwidth becomes a bottleneck for the performance. We tackled this from two directions, one is to enhance the network reception capability, and the other is to design an image compression transfer protocol that saved the bandwidth usage for each image.

With our product, users will be able to control the robot car remotely via a joystick while also viewing the car and its surroundings from a third-person perspective [1]. The perspective can be

easily shifted to allow for a better understanding of the car's surroundings.

## 1.3 Visual Aid

The figure on the left shows a car being navigated around a location. The figure on the right shows a user inspecting the environment by viewing reconstructed images of the surroundings and maneuvering the car using a joystick.

*Subfigure 1*: Environment Scenario

*Subfigure 2*: User Interface with Joystick Controller

**Figure 1 Pictorial Representation of Our Solution**

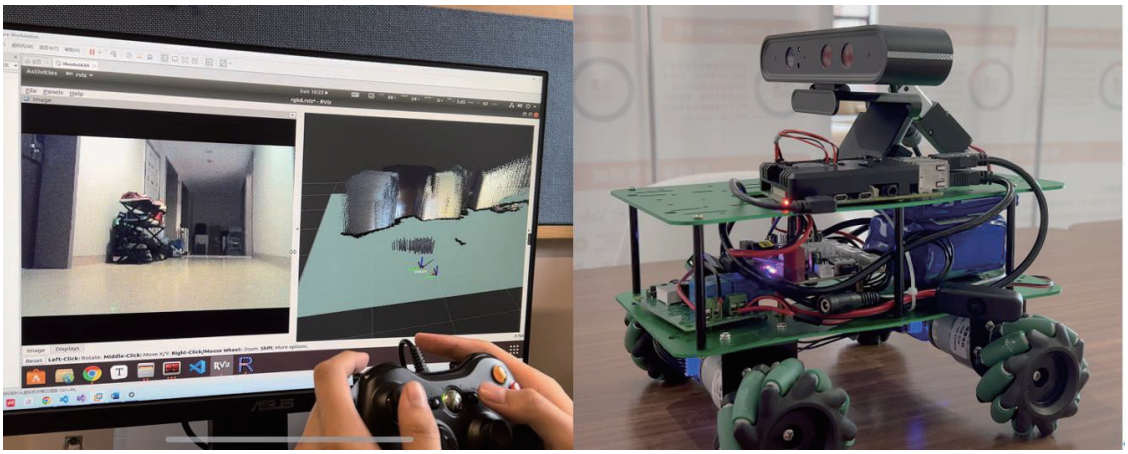

**Figure 2 Implemented User Interface and Robot Car**

*Figure 1* is generated by Power Point and *figure 2* is the screen shot of our implemented project. They should provide a good illustration of our solution. A more detailed design of our project is revealed in next section.

# 2. Design

## 2.1 Block Diagram

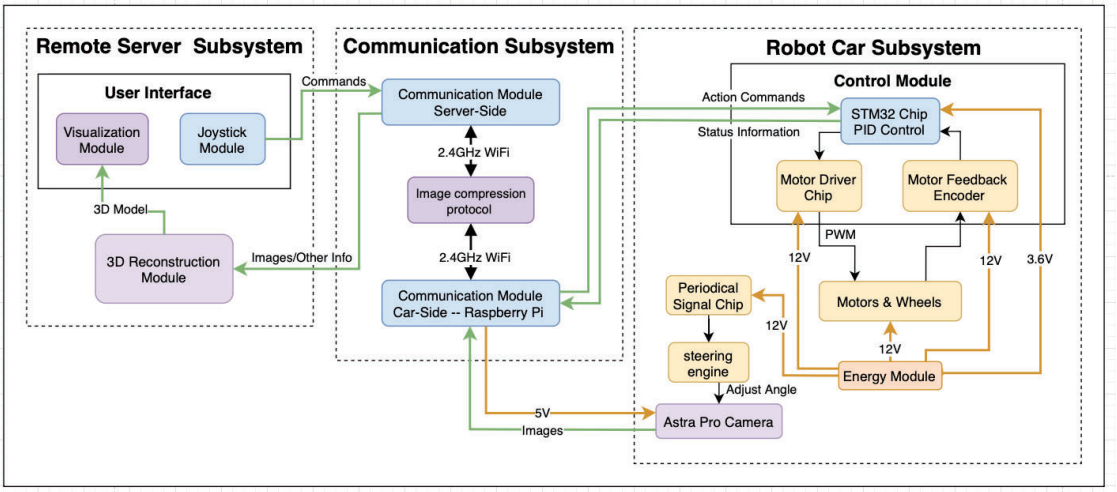


Figure 3 Block Diagram of our Solution

As shown in *figure 3*, our solution is divided into three subsystems framed with dotted lines,

1. a **Remote Server Subsystem** which will handle the visualization and the 3D-reconstruction tasks for users to observe the surroundings,
2. a **Communication Subsystem** which works as a communication bridge between the remote server and robot car for data and commands transfer, and
3. a **Robot Car Subsystem** which is a car platform that supports omnidirectional movement holds a RGBD camera to gather information and can be controlled by a joystick.

Each subsystem contains several modules that work together to achieve specific features.

## 2.2 Remote Server Subsystem

The remote server subsystem is a powerful computer that works as a terminal between the car and the user. It receives the control signals from the joysticks and sends them to the car while processing the images from the camera through the Raspberry Pi then output the map. Thanks to the light computation algorithm, the quality of the output map mainly depends on the resolution and the rate of the images. The 3d reconstruction result is presented in a point cloud and it can be viewed from different perspectives. Also, we write the joystick control function by ourselves to achieve timely and accurate control of the car.

As the brain of our project, it is critical that our server connects with other subsystems well. All the communication is built through the Ros system and the remote server is the master node. Nonetheless, we set up a fully functional user interface to aid the user with the map created concurrently. The user interface enables the user not only to look at what is in front of the car but also to have a clear view that where it is. The surrounding environment is presented as well.

### 2.2.1 3D reconstruction Module

A **3D reconstruction module** processes RGBD camera images with OpenCV and performs 3D reconstruction using algorithms such as SLAM (simultaneous localization and mapping) or a structure from motion. As the user gradually moves the car around the environment, the 3D reconstruction module should be working simultaneously to keep building up the surrounding

environment's model. We would use a Ubuntu system on the server to carry the ROS together with the RTAB-Map (Real-Time Appearance-Based Mapping) algorithm.

The whole process would be real-time which means the calculation time for each frame should be short (a small delay) and the reconstructed model should be smooth and continuous, namely even if the photos have gaps when we try to combine the continuous ones, the model should be able to fix the gap and make a consistent output.

The main reconstruction process is as below:

• **Image Denoising with Mean Filter**: The initial step involved the utilization of a mean filter to denoise the images received from the Raspberry Pi. This filter computed each pixel's value as the mean of its eight neighboring pixel values, thereby effectively reducing the noise and enhancing the quality of images.

• **Image Sampling**: In this stage, we calculated the color gradient of the images and established sample points based on a pre-determined threshold. This ensured a more detailed and precise representation of the image's features, leading to a more accurate reconstruction.

• **Feature Matching**: Feature matching was conducted to correlate sample points between consecutive images, thereby bridging the gap between them. This essential step enabled the continuity of the 3D reconstruction and improved the overall accuracy.

• **Trajectory Calculation**: Utilizing visual odometry, we calculated the trajectory of the camera, enabling us to stitch together continuous images seamlessly. This provided us with a detailed view of the robot car's path and facilitated a comprehensive spatial understanding.

• **Closure Detection**: The system was designed to record the features of each image for future closure detection. If the similarity measure between two images exceeded a set threshold, the algorithm identified that the camera has revisited a previous location, thereby detecting a loop closure.

• **Calibration**: Upon confirmation of a closure detection, the map underwent refinement. This process involved adjusting the height and relative location of the objects to better align with real-world measurements, enhancing the overall fidelity of the 3D reconstruction.

• **Post-Processing**: The final stage involved the refinement of the rudimentary map through post-processing. This involved verifying the location of each point against its k nearest neighbors and filtering the points within a certain radius to reduce overlap, thus further improving the accuracy and clarity of the final 3D model.

Here, we have implemented two strategies,

• The first one is image drop. Since our camera can provide up to 30fps images, we drop two images from every three inputs to reduce the redundant images.

• The other is we set up the closure detection rate to be relatively strict, i.e., 5 times for each image. Since our resolution of images is not high, a high resolution will be helpful to generate a consistent map.

Our user interface's reconstruction result is real-time with accurate location and size. Since our goal is not to recognize the objects, the texture of the objects will not be detailed. After the scanning, a final map will be represented in the point cloud, and the user can view it like a car inside the map to move around.

In our implementation, the image resolution is 640*480 while the rate is around 10fps, and the algorithm is remembering 10s images. So we need

$$Storage = ImageSize * Rate * MemoryPeriod = 19.53 \text{ Mb} \quad (1)$$

Based on this implementation, our final output will have the same resolution and the maximum points within the image size will be 50000. And due to the bandwidth of communication being 80Mb/s, the reconstruction rate is up to

$$Rate = \frac{BandWidth}{\frac{Resolution}{ImageRate}} = \frac{80 * 2^{20}}{\frac{640 * 480}{10}} = 27\text{fps} \quad (2)$$

But the channel has to transmit the signal of joysticks, so the rate of reconstruction is always lower than expected.

### 2.2.2 Joystick Module

The Joystick Module receives signals from the joystick's sticks, triggers, and buttons. Then converts them into $(Vx, Vy, Vz)$ as the output data, which is the target movement of the robot car. The control signal is transmitted to the Raspberry Pi on the robot car via the communication subsystem, and then the control module takes further control of the car's movement. The detailed joystick function map is attached as *Table 1*.

| Name | Function | Name | Function |
|---|---|---|---|
| Left Stick X-axis | Move Left/Right | Left Stick Y-axis | Move Forward/Backward |
| Left Trigger | Turn Left | Right Trigger | Turn Right |
| Left Button | Decrease Max Turning Speed | Right Button | Increase Max Moving Speed |
| Start Button | Emits a beep | Back Button | Stop Moving |

Table 1 Joystick Function Map

The sticks can give a value range from $(-1, 1)$ and triggers can give a value range from $(0, 1)$. So, to better control the movement of robot car, we can first define a maximum moving speed $V_{max,\ xy}$ and maximum turning speed $V_{max,\ z}$. They can be adjusted by left and right button. Then map the value of left sticks into $(-V_{max,\ xy}, V_{max,\ xy})$ and value of riggers into $(-V_{max,\ z}, V_{max,\ z})$ so the car can move under a large range of speed. In the actual test and demo session, the user experience of the joystick control of the car movement is very good.

## 2.3 Communication Subsystem

We rely heavily on WIFI to transmit the data between the car and the remote server. The transmitted data includes the RGB image and depth image captured from the camera, which will be sent from the car to the remote server so that we can perform 3D reconstruction based on these input images. Another transmitted data is the movement command sent from the server to the car so that we can control the movement of our car remotely.

### 2.3.1 Image Transmission and Compression

Accurate 3D reconstruction requires high-resolution RGBD images with low latency [4]. The RGBD camera is connected to the Raspberry Pi on the car, and once the image is captured, the Raspberry Pi will send the image back to the server through a WIFI connection. If the image is received with large latency, usually packet dropping will occur, and the reconstruction algorithm will be confused and won't be able to perform reconstruction well. We empirically find that a resolution of 640x480 and a frame rate of 10FPS or above are required to achieve a good 3D reconstruction result[5]. Suppose we can use 4 bytes to represent a single RGBD pixel (one byte for R, G, B, and depth channel respectively), the required bandwidth to transmit the original bitmap image with a frame rate of 10 FPS would be:

$$640 \times 480 \times 4 \times 10 \approx \frac{12.3MB}{s} = 98.4Mbps \quad (3)$$

This is a very high requirement for a WIFI connection, and from our connection test, we find that if we transmit the original image, usually only 5 FPS or even lower can be achieved, depending on the WIFI quality. It is important to take some actions to reduce the image transmission latency and ease the packet-dropping issue due to the high latency.

As a result, we propose to use two techniques: image compression and data throttle. Image compression can greatly reduce the transmitting image size and thus reduce the required bandwidth, and data throttle can restrict the frame rate to a certain number so that the transmission process and the data input can be smoother.

First, to reduce the required bandwidth, we use a JPEG image compression algorithm to compress the image before sending it, and then decompress the compressed one when received on the server side. JPEG (Joint Photographic Experts Group) is a widely used image compression algorithm that effectively reduces the size of digital images while preserving reasonable image quality. The JPEG compression algorithm works by exploiting the limitations of human visual perception and the statistical properties of natural images. It consists of several steps:

1. **Color space conversion**: The algorithm begins by converting the RGB color space of the original image into the YCbCr color space. The luminance (Y) component represents the brightness information, while the chrominance (Cb and Cr) components capture color differences.
2. **Downsampling:** Since human vision is more sensitive to changes in brightness than color, the chrominance components (Cb and Cr) are subsampled to reduce their resolution. This downsampling step reduces the amount of data required to represent the image while maintaining reasonable color accuracy.
3. **Block-based Discrete Cosine Transform (DCT):** The image is divided into 8x8 pixel blocks, and for each block, a Discrete Cosine Transform (DCT) is applied separately to the Y, Cb, and Cr components. The DCT converts each block from the spatial domain to the frequency domain, representing the image's content in terms of different frequency components.
4. **Quantization:** In this step, the DCT coefficients are divided by a set of quantization tables specific to the compression level chosen by the user. The quantization process eliminates high-frequency information that is less noticeable to the human eye. By quantizing more aggressively, higher compression ratios can be achieved at the cost of reduced image quality.
5. **Entropy coding:** The quantized DCT coefficients are then compressed using lossless entropy coding techniques, specifically the Huffman coding algorithm [9]. Huffman coding assigns shorter codes to frequently occurring values and longer codes to less frequent values, resulting in further compression.

Figure 4 Reconstruction Result of Lab

To reconstruct the image, the compressed data is reversed. The inverse of the entropy coding and quantization steps are applied, followed by the inverse DCT. The resulting YCbCr components are unsampled (in case of down sampling in the initial step) and converted back to the RGB color space. We find that up to a 5x compression ratio can be achieved using the JPEG image compression algorithm in our case [6].

Second, to ease the packet-dropping issue and stabilize the frame rate, we apply data throttle. To be specific, for every X sequential image data, we drop the last Y images among them, so that we can restrict the frame rate to (X-Y)/X of the original rate. For example, the original frame rate is 30 FPS for our camera. Since we want to achieve a stable 10 FPS frame rate for our 3D reconstruction algorithm to run smoothly, we can drop 2 images among 3 sequential images.

As shown in figure 4, our reconstruction result of the lab clearly shows the wall and the 3d printers. The angle of the wall and the connections are not awkward.

### 2.3.2 Control Command Transmission

Our car can be controlled remotely using a joystick connected to the server. The joystick control signal will be handled by the server and the server will send the corresponding high level control signal to the Raspberry Pi through Wi-Fi. The control signal packet has the following format:

1. For set_car_run command:

{"command": "set_car_run", "state": state, "speed": speed, "adjust": adjust}

2. For set_car_motion command:

{"command": "set_car_motion", "v_x": v_x, "v_y": v_y, "v_z": v_z}

3. For set_beep command:

{"command": "set_beep", "on_time": on_time}

The packet will be padded to the same size to avoid the sticky TCP packet issue. After receiving, the Raspberry Pi will parse the control command and control the STM32 board accordingly.

## 2.4 Car Subsystem

The car subsystem is a movable robot car platform, including a car platform, wheels, motors, and a STM32 based control module integrated with an energy module. The main functionality of the car subsystem is that it can accept movement commands $(V_x, V_y, V_z)$ to perform actions such as moving and turning and provide an expandable platform for various devices placement such as Raspberry Pi and RGBD camera.

As an essential platform and carrier for our project realization, the robot car subsystem is closely connected and interacted with other subsystems and modules. For example, it needs to first interact with the communication module (car side), read the control signals coming from the remote server, and then adjust the movement of the car according to the control signals.

### 2.4.1 McNamee Wheels and Motors

To enhance the ability of robot car to move freely in complex environments, the McNamee wheels were selected to support omnidirectional movement. The McNamee wheel is a classical mechanic for omnidirectional movement. However, the adoption of the McNamee wheel requires the control module to have individual control over each wheel and motor. The configuration and simple kinetic analysis of the McNamee wheel is shown in *figure 5*.

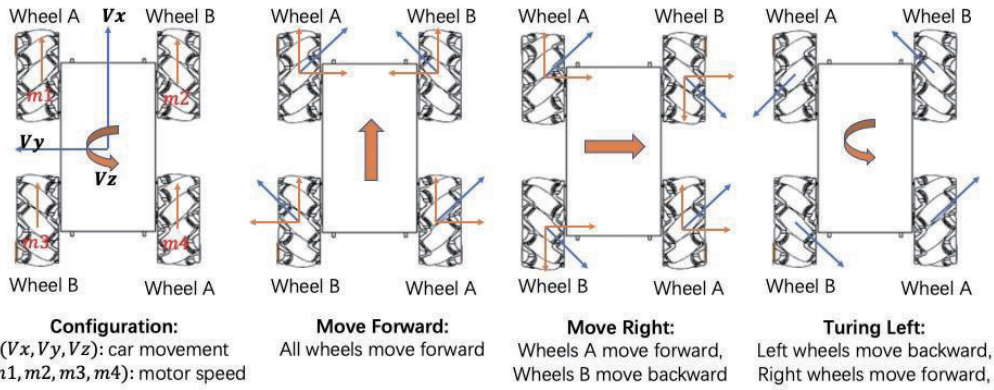


**Figure 5 Configuration and Simple Kinetic Analysis of McNamee Wheels**

Given the configuration, the forward and inverse kinematics of a robot car with four McNamee wheels gives the following relationship between the target robot car movement speed $(V_x, V_y, V_z)$ and the speed of each wheel $(m_1, m_2, m_3, m_4)$. Equation 4 to 7 give the forward kinematics and equation 8 to 10 give the backward kinematics [7].

$$m_1 = v_x - v_y - (l_x + l_y)v_z \tag{4}$$

$$m_2 = V_x + V_y + (l_x + l_y)V_z \tag{5}$$

$$m_3 = V_x + V_y - (l_x + l_y)V_z \tag{6}$$

$$m_4 = V_x - V_y + (l_x + l_y)V_z \tag{7}$$

$$\text{Longitudinal Velocity: } V_x = \frac{m_1 + m_2 + m_3 + m_4}{4} \tag{8}$$

$$\text{Transversal Velocity: } V_y = \frac{-m_1 + m_2 + m_3 - m_4}{4} \tag{9}$$

$$\text{Angular Velocity: } V_z = (-m_1 + m_2 - m_3 + m_4)\frac{1}{4(l_x + l_y)} \tag{10}$$

The motor needs to provide sufficient power for the movement of the robot car system and guarantee a reasonable running speed under a limited weight load. We choose to use the MD520Z30 12V motor with AM2857 Power Driver to satisfy our requirement.

As measured in our robot car platform, half of the sum of the chassis motor spacing: $(l_x + l_y) = 164.555$ mm. Then, given the target $(V_x, V_y, V_z)$, the target speed of each McNamee Wheels $(m_1, m_2, m_3, m_4)$ an be quickly calculated by forward kinematic.

### 2.4.2 Control Module

Inside the robot car subsystem, the control module receives the robot car's motion command $(V_x, V_y, V_z)$ and generate the PWM signals $(m_1, m_2, m_3, m_4)$ to drive the motors to achieve the required movement. We use an STM32-based control board to control the four McNamee motors individually. STM32 is a widely used low-energy microcontroller that can easily communicate with Raspberry Pi and PC via serial communication. Besides, STM32's advanced timer can facilize PWM signal generation for the motor driver chips and calculate motor speed through encoders.

Although one can directly control the motion of the car by calculating the PWM value of each wheel given its target speed explicitly, the car as a whole is likely to fail to reach or maintain the target speed when facing obstacles or working on rough road surfaces. In this case, PID algorithms are widely used to provide stable control and implemented as follows.

1. First, the car's true speed of wheel $m_i$ is calculated using encoders. The encoder can count the number of motor rotations $C$. Then the control module can use the change in number of rotations $\Delta_C$ multiplied by the length of wheel $l_i$, divided by the time interval $\Delta_t$, which gives the speed of the wheel $m_i$. The calculation is given by equation 11.

$$m_i = \frac{\Delta_C \times l_i}{\Delta_t} \tag{11}$$

2. Second, based on the error $e_i$ between target speed, the correct PWM signal for the four motors are generated according to the incremental PID control algorithm. Digital implementation of the PID controller [8] gives the following equations,

$$\Delta u(t_k) = K_p[e(t_k) - e(t_{k-1})] + K_i e(t_k) + K_d[e(t_k) - 2e(t_{k-1}) + e(t_{k-2})] \tag{12}$$

$$u(t_k) = u(t_{k-1}) + \Delta u(t_k) \tag{13}$$

   where in equation 12, $\Delta u(t_k)$ is derived with a digital PID algorithm. $e(t_k)$ is the difference between the target value and the current value at time $t_k$. We use $K_p = 0.8$, $K_i = 0.06$, and $K_d = 0.5$ as our PID algorithm parameters. In equation 13, $u(t_k)$ is the actual value at time $t_k$ calculated by adding last time unit output value $u(t_{k-1})$ with an incremental part $\Delta u(t_k)$.

3. Then, four DC motor driving chips are used to receive the PWM signals and drive the motor accordingly.

The following Pseudocode in *figure 6* clearly illustrates the workflow of Robot Car control via PID control algorithm, from receiving control signal $(Vx, Vy, Vz)$ from Joystick Module.

```
def motor_motion_control():
    while (Joystick_Module_Ready == True) and (Motor_Ready == True):
        (Vx, Vy, Vz) = receive_control_signal_from_Joystick_Module()
        (cur_m1, cur_m2, cur_m3, cur_m4) = read_motor_speed_from_encoders()
        (tar_m1, tar_m2, tar_m3, tar_m4) = forward_kinematics(Vx, Vy, Vz)
        for wheel in wheels:
            wheel_pwm_value = PID_calculation(
                wheel = wheel,
                target_val = (cur_m1, cur_m2, cur_m3, cur_m4),
                actual_val = (tar_m1, tar_m2, tar_m3, tar_m4)
            )
            set_motor_speed(wheel, wheel_pwm_value)
```

**Figure 6: Pseudocode of Robot Car Control**

### 2.4.3 Astra Pro Camera and Car Platform Placement

In our design, an RGBD camera (Astra Pro) will be placed on the robot car. The camera is connected to and powered by Raspberry Pi through USB. The camera could provide live information of color and depth with precision in millimeters. The images captured by the camera will be sent to a remote server via a communication module for further processing. To provide stable image signals for 3D reconstruction, the horizontal movement speed $V_x$ and the rotational speed $V_z$ of the robot car need to be limited. The suitable maximum speed of $(V_x, V_y, V_z)$ is $(0.4, 0.4, 0.2)$.

An expandable car platform is the base of our car subsystem, with wheels, motors, a control module, communication module (car side), and an RGBD camera to be assembled and fixed on it.

We use pre-punched metal plates combined with copper columns to construct the Car Platform that provides sufficient space and rich mounting positions for device placement. The Car Platform also guarantees the structural integrity and stability of the equipment placement when working under the vibration of the motors and bumps during movement. Besides, the robot car platform can be facilitated for assembling and disassembling devices such as motors, Raspberry Pi, and RGBD cameras.

### 2.4.4 Mechanical Structure

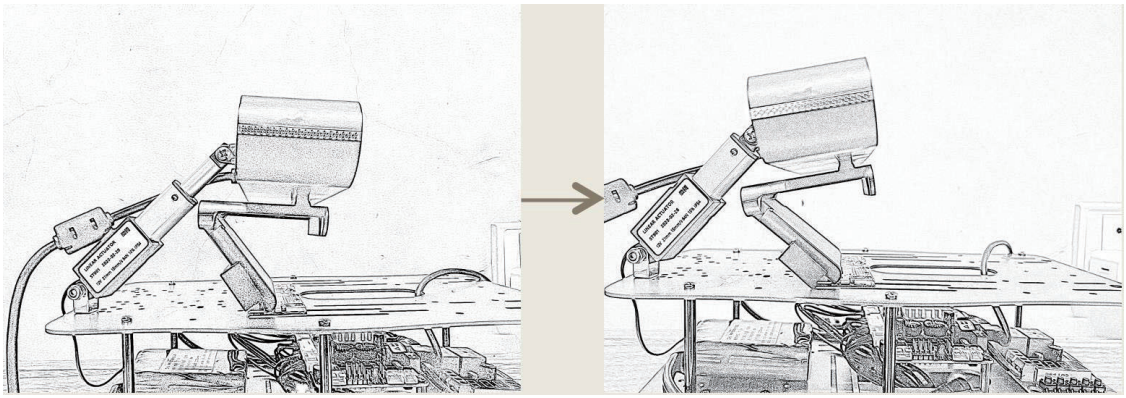

Figure 7 A Structure that Helps the Camera to Adjust the Pitch Angle

During the above testing process, we discovered that due to the limited height of the car and the limited field of view of the camera, incomplete modeling could occur. For example, there could be missing floor or ceiling areas, and objects at higher positions were often difficult to capture by the camera. To address this issue, we designed a mechanical structure for automatically adjusting the camera's pitch angle.

We utilized a small Linear Actuator-driven rod, with one end fixed to the rear of the car and the other end fixed behind the camera. The extension and retraction of the rod would cause changes in the camera's pitch angle. The motion direction of the Linear Actuator was determined by the polarity of the engaging electrodes. To control its periodic variation, a customized PCB was developed to achieve alternating polarity of the power supply. As a result, this device can now automatically and periodically rotate the camera up and down, effectively solving the problem of occasionally missing the ceiling or floor during capture.

# 3. Design Verification

## 3.1 Remote Server Subsystem

A 3D reconstruction module processes RGBD camera images with OpenCV and performs 3D reconstruction using algorithms such as SLAM (simultaneous localization and mapping) or a structure from motion. As the user gradually moves the car around the environment, the 3D reconstruction module should be working simultaneously to keep building up the surrounding environment's model. We would use an Ubuntu system on the server to carry the ROS together with the RTAB-Map (Real-Time Appearance-Based Mapping) algorithm.

The whole process would be real-time which means the calculation time for each frame should be short (a small delay) and the reconstructed model should be smooth and continuous, namely even if the photos have gaps when we try to combine the continuous ones, the model should be able to fix the gap and make a consistent output. Detailed requirements and verification would be illustrated in the chart.

### 3.1.1 R&V of 3D Reconstruction and Visualization

| Requirement | Verification |
|---|---|
| 1. Reconstruct the frame with 10fps and the output will be not less than 0.05m resolution | 1. The reconstruction and resolution will be shown on the user interface. And it can be verified with grid size. |
| 2. The whole reconstruction should be done within 10% of scanning time and the processing delay should be less than 0.5s. | 2. The delay can be check with rostopic /rtabmap/cloud_map/, and the concurrent reconstruction result will be shown on the user-interface. |
| 3. The User should be able to see clear icons for mode switching | 3. We provide two user interfaces, both of them including mode switching icons. |
| 4. The error of the final result should be limited under 10%. For example, the length and height of objects should be in range 90% to 110% with respect to the real one. | 4. There are measurement tools to check to value of the object height and length inside the point-cloud. |

Table 2 R&V of 3D Reconstruction and Visualization

### 3.1.2 R&V of Joystick Module

| Requirement | Verification |
|---|---|
| 1. Joystick Module can monitor and read the joystick signal correctly.<br>2. Joystick Module can phrase the joystick signal to the control command for Control Module | 1. Connect the joystick to the remote server and run the script, the correct joystick signal value should be shown.<br>2. The user's control action should be phrased correctly and generated in a predefined format. |

Table 3 R&V of Joystick Module

## 3.2 Communication Subsystem

### 3.2.1 R&V of Communication Subsystem

| Requirement | Verification |
|---|---|
| 1. RGB and depth images with a resolution of at least 240x320, optimally 480x640, will be sent from the Raspberry Pi to the remote server at a speed of 5-10 FPS. | 1. The RTABMAP should be able to receive the RGBD signal from the camera and launch the reconstruction process on the remote server properly. |
| 2. The image is compressed during transmission to save bandwidth. | 2. Compare the image data size prior to sending and after receiving to determine whether the compression works. |
| 3. Control commands can be delivered smoothly to Raspberry Pi from the remote server. | 3. The final demo will be a live presentation with the output on the user interface. And the resolution details should show on the screen. |

Table 4 R&V of Communication Subsystem

### 3.2.2 Image Transmission and Compression

For image transmission, the requirement is that both RGB and depth images should have a resolution of at least 320x240, optimally 640x480 and the transmission speed from the Raspberry Pi to the server should be 5-10 FPS. We verify the image resolution by making sure that the output image resolution from the camera is the desired resolution and verify the frame rate from RTAB-map UI.

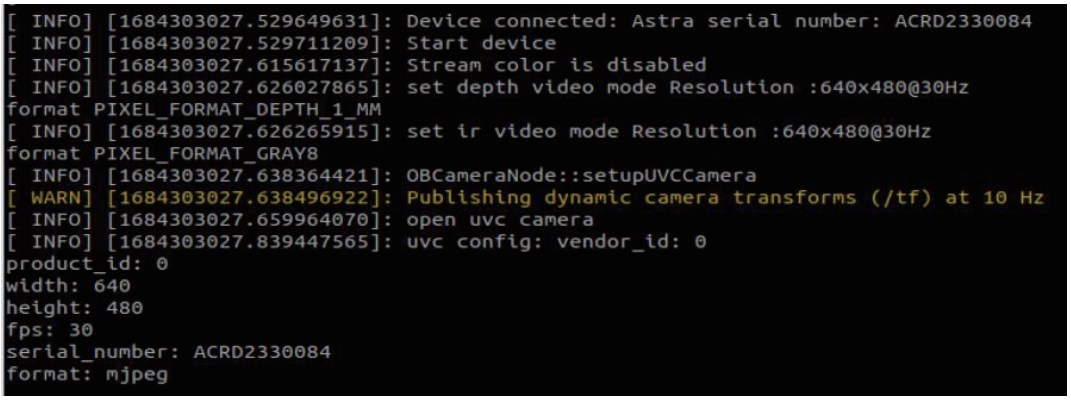


Figure 8 Camera Information

From the camera information output shown in *figure 8*, we can see that the RGB and depth image are both in a resolution of 640x480. Although the camera FPS is 30 as shown in the figure, the frame rate we observe from the RTAB-map UI is fluctuating between 3 – 12 FPS depending on the software running condition and the distance between the car and the WIFI router.

For image compression, the transmitted image is required to be compressed, and the bandwidth should be reduced compared to the uncompressed image. We verify reduced bandwidth using the rostopic bw command.

As shown in the *figure 9* below, we can achieve a compression rate of 0.12 using the JPEG image compression algorithm. Consequently, shown in *figure 10*, the required bandwidth is dropped to nearly 10% of the original bandwidth. It shows the necessity and importance of the image compression algorithm if we want to use our car with limited bandwidth.

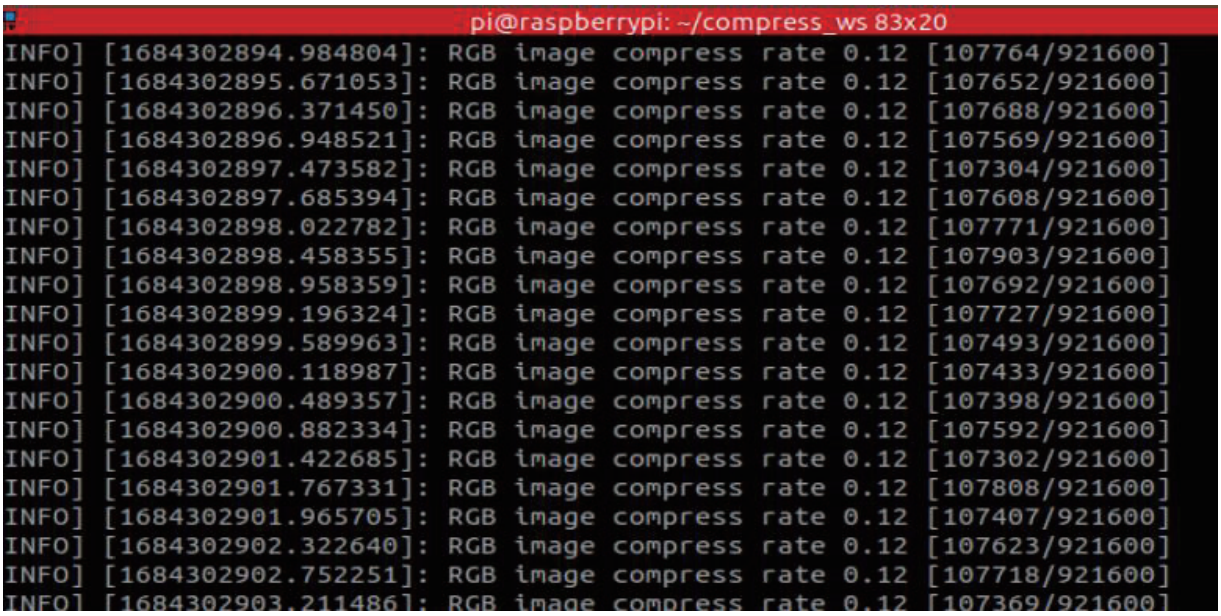


Figure 9 Image Compression Rate

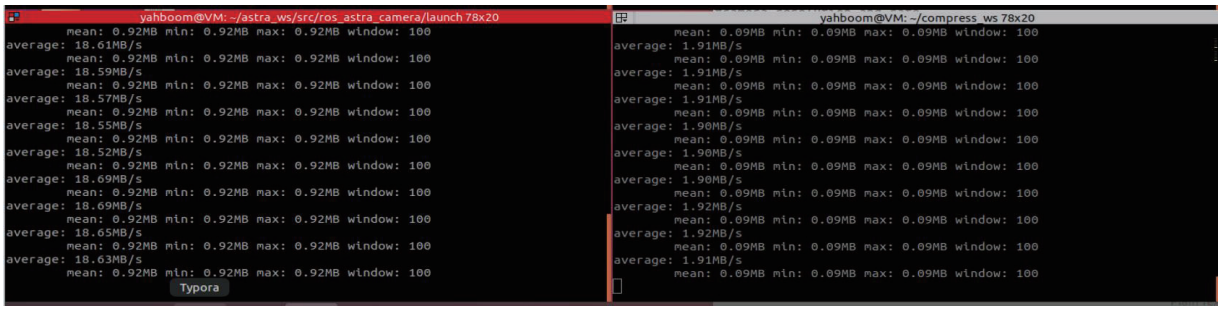


Figure 10 Bandwidth w/wo Image Compression

### 3.2.3 Control Command Transmission

For control command transmission, the requirement is that the command signal can be sent successfully from the server to the car and the car can handle the command signal correctly. We verify this by checking the actual movement of the car after receiving the control signal. Through our testing, the car responds fast and moves smoothly, meaning that the control command can reach the car without delay.

## 3.3 Car Subsystem

### 3.3.1 R&V of Car Subsystem

| Requirement | Verification |
|---|---|
| **Structural Integrity of Car Platform**<br>1. The car platform is light, with the strong structural integrity of a 3-4 kg loading.<br>2. No equipment loose or drop during long working periods. | 1. Load testing of the robot car platform with 3-4 kg of weight.<br>2. Perform a movement test on bumpy roads, and do acceleration, deceleration, and turning operations.<br>3. Record the car condition among the tests as the results. |
| **Control Module**<br>1. The target motor speed can be calculated by forward kinetic of McNamee wheels to achieve high-level movement command $(V_x, V_y, V_z)$.<br>2. The current motor speed can be calculated using motor encoder.<br>3. The motor speed can be controlled accurately using PWM calculated by incremental PID algorithm.<br>4. Robot car can turn at a speed of about 25 degrees/second according to the command.<br>5. Robot car can move omnidirectionally and turn simultaneously based on the commands. | 1. Perform a moving test with custom movement commands sent from Raspberry Pi, i.e., left and right panning, forward and backward, rotation, and a combination of these movements.<br>2. Evaluate the performance of these movements, i.e., how well they meet the user's expectations.<br>3. Write down a reference table for the data interface of the control module.<br>4. Record and evaluate the performance of the average time delay between a given input and the output. |
| **Motor and Wheels**<br>1. Robot car can move freely in complex environments and achieve omnidirectional movement.<br>2. The motor and wheels can provide the appropriate and adjustable speed, i.e., moving omnidirectionally at a maximum speed of 0.5-1 m/s, turning 180 degrees in 2s.<br>3. The motor needs to provide sufficient power to enable the robot car to run at a normal speed of 0.5-1m/s when forward and backward under a reasonable weight load 2 kg. | 1. Perform a moving test with predefined movement to cover an omnidirectional movement set, i.e., left and right panning, forward and backward, rotation, and a combination of these movements.<br>2. Perform a forward and backward movement test under a 2 kg weight load, the speed is within 0.5-1m/s.<br>3. Perform a movement test on bumpy roads, and do acceleration, deceleration, and turning operations.<br>4. Take a video for the omnidirectional movement test, and record the speed and rotation performance in a table. |

Table 5 R&V of Car Subsystem

The verification of the car subsystem mainly focuses on the structural integrity of the car platform, the operability of the control module, and the quick response of the motor and wheels. Testing the operability of the car subsystem is more important than testing individual modules of it and can give a clearer reflection of the performance of a robot car. Therefore, the verification of the car subsystem is combined with the test of the Joystick module.

In the actual test, the car is very flexible, and can move omnidirectionally and turn simultaneously. When moving on a more frictional carpet, the robot car moves at a reduced speed, but can still maintain normal movement speed by increasing power. However, when moving in an environment where the ground is slightly unstable, the car may get trapped. The solution is to stop the car and then start again.

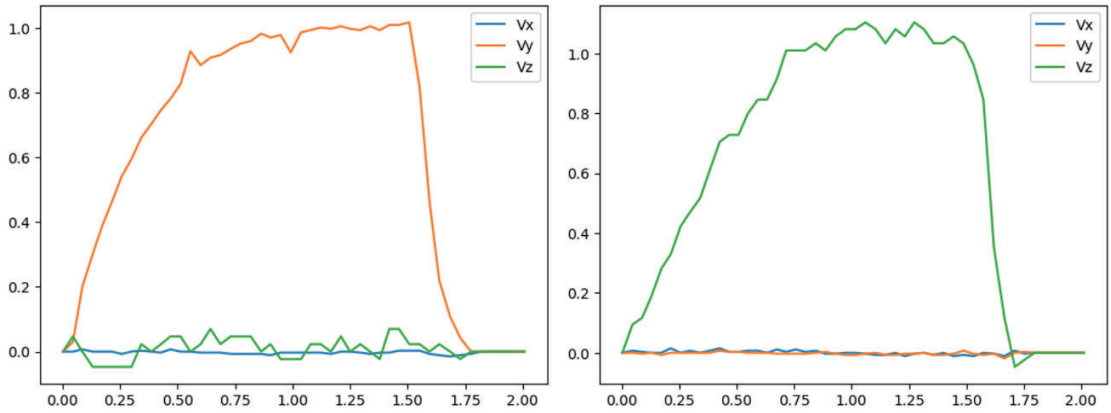


**Figure 11 Movement Speed vs. Time**

The rotation speed of each wheel can be calculated by recording the motor speed through the encoder, and the overall speed of the car can be further calculated based on the backward kinetic. The above *figure 11* shows that the car can reach the set speed of 1m/s in about 0.8s for both horizontal movement and rotation, where 1m/s is the maximum running speed of the car. Besides, it can also stop the motion within 0.2s.

# 4. Costs

## 4.1 Parts

| Part | Manufacturer | Retail Cost ($) | Bulk Purchase Cost ($) | Actual Cost ($) |
|---|---|---|---|---|
| Cooling Fan | RPi Club | 5.6 | / | 5.6 |
| Linear Actuator | ShaoTeng | 18.8 | / | 18.8 |
| PCB | ShaoTeng | 11.9 | / | 11.9 |
| 12V Li Battery | YahBoom | 12.7 | / | 12.7 |
| STM32 Board | YahBoom | 54.2 | / | 54.2 |
| 12.6V Li Battery & Charger | YahBoom | 12.6 | / | 12.6 |
| Car Platform | YahBoom | 61.3 | / | 61.3 |
| Astra Pro Camera | YahBoom | 150.0 | / | 150.0 |
| Joysticks | Microsoft | 50.0 | / | 50.0 |
| Raspberry Pi | WAVE ELEC-TRONICS | 75.4 | / | 75.4 |
| **Total** | | **452.5** | **/** | **452.5** |

Table 6 Parts Costs

## 4.2 Labor

Labor for each partner: $37(\$/\text{hour}) * 10(\text{hours/week}) * 7(\text{weeks}) = \$5180$

## 4.3 Total

The sum of costs into a grand total $5180 * 4 + 452.5 = \$21172.5$

# 5. Conclusion

## 5.1 Accomplishments

We conducted tests of our system in various scenarios. The system successfully reconstructed the indoor environment, including rooms, furniture, and walls. The accuracy of location and size measurements in the reconstructed model was high. The texture details of objects were adequately captured, providing a realistic representation of the environment.

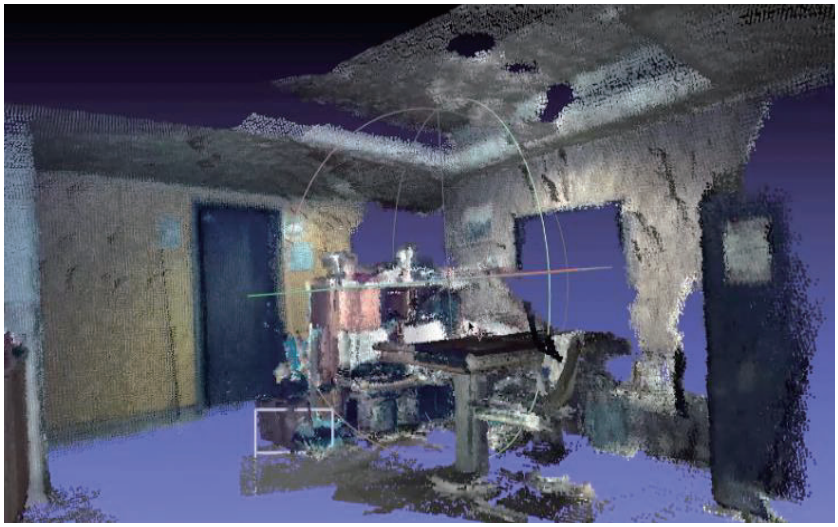

Figure 12 A Reconstructed Model Built in The Dormitory Unit Area

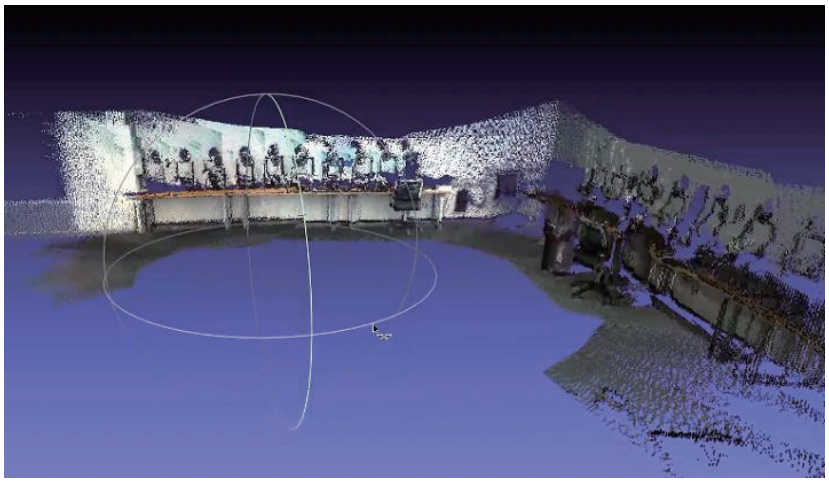

Figure 13 A Reconstructed Model Built in The Electronic Innovation Laboratory

In this project, we developed a remote vehicle control system that addresses the challenges associated with operating vehicles in complex environments. The system focuses on providing a safe, fast, low-latency, and robust control system by incorporating comprehensive information about the surroundings and optimizing size and power consumption.

To overcome the lack of comprehensive information about the surroundings, we utilized the Simultaneous Localization and Mapping (SLAM) technique. SLAM enables the vehicle to estimate its pose and construct a real-time map of the environment. We specifically implemented the RTAB-Map algorithm, which combines visual appearance-based mapping with loop closure detection and global optimization techniques. This algorithm allows for accurate localization and

mapping by integrating data from multiple sensors, such as cameras, LIDAR sensors, and RGB-D sensors.

To handle the issue of excessive size and high power consumption, we proposed offloading the computational tasks to a remote server. The vehicle captures camera images and transmits them to the server for processing, visualization, and 3D reconstruction. This distributed processing approach optimizes energy and computation usage on the vehicle platform, making it more suitable for low-power and endurance-focused applications.

The communication subsystem bridges the remote server and the robot car, enabling the transfer of data and commands. A reliable Wi-Fi connection is established for efficient and dependable communication. To mitigate latency and bandwidth limitations, we introduced image compression techniques and data throttling to optimize the transmission of high-resolution RGBD images.

For the design of remote robot car control, we chose to use the McNamee wheel to achieve omnidirectional movement and programmed the PID control algorithm in the control system for stable motion control. Besides, to provide a better operating experience, we designed various operations in the joystick module based on the actual operating requirements, so that the user can easily control the robot car's movement, such as omnidirectional movement, turning, and adjusting the maximum moving speed.

Furthermore, we enhanced the field of view for the vehicle's camera by incorporating a mechanical structure that controls the camera's tilt angle. This improvement enables the camera to capture a broader perspective, leading to more accurate 3D reconstruction results.

The results of our system demonstrate its effectiveness in reconstructing indoor environments with high accuracy in location, size measurements, and texture details. The system successfully addressed the challenges posed by complex environments and achieved the objectives of safe, fast, and robust vehicle control.

## 5.2 Uncertainties

While our system has demonstrated successful 3D reconstruction capabilities during our demonstrations and rigorous verification processes, it is important to acknowledge the presence of certain uncertainties and limitations. These include:

**Environmental Factors:** The accuracy and quality of 3D reconstruction heavily depend on the environmental conditions in which the robot car operates. Factors such as lighting conditions, texture variations, and reflective surfaces can potentially impact the reliability of the reconstruction. During our testing, we empirically find that if the environment is too simple and lacks color changing, it is likely that the visual odometry and close loop detection used during the 3D reconstruction process will fail because there are not enough visual key points. It is crucial to consider these factors and understand that the system's performance may vary in different environments.

**Latency and Network Stability:** Remote control and real-time transmission rely on network connectivity and stability. Although we have proposed several techniques including image compression and data throttle to ease any potential network bandwidth issues, the presence of latency in the network communication may still introduce delays in the control inputs and the delivery of images to the remote server. Moreover, fluctuations in network stability can affect the system's performance, leading to potential disruptions in the control and reconstruction processes.

**Computational Resources:** The complexity of real-time 3D reconstruction requires substantial computational resources. While our system is designed to perform reconstruction on a remote server, the processing power and capabilities of the server can impose limitations. Moreover, the

Raspberry Pi is also responsible for some computation-heavy tasks such as compressing images and aligning depth and RGB images, and we have observed performance bottleneck on image compression which in turn affects the final frame rate.

While our system has achieved significant advancements in remote robot car control and 3D reconstruction, it is essential to recognize and address these uncertainties and limitations. Continued research, development, and collaboration with experts in the field will help us further refine and improve the system's capabilities, enabling more robust and accurate 3D reconstructions in diverse scenarios.

## 5.3 Ethical Considerations

In addition to technical aspects, addressing the ethical considerations associated with this project is crucial. The following contents are listed to highlight the key ethical concerns:

**Privacy and Data Protection:** As the system captures and transmits images from the robot car's RGBD camera, the protection of privacy and data becomes a significant ethical consideration. It is essential to ensure that the system adheres to strict privacy policies and regulations, particularly when operating in public or sensitive areas. Measures such as anonymization of data, secure transmission protocols, and user consent frameworks should be in place to safeguard privacy and protect the personal information of individuals captured in the images.

**Consent and Awareness:** Obtaining appropriate consent is crucial when utilizing the system in environments involving human subjects. Broadcasting notification should be established to inform individuals about the presence and purpose of the robot car and the collection of visual data. Raising awareness about the system's capabilities and potential impact on privacy and allowing individuals to opt-out or request restrictions on data collection are essential ethical considerations.

**Responsible Use and Impact:** The development and deployment of the system should adhere to responsible use practices, considering the potential impact on individuals, communities, and the environment. It is essential to conduct thorough risk assessments to identify and mitigate potential negative consequences. Proactive measures should be taken to address any unintended harm, promote transparency, and ensure the system's overall benefits outweigh its potential risks.

By addressing these ethical considerations, and adhering to relevant legal and ethical guidelines, we can strive to develop and deploy a remote robot car control system that respects privacy, promotes fairness, and upholds ethical values while harnessing the potential benefits of advanced 3D reconstruction technology.

## 5.4 Future Work

The *figure 14* below is a result of the reconstruction of my room. It is evident that a portion of the wall behind the objects on the upper shelf lacks any point cloud data. This is due to occlusion, where the walls below the extended lines connecting the camera and the outer edges of the objects are not captured by the camera. Even adjusting the camera's pitch angle, this issue remains challenging to resolve. Future research may involve exploring adjustments to the overall camera height to capture the occluded parts effectively.

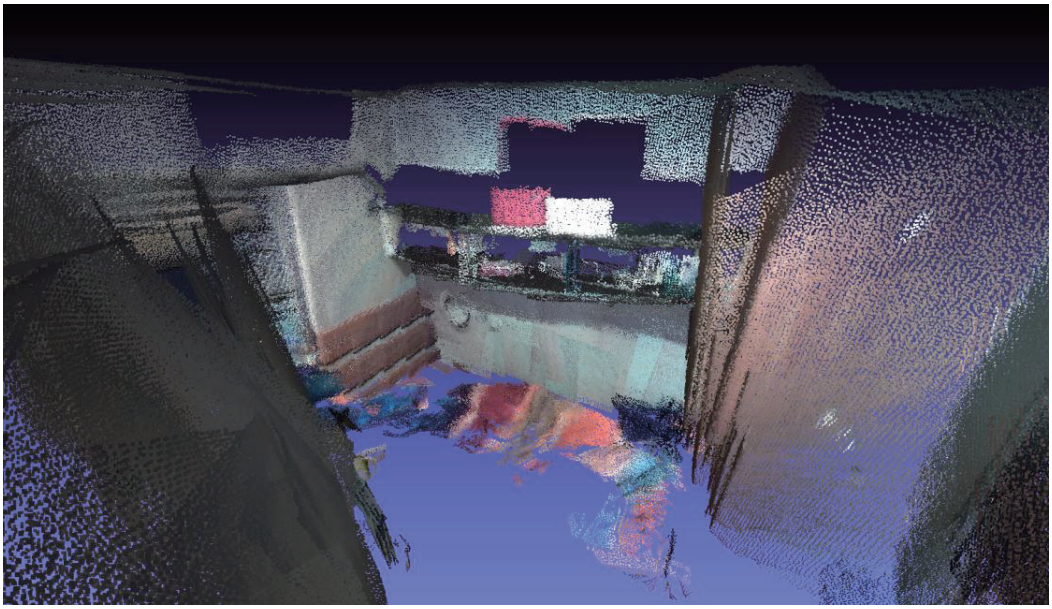

Figure 14 A Reconstructed Model Build of Students' Room

## References


[1] M Aharchi and M Ait Kbir. A review on 3d reconstruction techniques from 2d images. *In Innovations in Smart Cities Applications Edition 3: The Proceedings of the 4th International Conference on Smart City Applications* 4, pages 510–522. Springer, 2020.

[2] Angela Dai, Matthias Nießner, Michael Zollh¨ofer, Shahram Izadi, and Christian Theobalt. Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface reintegration. *ACM Transactions on Graphics (ToG)*, 36(4):1–18, 2017.

[3] Michael Firman. Rgbd datasets: Past, present and future. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 19–31, 2016.

[4] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, David Freeman, Andrew Davison, and Andrew Fitzgibbon. Kinectfusion: Real-time 3d reconstruction and interaction using a moving depth camera. *In Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 559–568. ACM, 2011.

[5] Chen Liu, Jiaye Wu, and Yasutaka Furukawa. Floornet: A unified framework for floorplan reconstruction from 3d scans. *In Proceedings of the European conference on computer vision (ECCV)*, pages 201–217, 2018.

[6] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. *In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4460–4470, 2019.

[7] H. Taheri, B. Qiao, and N. Ghaeminezhad, “Kinematic model of a four mecanum wheeled mobile robot,” *International Journal of Computer Applications*, vol. 113, no. 3, pp. 6–9, Mar. 18, 2015, ISSN: 09758887. DOI: 10.5120/19804-1586.

[8] P. Bhandari and P. Z. Csurcsia, “Digital implementation of the PID controller,” en, *Software Impacts*, vol. 13, p. 100 306, Aug. 2022, ISSN: 2665-9638. DOI: 10.1016/j.simpa. 2022.100306.

[9] Moffat, Alistair. "Huffman coding." *ACM Computing Surveys (CSUR)* 52.4 (2019): 1-35.