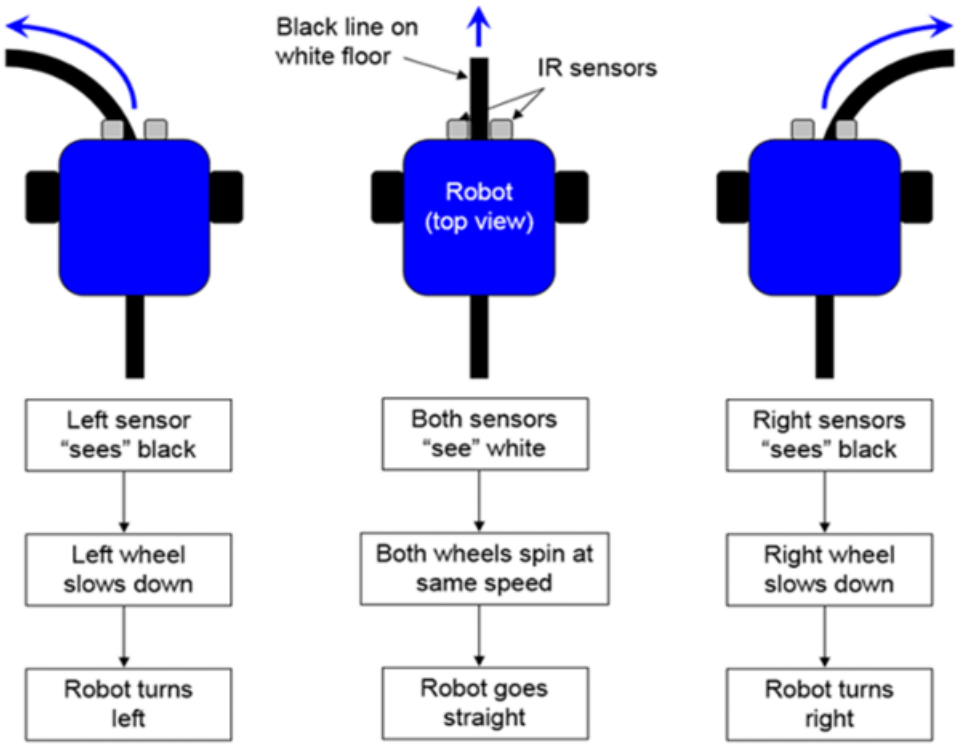
Line Following Buggy with SLAM Capability

The vehicle uses an onboard computer to convert the images and data received from the camera attached on the vehicle into 3d structures which is later processed into a map. This explains the concept of SLAM. As of now the vehicle uses a line following system for movement and actuation.



How does it move?

The robot uses a simple yet effective line-following mechanism for navigation. It is equipped with an onboard infrared (IR) sensor that continuously monitors the surface beneath it. The working principle is based on the reflective property of surfaces: when the IR sensor emits infrared light, it detects the amount of light that is reflected back. If the surface is white or reflective, the IR light bounces back and is detected by the sensor, indicating that the robot is not on the black line. In such a case, the robot recognises that it has deviated from its path and adjusts its direction accordingly. On the other hand, if the surface is black, it absorbs the infrared light, and little to no light is reflected back to the sensor. This signals the robot that it is on the correct path, prompting it to continue moving forward.

For movement, the robot uses two DC motors that are connected to the rear wheels, enabling forward and backward motion. Steering is managed by a servo motor connected to the front wheel assembly. What makes this setup unique is that the steering mechanism isn't fixed to a specific path; instead, it actively scans the area. The IR sensor is mounted on a servo that rotates it to the left, right, and forward, helping the robot determine the direction where the black line continues. Based on this input, the servo adjusts the steering to keep the robot aligned with the path.

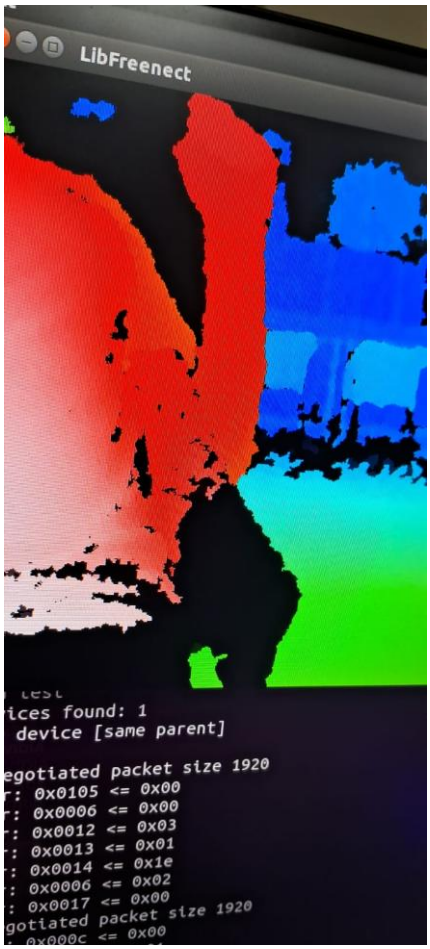
This approach not only allows the robot to follow a predetermined track, but also gives it a degree of adaptability to find the correct route even at intersections or sharp turns.

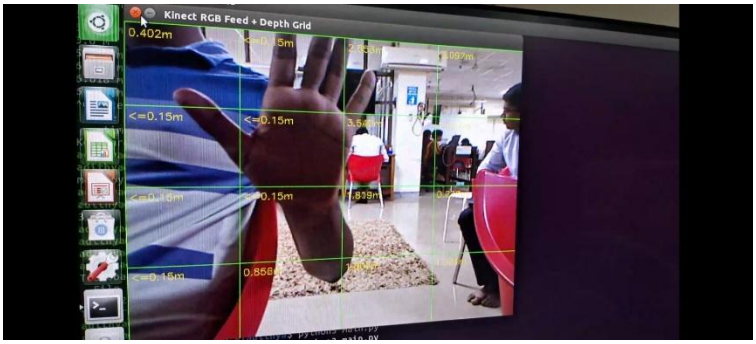
WEEK 1

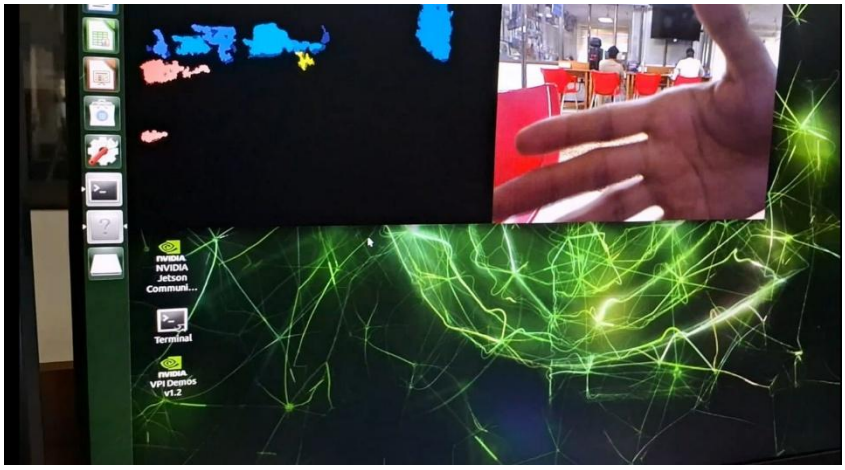
The initial phase of the project involved developing a clear understanding of the buggy's architecture and the individual components integrated into the system. The primary hardware modules identified at this stage were the Xbox Kinect V1, NVIDIA Jetson Nano, and ESP32 microcontroller. I reviewed the respective datasheets and technical documentation for each component to gain a comprehensive understanding of their specifications, capabilities, and interfacing requirements.

Following the hardware analysis, I proceeded to explore suitable software frameworks for processing data from the Kinect V1. After surveying various open-source libraries, I selected **libfreenect** due to its lightweight design and ability to stream both RGB and depth data directly to the Jetson Nano in real-time. I successfully interfaced the Kinect with the Jetson Nano using libfreenect and utilized **OpenCV** for basic image processing and depth-based distance estimation. This setup formed the foundation for subsequent vision-based tasks and concluded the first week of the project.

WEEK 1 PROGRESS







WEEK 2

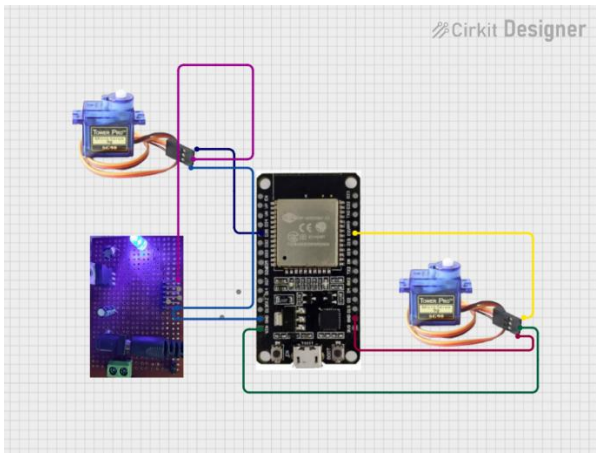
In the second week, I received the remaining components required for the system, including the ESP32 microcontroller, servo motor, and infrared (IR) sensors. I began by integrating these modules with the Jetson Nano to test compatibility and establish communication. I wrote custom firmware for the ESP32 to interface with both the servo and the IR sensor, while making necessary adjustments—such as tuning the IR sensor's sensitivity to improve detection accuracy and ensuring stable PWM control of the servo motor via the ESP32.

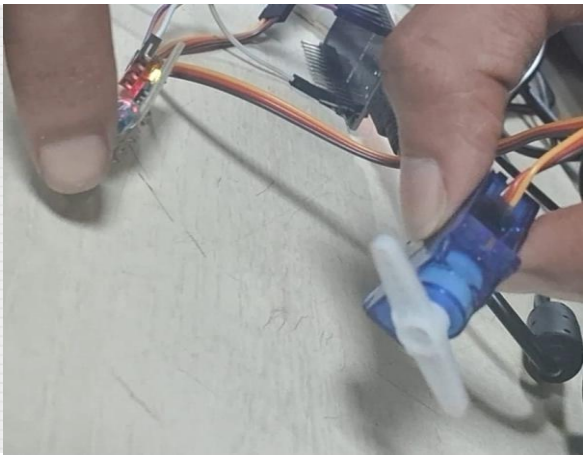
During this phase, I was also introduced to the buck converter, which played a critical role in stepping down the input voltage from 12V to 5V. This allowed me to safely power all actuators and sensors, while ensuring that only logic-level signals were connected to the ESP32's GPIO pins. The use of the buck converter also improved system stability by electrically isolating high-current components from the control logic and preventing voltage mismatches.

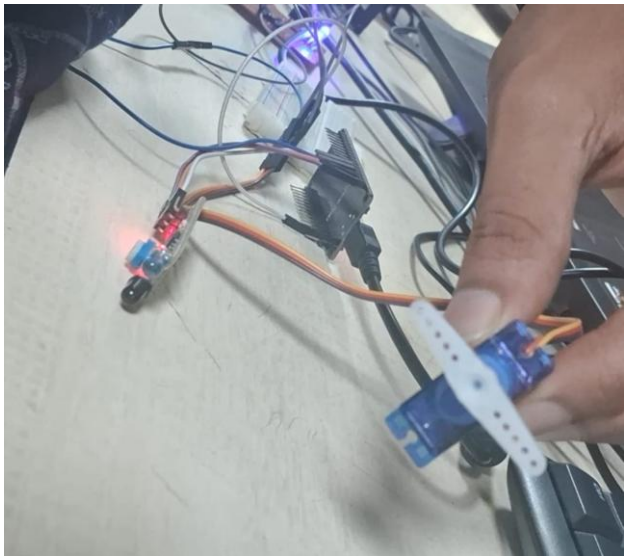
To verify the IR sensor's functionality, I conducted basic tests by placing a reflective object (e.g., a finger) in front of the module. The onboard indicator LED responded correctly, confirming successful detection. Based on the sensor signal, I programmed the ESP32 to trigger a 180-degree sweep of the servo motor to dynamically scan for a black line, providing directional correction during navigation.

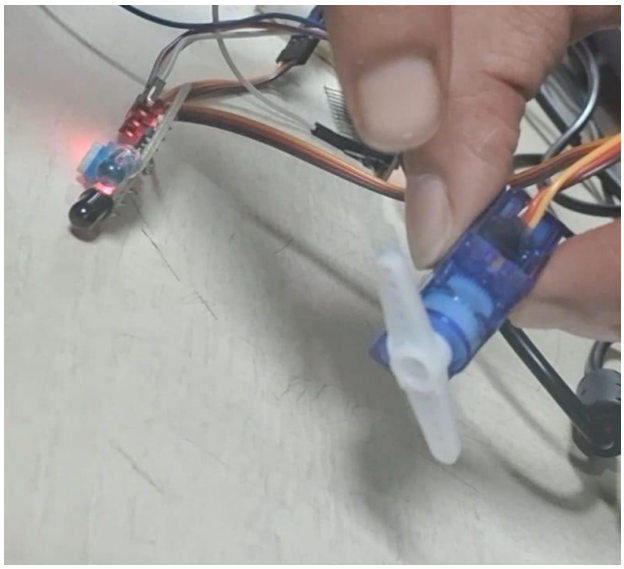
Additionally, I attempted to integrate servo control with a Pixhawk flight controller and simulate its behavior using ArduPilot software. However, this simulation was unsuccessful due to unforeseen configuration or compatibility issues, and further debugging was deferred to prioritize the working ESP32-based system. As the project evolved and system complexity increased, the servo-based scanning approach was eventually revised for greater efficiency. A more robust and responsive design was implemented using two fixed-position IR sensors for real-time line tracking, eliminating the need for mechanical movement and simplifying the overall control logic.

WEEK 2 PROGRESS









WEEK 3

In the third week, I focused on exploring **SLAM (Simultaneous Localization and Mapping)** and **3D mapping techniques** that could be implemented on the buggy. The goal was to enable the system to generate a spatial map of its environment in real time as it moved. While evaluating available solutions, I had to take into account the **hardware limitations** of the **Jetson Nano**, particularly its **limited storage**, **restricted computational power**, and **thermal stability**.

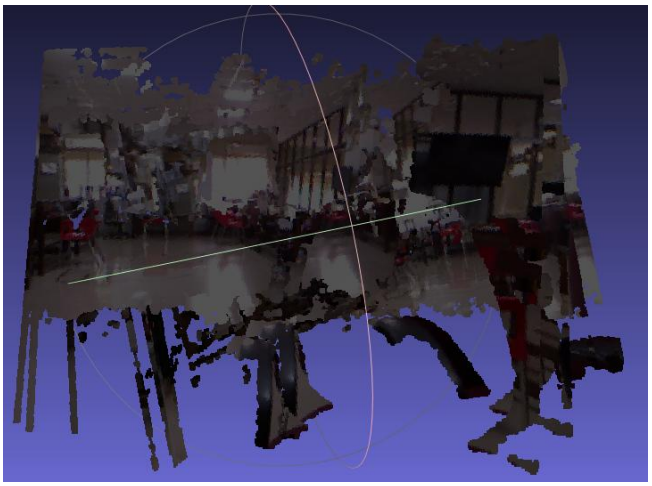
During initial experimentation, I attempted to install and run various open-source SLAM frameworks such as **RTAB-Map**, **ORB-SLAM2**, and **LDSO**. However, repeated failures and system crashes occurred primarily due to memory constraints and resource-heavy dependencies, which overwhelmed the Jetson Nano's capabilities.

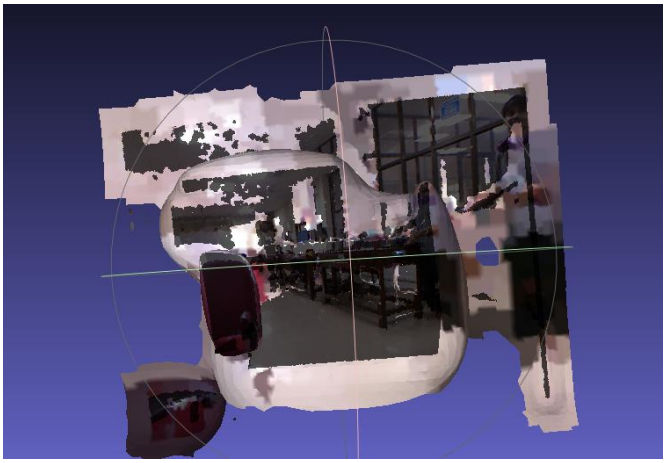
Given these challenges, I pivoted to lighter alternatives and focused on **processing the depth and RGB streams from the Kinect V1**. I explored image-based reconstruction techniques by **stitching sequential frames** and attempted to generate a composite visual representation of the environment. Additionally, I experimented with **Open3D**, a lightweight 3D data processing library, to construct **occupancy maps** and generate **point cloud reconstructions**.

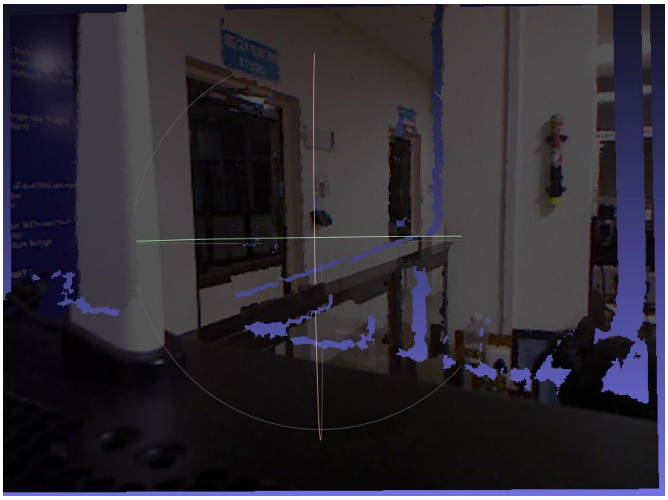
While initial attempts using a stationary Kinect only allowed me to visualize nearby objects (e.g., a hand held in front of the sensor), the resulting 3D occupancy maps confirmed successful integration and rendering. To improve spatial coverage, I began manually capturing point cloud frames at different positions and combining them to approximate a more complete spatial representation of the environment.

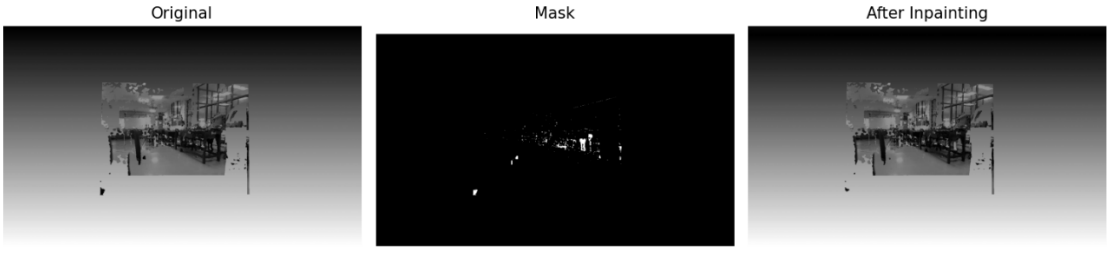
This phase provided valuable insights into the limitations of real-time mapping on low-power hardware and laid the groundwork for future optimization and hardware augmentation (e.g., potential integration of external storage or lightweight neural inference).

WEEK 3 PROGRESS









WEEK 4

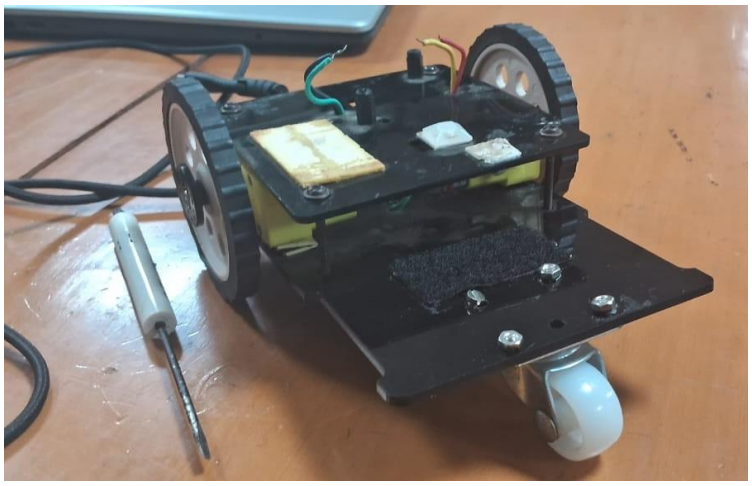
In the fourth week, I focused on **integrating the SLAM and actuation systems** developed in the earlier phases with the **physical buggy frame**. Upon receiving the chassis and **BO motors**, I mounted the motors onto the buggy and began conducting initial mobility tests on a designated test track. This phase marked the transition from modular testing to full system integration.

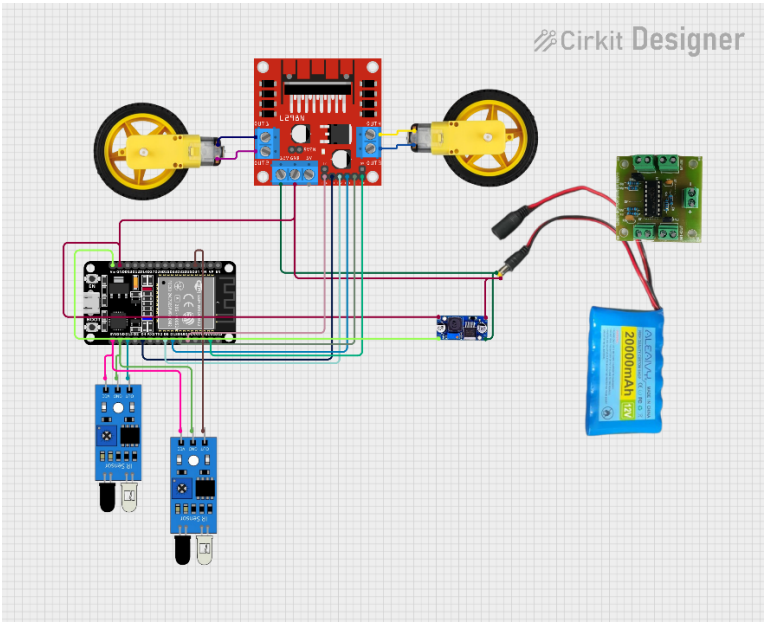
To ensure a systematic approach, I created **CAD models and mechanical layout sketches** of the buggy using **TinkerCAD**. These models helped visualize component placement and mechanical feasibility before physical assembly. Additionally, I used **CirkitDraw** software to simulate the **electrical circuit** of the buggy, enabling me to validate the wiring and power distribution logic in advance.

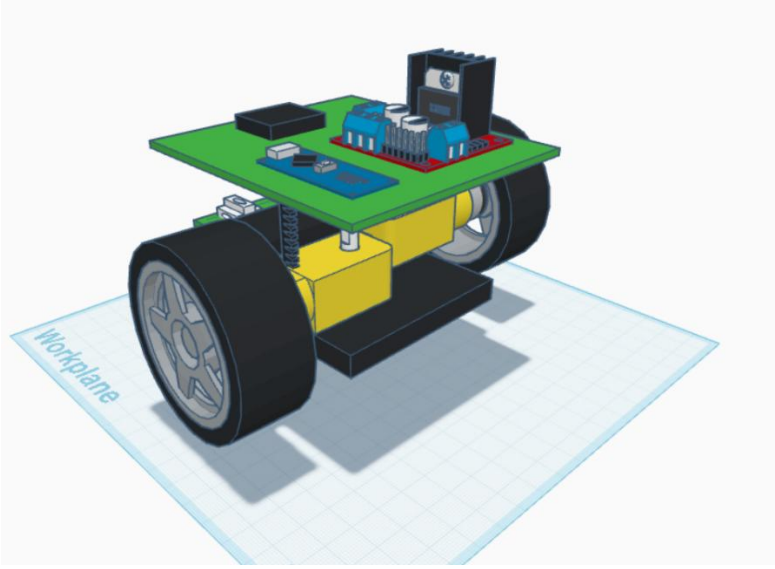
As part of the hardware integration, I studied the electrical and operational characteristics of **BO motors**, particularly their **current and voltage requirements**. This led to the careful selection and configuration of a suitable **motor driver**, specifically the **L298D full-bridge driver**. I ensured that the driver was configured to supply only the necessary current to avoid overheating or overloading the motors. I also familiarized myself with the internal working of the L298D, including its **H-bridge structure**, **PWM control support**, and **dual-channel capability** for driving two motors independently.

With all components—ESP32, motor driver, IR sensors, and power regulation modules—assembled and tested individually, I moved on to the **physical integration** phase. I designed the layout and mounting of each component on the buggy, ensuring optimal space utilization, cable management, and thermal considerations. The finalized component arrangement was documented using a **3D model**, which served as a blueprint for the physical assembly and future iterations.

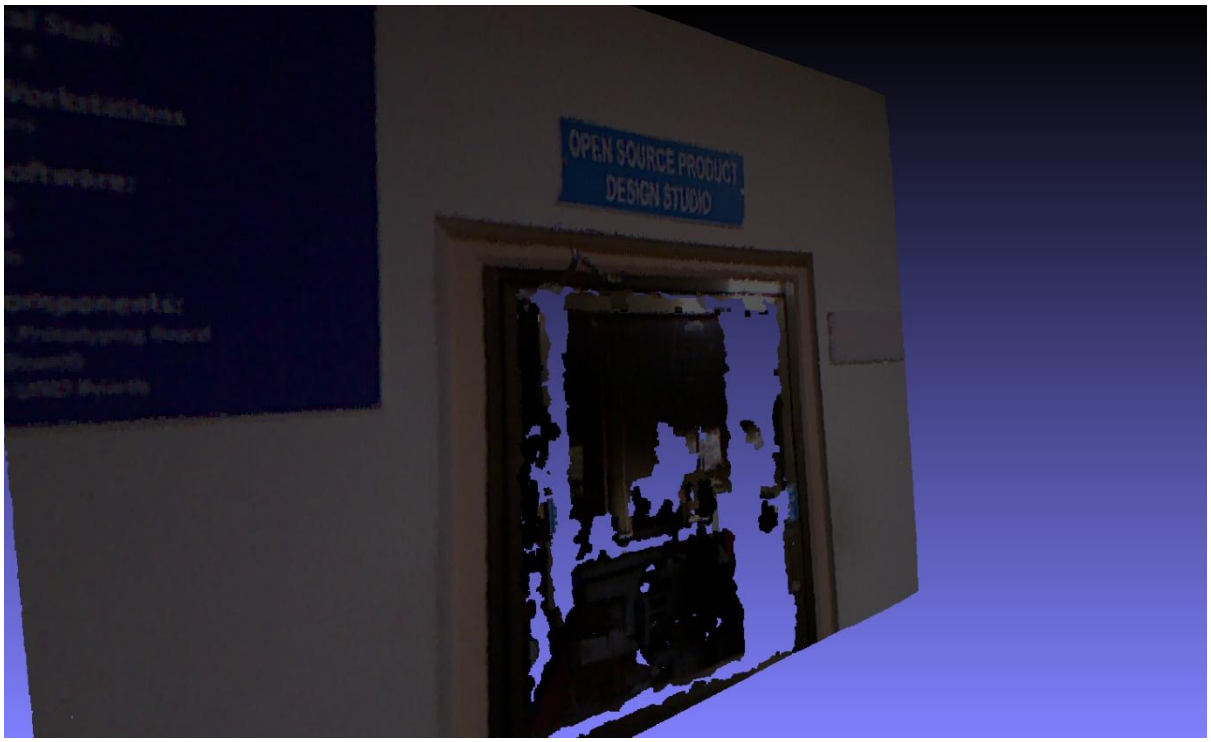
WEEK 4 PROGRESS

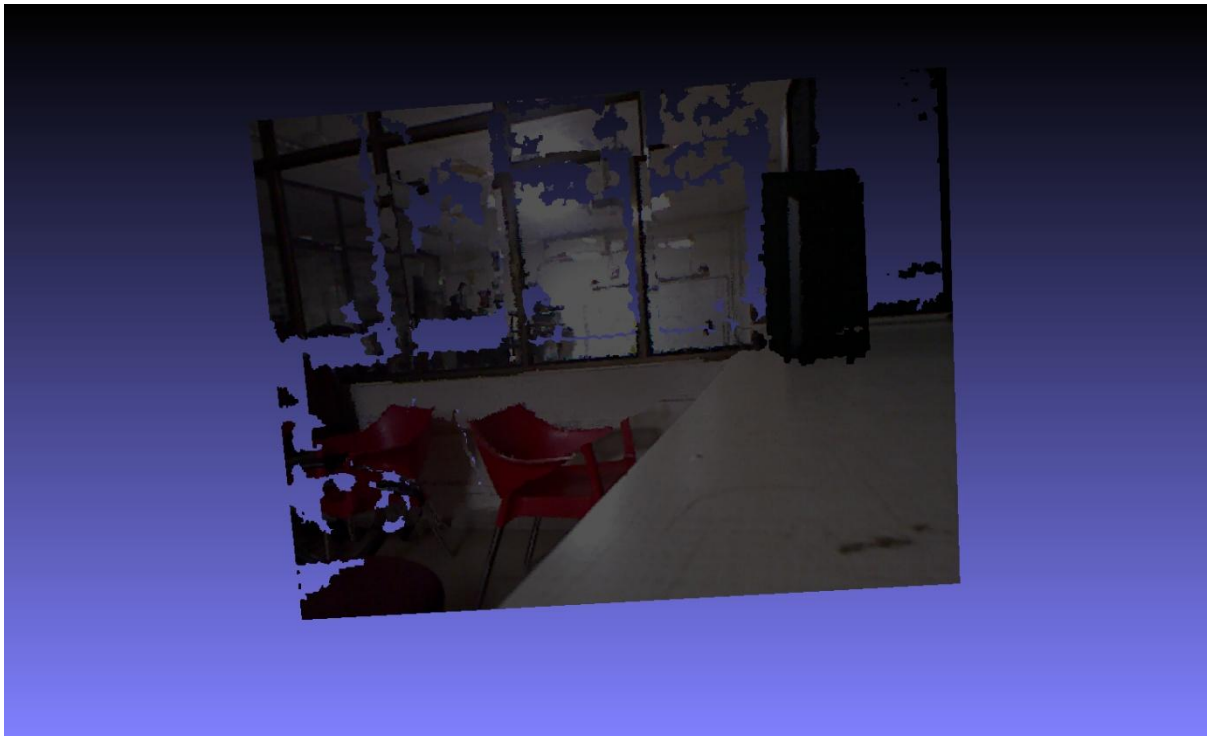


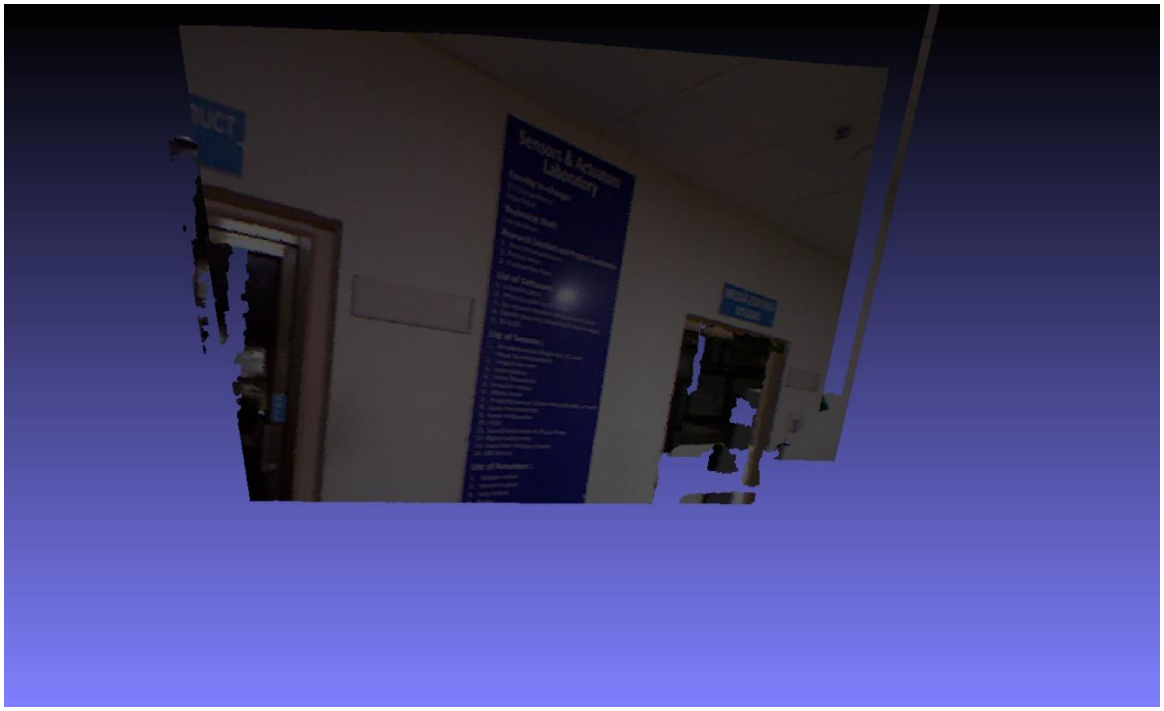


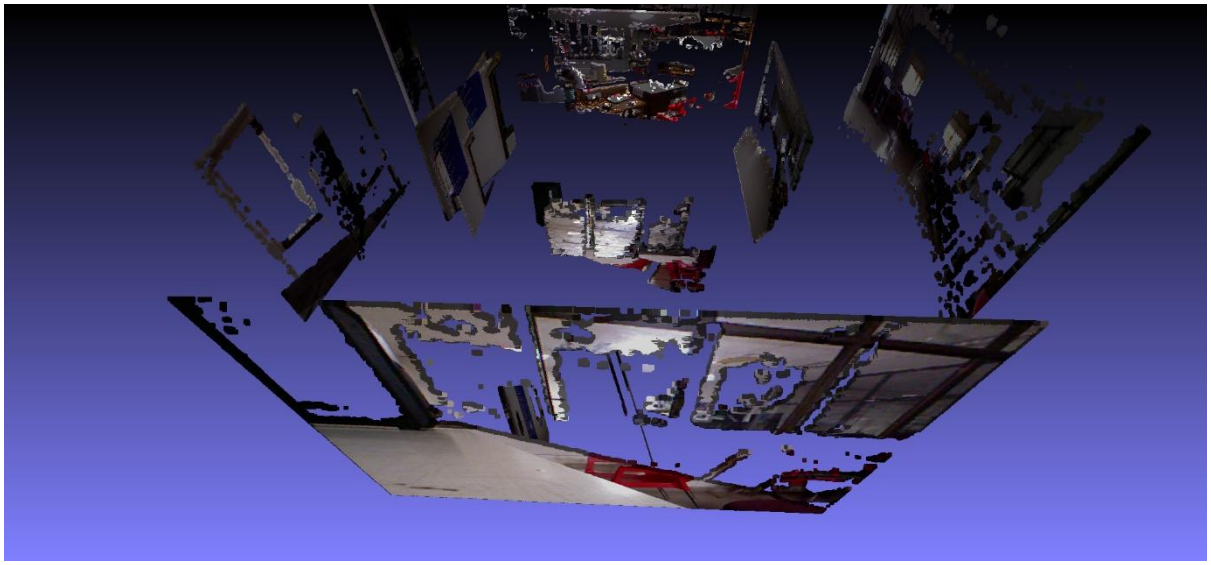


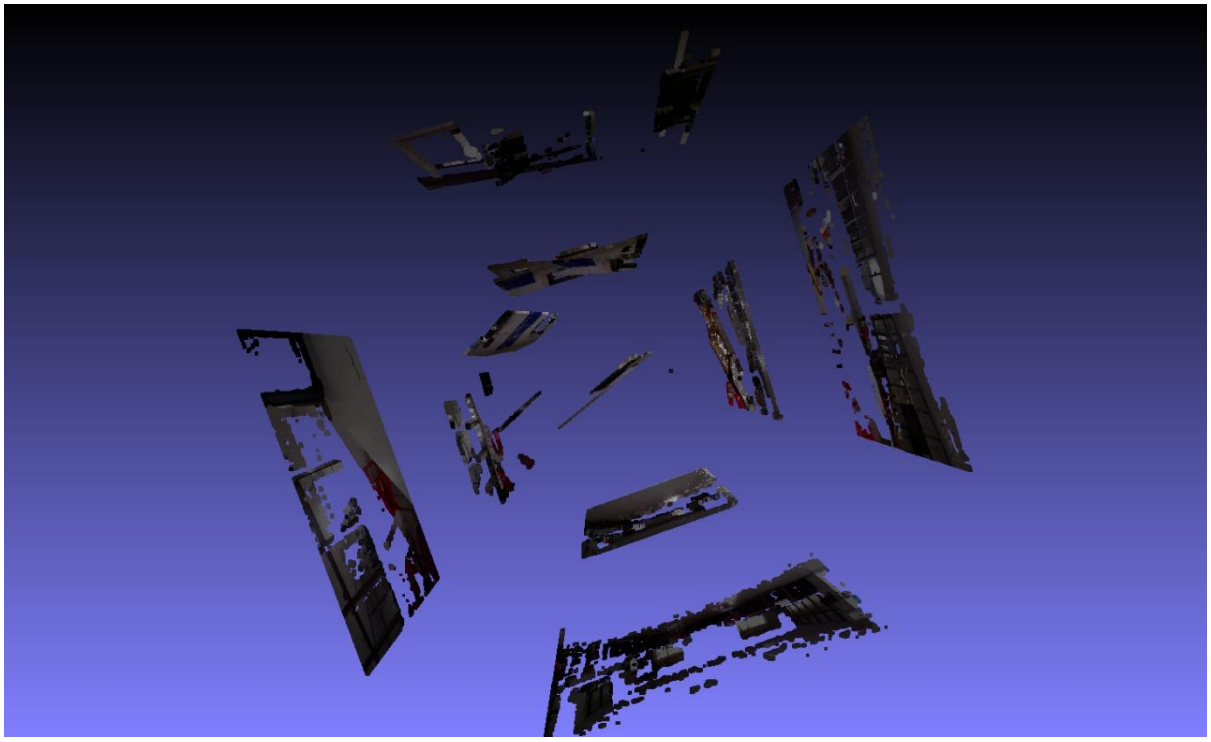
RESULT

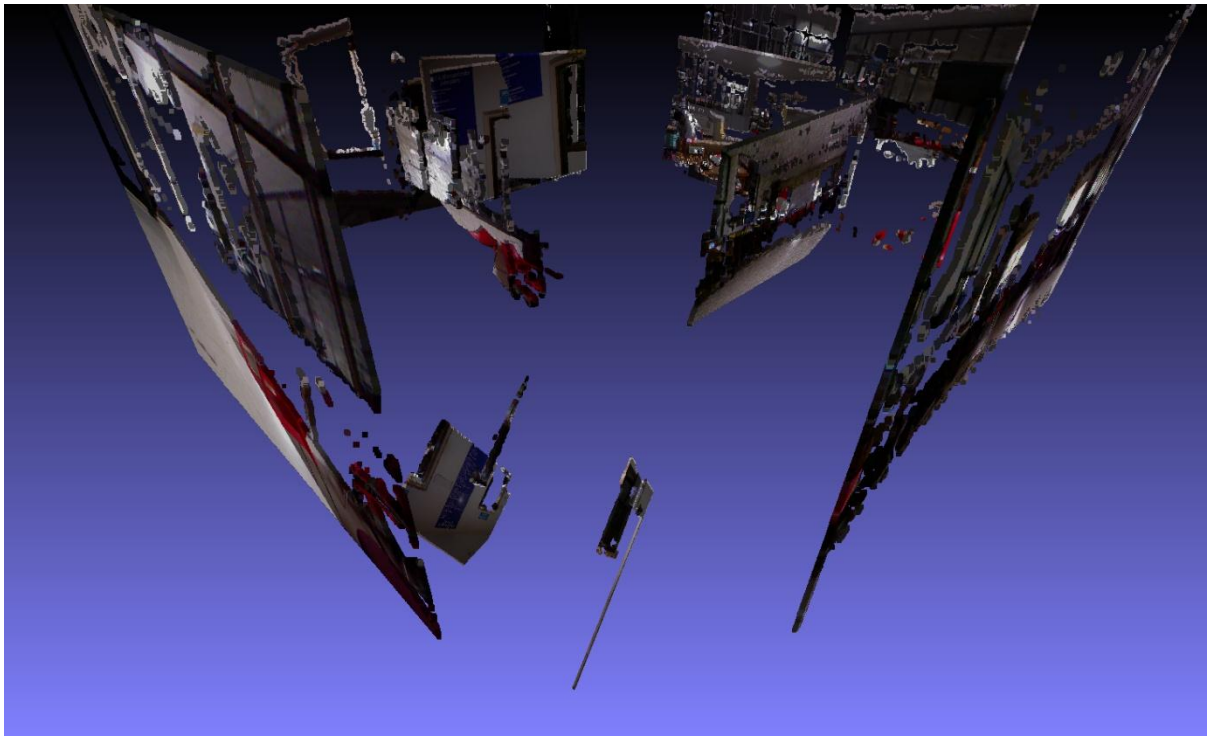












PROJECT LINKS

GITHUB

LATEST BRANCH

https://github.com/adithya1770/nitk/tree/last

BUGGY DRIVING CODES

https://github.com/adithya1770/nitk/tree/working

SOFTWARE'S USED

- ***MeshLab*** **to view point cloud files.**
- ***Python*** **to code all the components and microcontrollers.**
- ***Open3D*** **to retrieve and process the images.**
- **Ubuntu 18.04 was used as the OS on Jetson board.**
- ***Cirkit Designer*** **and *TinkerCAD* was used to design circuits and make models.**
- ***Webots*** **and *Gazebo*** **was used to simulate the vehicle.**