GAN Assisted Map Reconstruction for first responders using Sensor Fusion

Christian Jeardoe, Quinn Leboeuf,
Kyle Smejkal, Rufus Tadpatri

Concept of Operations

Revision - 3
2 December 2024

Concept of Operations
for
GAN Assisted Map Reconstruction for first responders using Sensor Fusion

TEAM <27>

APPROVED BY:

Rufus Tadpatri	*12/02/2024*
Project Leader	Date
Dr. Lusher	Date
T/A	Date

CHANGE RECORD

Rev.	Date	Originator	Approvals	Descriptions
1	09/15/2024	GAN Assisted Map Reconstruction		Draft Release
2	09/26/2024	GAN Assisted Map Reconstruction		Draft Release
3	12/02/2024	GAN Assisted Map Reconstruction		Final Edits

Table of Contents

List of Tables

List of Figures

1. Executive Summary

Emergency situations commonly occur spontaneously with little to no knowledge provided to rescuers and/or first responders. Fires, chemical incidents, or dangerous shootings are relatively common emergencies where these first responders must put themselves in danger to get an assessment of the environment in order to carry out important tasks such as retrieving someone from a dangerous area. To help aid in these types of situations and reduce physical human involvement during dangerous scenarios, we aim to give first responders the capability to receive a digital map of any enclosed environment safely.

With AI technology rapidly growing, we can use generative AI to help us create digitally mapped environments based on sensor data input. Multiple sensors will work in unison to provide very precise data for the software to build a 3D digital map. Using multiple sensors also allows the device to create a mapped environment even if visibility, heat, or other environmental factors create technical issues. A mobile robotic machine will be used to transport the sensors into wanted environments to scan data, which can then be retrieved after completion for analysis.

2. Introduction

This document explains the future operation for GAN-assisted map reconstruction using sensor fusion. The system will utilize RGB, infrared (IR), Realsense/Kinect, and ultrasonic sensor data. Each subsystem will produce individual point clouds, which will be integrated into a cohesive 3D map during the project's second stage. The final goal is a fully integrated 3D model of an enclosed environment

2.1 Purpose

The goal is to develop a system that generates 3D point clouds from various sensor inputs using Generative Adversarial Networks (GANs). By utilizing the data from RGB, infrared, Realsense/Kinect, and ultrasonic sensors, the system will create accurate 3D reconstructions of environments, which can be used in robotics, surveillance, and reconnaissance applications. The project aims to overcome the challenges

of sparse, noisy, or low-resolution sensor data by applying multiple types of data in collaboration with generative AI. Ultimately, we aim to integrate different sensors' point cloud data into a unified 3D model. The system will be designed so it can be utilized in multiple scenarios with various technical systems.

2.2 Overview

For our provided goals, our system consists of 4 subsystems which are RGB, infrared, Realsense/Kinect, and ultrasonic sensors, which each will collect and provide unique data for the system. For each sensor type, GAN models will be developed to convert raw sensor data into 3D point clouds. These individual GAN models will be designed individually for each sensor, in which the generated point clouds from all sensor data will be integrated into a 3D model. This model will be visualized in digital twin environments such as Gazebo or Unity. The system will be designed to be flexible in required technological resources to account for deployment on resource-constrained platforms like a Rpi5 or Jetson Nano. Through this integration of sensor data and advanced machine learning, the project will allow users to see unknown yet accurate 3D environments digitally reconstructed for various purposes.

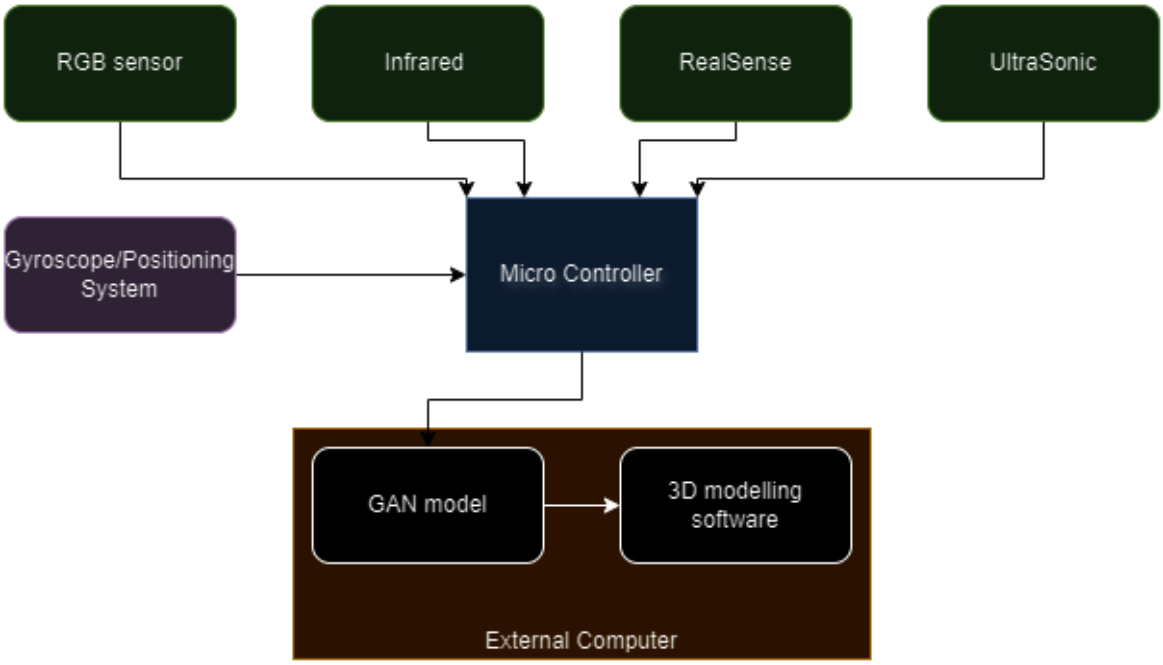

Figure 1: System Operations Overview

2.3 Referenced Documents and Citations

1. Figure 2: Wei, Y., Vosselman, G., Yang, M. Y., & University of Twente. (n.d.). Flow-based GAN for 3D Point Cloud generation. In the University of Twente [Journal-article]. https://weiyao1996.github.io/files/publications/BMVC_2022.pdf
2. Versatile and Scalable 3D RGB Point Cloud Generation from 2D Images in Unsupervised Reconstruction.
3. Point Cloud Segmentation Using RGB Drone Imagery.

3. Operating Concept

3.1 Scope

This report covers the development and integration of GAN models for different sensor types: RGB, Infrared (IR), Realsense/Kinect (RGB-D), and Ultrasonic. It includes details on responsibilities, tasks, deliverables, and the timeline for the project.

3.2 Operational Description

Tasks: Combine point clouds from various sensors into a cohesive 3D model. Visualize and optimize the integrated 3D model in digital twin environments (Gazebo/Unity). Figure 2 provides an example that relates to our task goals, but we will be visualizing an environment instead of an object.

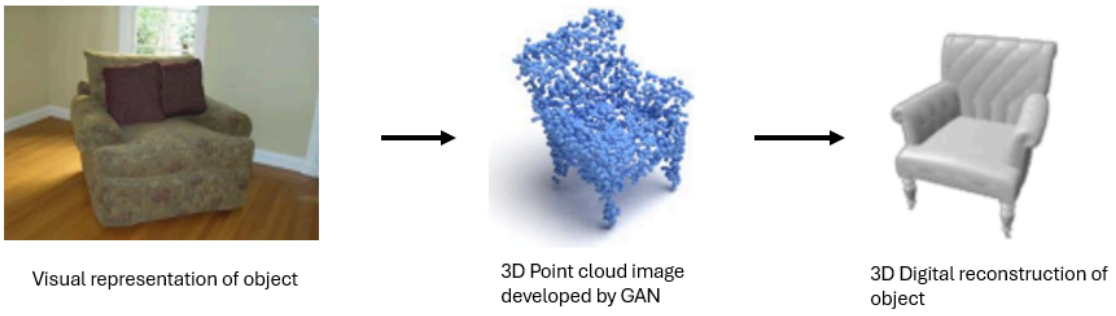

Figure 2: Example of Digital 3D Reconstruction of an Object.

3.3 System Description

3.3.1 RGB

A GAN model will be developed that converts RGB images to 3D point clouds. RGB data will be preprocessed to optimize input for the GAN. The RGB sensor allows the ability to train the GAN model to convert from RGB to point cloud in order to integrate into the final 3D model.

3.3.1.1 Existing Technologies

Currently, there exists a technology that utilizes CNN layers and diffusion-denoising approaches. Convolutional neural networks use several layers to maintain relevant information without causing a storage issue.

3.3.1.2 Alternative Methods

Our RGB will use machine learning algorithms to train the model in order to develop a GAN model. This model will be trained to preprocess RGB data; this will help with processing speed as well as the reliability of the model.

3.3.2 Infrared

The Infrared sensor will capture thermal imaging of the environment in order to train the GAN model to create a 3D point cloud of said environment. This sensor will be able to accurately capture data in areas that the other sensors may not be able to perform as well, such as low-light environments.

3.3.2.1 Existing Technologies

Infrared sensors are used quite often to create point clouds using LiDAR. The infrared sensor in these systems is used to calibrate the positioning of RGB points when generating a point cloud in an environment. They are also useful for making 3-D heat maps of large environments.

3.3.2.2 Alternative Methods

The Infrared sensor in this case will be used to generate a point cloud using our GAN model which will be used later to combine with the model output of other sensors. This model will be trained on openly available infrared sensor data as well as data that will be captured using the sensor.

3.3.3 Realsense/Kinect

The RealSense/Kinect camera plays a pivotal role in capturing depth information, and when combined with its RGB imaging capabilities, it can independently generate a detailed 3D map of a room. This dataset is essential for the GAN model, as it enhances the depth data, enabling the creation of high-resolution point clouds for accurate and refined 3D reconstructions

3.3.3.1 Existing Technologies

While Intel Realsense cameras are not widely used in common settings, researchers have found great use for them in tracking body movements for both human and animal applications. The Microsoft Kinect camera is a sensor that was sold along with the Xbox 360 and Xbox One as a tool for hands-free controls and gaming applications. Both of these devices are mainly used for tracking the movement of an object.

3.3.3.2 Alternative Methods

Unlike the common uses for these sensors, we won’t be tracking the motion of objects, rather, the sensor’s depth-sensing camera will be used to determine the distance of objects to map the RGB camera input better.

3.3.4 Ultrasonic

The ultrasonic sensor collects data that provides information on the distance between objects using sound waves. While there will be extreme challenges in getting detailed information from this type of sensor in complex environments, it can be used extensively to confirm the accuracy

of the developed information from other sensors such as a room/object's size, length, etc. Ultrasonic sensors do not need visibility, providing a solution to environments covered in smoke or dark.

3.3.4.1 Existing Technologies

Ultrasonic sensors are used in a wide variety of applications in today's world. This includes obstacle detection for moving vehicles, liquid level measurements within containers such as wastewater treatment plants and water tanks, and even automatic security/door operations. Similar applications to this project include using ultrasonic sensors to map inaccessible areas and generate 3D environments. In regards to GAN model applications, these models are mostly used for image generation and 3D object generation.

3.3.4.2 Alternative Methods

Gan models often generate brand-new, unique images from limited data sets. While using the ultrasonic sensor to collect various data points from any surrounding environment, the GAN model must be trained to recognize differences between objects, walls, and other surfaces from simple distance measurements.

3.4 Modes of Operations

3.4.1 Reconnaissance

The only mode of operation for this system is to perform reconnaissance in an emergency situation.

3.5 Users

Primary users would be first responders. The intended purpose of this system is to assist in an emergency situation, so the user would not need prior knowledge other than basic robotic control. The user would maneuver the robot around the room, capturing imagery using the sensors of the robot.

4. Scenarios

4.1. Search and Rescue

The primary scenario for which the robot is being designed is search and rescue missions. These operations often occur in hazardous or inaccessible areas, such as collapsed buildings, dense forests, or areas affected by natural disasters. The GAN robot, equipped with various sensors, can navigate these dangerous environments and generate real-time, high-resolution maps. The robot can use its GAN model to reconstruct partially occluded regions and enhance imaging data, allowing rescue teams to detect and locate victims faster without putting themselves at risk.

4.2. Cave Exploration

Exploring cave systems is challenging due to the lack of natural light and complex, often unstable, terrain. The GAN robot can utilize depth sensors, infrared cameras, and LiDAR to generate detailed 3D maps of the cave environment. Its GAN model can enhance low-resolution or incomplete sensor data to create a comprehensive view of the cave, helping explorers or researchers navigate safely while minimizing the risk of getting lost or encountering dangerous areas.

4.3. Sea-floor Mapping

Mapping the seafloor is crucial for marine research, underwater construction, and locating shipwrecks. Traditional methods using sonar or cameras can result in noisy or incomplete data due to water conditions. The GAN robot, equipped with underwater sensors, can enhance this data in real time, providing clearer and more accurate images of the sea floor.

4.4. Healthcare

While our GAN model is being developed to be used on robots, it is not limited to that use. For example, in healthcare, the GAN model can be used to improve medical imaging techniques. By generating high-resolution images from noisy or low-quality sensor data (such as

MRIs, CT scans, or ultrasounds), GANs can help doctors make more accurate diagnoses.

5. Analysis

5.1 Summary of Proposed Improvements

A robot with these sensors can capture a 3D model of the environment more accurately with sensors backed by machine learning to create the 3D environment while being able to fill the gap caused by environmental factors for one or more sensors.

5.2 Constraints

Many different factors can lead to constraints in the operation of the system, as outlined in the table below.

Constraints	Reasoning
1) Microcontroller	The GAN model will be running on a microcontroller which limits the computation power of the model.
2) Sensor Resolution	Sensor resolution can greatly affect the quality of the 3D model and different sensors can capture data in different ways.
3) Time	The GAN model will be limited due to the limited training data and training times due to deadlines.

Table 3: List of Design Constraints

5.3 Impact

This technology could have a great impact for mankind, as it would allow us to more accurately map areas before humans need to enter. The best example of this would be for first responders getting a 3D map of an environment and its hazards before people have to go into said environment. Another positive impact would be mapping and charting

areas that are unknown or unsafe for humans such as caves or the ocean floor.

5.4 Risks and Mitigation

1. **Model Accuracy**: GAN models may not achieve the desired accuracy.
    1.1. *Mitigation*: Iterative testing and validation; explore alternative architectures.
2. **Integration Issues**: Difficulty in merging point clouds from different sensors.
    2.1. *Mitigation*: Early and continuous integration testing; effective communication within the team.
3. **Visualization Challenges**: Inaccurate or inefficient visualization of the 3D model.
    3.1. *Mitigation*: Regular feedback sessions; thorough testing in Gazebo/Unity.

5.5 Alternatives

There are other projects that are using sensor fusion and machine learning to create 3D point cloud maps, like Nvidia. However, these projects mainly focus on making a model based on RGB and Lidar data rather than the 4 sensors used here.

Conclusion

This ConOps provides a comprehensive framework for the successful development and integration of GAN models for generating 3D point clouds from various sensor types. By adhering to this plan, the team will produce a unified 3D model and demonstrate its capabilities in a digital twin environment, contributing valuable insights and tools for 3D reconstruction and visualization

GAN Assisted Map Reconstruction for first responders using Sensor Fusion

Christian Jeardoe, Quinn Leboeuf,
Kyle Smejkal, Rufus Tadpatri

INTERFACE CONTROL DOCUMENT

REVISION – 2
4 December 2024

Interface Control Document
FOR
GAN Assisted Map Reconstruction for first responders using Sensor Fusion

Approved by:

Rufus Tadpatri	*12/04/2024*
Project Leader	Date
John Lusher II, P.E.	Date
T/A	Date

Change Record

Rev.	Date	Originator	Approvals	Description
1	10/01/2024	GAN Assisted Map Reconstruction		Draft Release
2	12/04/2024	GAN Assisted Map Reconstruction		Final Version

Table of Contents

List of Tables

1. Overview

The Interface Control Document (ICD) for the GAN-assisted sensor fusion map reconstruction project for first responders will outline how components from the Concept of Operations (ConOps) and Functional System Requirements (FSR) are developed and integrated. It will cover the GAN architecture, sensor input/output formats, preprocessing steps, computational needs (GPU/CPU), and model deployment workflow. Additionally, it will detail the interaction between GAN subsystems, including data flow, training processes, and evaluation metrics, ensuring alignment with the goals and requirements in the FSR and ConOps.

2. References and Definitions

2.1. References

OpenCV Documentation (v4.5.5)
Computer Vision Library Reference
Released Apr 2021

PyTorch Framework Documentation (v2.0)
Machine Learning Framework for Neural Networks
Updated Mar 2024

Intel RealSense SDK 2.0
Software Development Kit for RealSense Cameras
Updated Oct 2024

PyRealSense2
PyRealSense2 Python Wrapper Documentation
Updated Oct 2024

Waveshare MLX90640-D55 Thermal Camera Overview
Documentation for thermal camera

PlayStation 12240 PS3 Eye Camera Overview
Documentation for RGB camera

Code Laboratories Eye Platform Driver
Software for RGB camera

2.2. Definitions

GAN	Generative Adversarial Network
FPS	Frames Per Second
SDK	Software Development Kit
CL	Code Laboratories
I2C	Inter-Integrated Circuit
PS	Play Station
V	Volts
A	Amps
mA	MilliAmps

3. Physical Interface

3.1. Weight

Due to this project's inherent software nature, there are only a few components that are being physically used.

Table 1: Weights of Components Used in Project

Component	Weight
Intel Realsense Camera	2.54 oz
HP OMEN Laptop	4.68 lbs
MLX90640 Thermal Camera	0.2 oz
Raspberry Pi 3	1.48 oz
Arduino Uno R3	0.88 oz
HC-SR04 Ultrasonic Sensor	0.3 oz
Dell XPS Laptop	2.6 lbs
PS3 Move Eye Camera (RGB Sensor)	4.79 oz

3.2. Dimensions

Table 2: Dimensions of Components Used in Project

Component	Diameter	Length	Width	Height
Intel Realsense Camera	N/A	3.54 “	0.98 “	0.98 “
HP OMEN Laptop	N/A	15.07 “	9.67 “	0.61 “
Intel Realsense USB 3.1 Cable	0.20 “	2.5 “	N/A	N/A
Raspberry Pi 3	N/A	3.35 “	2.2 “	0.8 “
MLX90640 Thermal Camera	N/A	1.1 “	0.63 “	0.47 “

Arduino Uno R3	N/A	2.7 “	2.1 “	N/A
HC-SR04 Ultrasonic Sensor	N/A	1.77 “	0.787 “	0.591 “
Dell Precision 5540	N/A	9.26 “	14.06 “	0.44 “
Dell XPS Laptop	N/A	11.62 ”	7.84 ”	0.60 ”
PS3 Move Eye Camera (RGB Sensor)	N/A	3.31 ”	2.64 ”	2.24 ”

3.3. Mounting Locations

3.3.1. Intel RealSense D435 Mounting Points

The Intel Realsense D435 Camera has two mounting points, one for a single 1/4-20 UNC thread and two for an M3 thread mount.

3.3.2. MLX90640 Thermal Camera Mounting Points

The MLX90640 Thermal Camera has two points, both are 2mm located on the camera side of the PCB

3.3.3. HC-SR04 Ultrasonic Sensor Mounting

The HC-SR04 has a separate mounting device designed specifically for a servo motor. This will be mounted in conjunction with an RGB camera for 360 degree room scanning to collect model training data as well as input data.

4. Thermal Interface

Our project has very few heating concerns. The cameras will heat up when in use, noticeable to the touch, but not to the point of failure. No additional cooling or heating requirements will be needed at this stage of the project.

5. Electrical Interface

5.1. Primary Input Power

5.1.1. Primary Power Input for RealSense Subsystem

The RealSense subsystem will draw power directly from the USB 3.1 port on the HP Omen laptop. The laptop itself will be powered either by its internal 12V battery for portability or through a standard wall outlet using its 120W charger for extended operation.

5.1.2. Primary Power Input for Thermal Camera Subsystem

The MLX90640 operates between 3.3V and 5V with <23 mA which is provided by pin 4 on the Raspberry Pi 3. The Raspberry Pi itself requires 5V and 2.5A which can be provided via Micro USB.

5.1.3. Primary Power Input for RGB Camera Subsystem

The RGB subsystem draws power directly from the Dell XPS laptop through the USB-C power via a USB-C to USB adapter. The laptop itself will be powered either by its internal 12V battery for portability or through a standard wall outlet using its 120W charger for extended operation.

5.1.4. Primary Power Input for Ultrasonic Subsystem

The HC-SR04 ultrasonic sensor utilizes a 5 volt power supply from the Arduino itself (which draws its own power from a precision 5540 Dell laptop), as well as a current of 15 mA. The Arduino is connected to a computer through a 3M Arduino UNO USB data sync cable.

5.2. Signal Interfaces

5.2.1. Signal Interface for RealSense Subsystem

The Intel RealSense D435 camera is connected to an HP Omen laptop through a USB 3.1 port, ensuring high-speed data transfer rates required for real-time depth map and RGB image acquisition. The USB 3.1 interface supports a data transfer rate of up to 10 Gbps, enabling the subsystem to handle the bandwidth demands of high-resolution depth and color streams. The stability and throughput of the USB 3.1 connection are critical for achieving seamless sensor data fusion and processing during operation.

5.2.2. Signal Interface for Ultrasonic Subsystem

The pin configuration and other electrical characteristics are shown in the table below. The TRIG signal requires a 10-microsecond high pulse to begin measuring. The Sensor then emits 40 kHz ultrasonic bursts upon receiving the pulse. The ECHO pin outputs a high pulse, used to calculate the distance in centimeters. These pins are configured via an Arduino IDE connected through the USB 2.0 cable type A to B Male for Arduino.

Table 3: Ultrasonic Sensor Pin Interface

Pin	Number	Description
1	Vcc	Power supply
2	TRIG	Trigger pin (input)
3	ECHO	Echo pin (output)
4	GND	Ground Connection

5.3. Video Interfaces

5.3.1. Video Interface for RealSense Subsystem

The RealSense camera's video feed (both depth and RGB) is displayed in real-time on the HP Omen via the RealSense SDK 2.0. The video feed allows you to see what exactly the camera is seeing, which helps with testing. Along with this, via OpenCV, you can display the generated point cloud in a 3D environment where it can be manipulated.

5.3.2. Video Interface for RGB Subsystem

The RGB camera's video feed is displayed in real-time on the Dell XPS laptop via the CL-Eye Test program. The video feed allows you to see what exactly the camera is seeing, which helps with testing. Along with this, via OpenCV, you can display the generated point cloud in a 3D environment where it can be manipulated.

5.4. User Control Interface

5.4.1. User Control Interface for RealSense Subsystem

The UI for the RealSense subsystem is currently all within the Windows command prompt (CMD). The training program is all automated; it just needs to be executed, same for the inference program. The data collection program requires the press of either the 's' key to save an image or the press of the 'q' to quit the program. The point cloud generating program requires the press of the 'e' key to save a .PLY file and a press of the 'q' key to quit the program. Finally, the point cloud displayer is also fully automated, it just needs to be executed after the generator.

5.4.2. User Control Interface for Thermal Camera Subsystem

The UI for the Thermal Camera Subsystem is all through a Python program. Upon starting the program, the camera boots up and refreshes at a rate set in the program between 0.5 and 64 Hz. The data is formatted as an array and needs to be reformatted into the dimensions of the camera and saved. The program captures thermal images at the set refresh rate until the 'q' key is pressed. The images are saved as '.png' and transferred to the computer with the GAN model via a USB storage device.

5.4.3. User Control Interface for RGB Camera Subsystem

The UI for the RGB camera runs through a Microsoft Visual Studios python script. Upon execution of the program, the user will set up the sensor and press the ‘s’ key to capture an image or press the ‘q’ key to quit the program. Once an image has been captured, the point cloud generation will begin automatically. After about 10-15 seconds, the point cloud will open up and the user can use the mouse to maneuver around the 3d environment displayed.

5.4.4. User Control Interface for Ultrasonic Subsystem

The UI for the ultrasonic sensor uses Arduino API to design the triggers for pins and collect consecutive distance measurements. Using the Arduino IDE, the code for a similar ultrasonic sensor is already given, which can be edited to accommodate for the HC-SR04 pins. Distance is rapidly calculated through the formula:

$$Distance\ (cm)\ =\ (Pulse\ Width\ (\mu s) \times 0.034)\ /\ 2$$

The result will be displayed using Serial.print(). A delay of at least 60 milliseconds between measurements will be added for stability.

5.5. Microcontroller Pin Interface

5.5.1. Microcontroller Pin Interface for the Thermal Camera Subsystem

The MLX90640 Thermal Camera has 4 pins connected to the Raspberry PI 3. VCC, GND pin for the power supply, VCC connected to the control 3.3V or 5V power supply, GND corresponds to the connection of the GND. SDA is the data pin of I2C, connected to the GPIO 8 pin 3 of the Raspberry PI 3, without an external pull-up resistor. SCL is the clock pin of I2C, connected to the GPIO 9 pin 5 of the Raspberry PI 3, without an external pull-up resistor.

5.5.2. Microcontroller Pin Interface for the Ultrasonic Sensor Subsystem

The Pin interface used on the Arduino Uno is based on the ultrasonic sensor pin interface shown in table 3. There are 4 pins used on the microcontroller. The 5V and GND pins are used to power the ultrasonic sensor, and pins 9 and 10 were used to connect the sensor TRIG and ECHO regions (though, any pin 2 through 13 can be used for this connection).

6. Communications / Device Interface Protocols

6.1. Host Device

6.1.1. Host Device for the RealSense Subsystem

The host device for the project is an HP Omen laptop equipped with USB 3.1 ports, an AMD 2600X CPU, and an Nvidia 960M GPU. The host runs Windows 10 and interfaces with the Intel RealSense D435 camera through the RealSense SDK 2.0, specifically the pyrealsense2 wrapper. Communication between the host and the device is managed using USB 3.1 protocols, ensuring high-speed data transfer and efficient utilization of hardware resources. The host also manages data preprocessing, GAN model inference, 3D point cloud generation, and performance evaluation.

6.1.2. Host Device for the RGB Subsystem

The host device for the RGB subsystem is a Dell XPS laptop. The host runs Windows 10 and interfaces with the PS3 Move Eye Camera through the CL-Eye Test program. Communication between the host and the device is managed through the USB connection, ensuring high-speed data transfer and efficient utilization of hardware resources. The host also manages data preprocessing, GAN model inference, 3D point cloud generation, and performance evaluation.

6.1.3. Host Device for Thermal Camera Subsystem

The host device for the Thermal Camera subsystem is a Dell Precision 5520 laptop. The host runs Windows 10 and interfaces with the MLX90640 thermal camera through a Raspberry PI 3.

6.1.4. Host Device for Ultrasonic Sensor Subsystem

A Dell Precision 5540 is used in conjunction with the Arduino Uno to host the Ultrasonic sensor data collection. This device also is used for the use of the GAN model, including training and data processing. Device programs include VScode for GAN model development and the Arduino IDE for data collection.

6.2. Video Interface

6.2.1. Video Interface for the RealSense Subsystem

The video interface between the RealSense D435 camera and the host laptop uses a USB 3.1 connection. The interface streams synchronized depth and RGB video data at resolutions of up to 1280x720 for depth and 1920x1080 for RGB, although for our purposes this has been turned down to 640x480 for both. The video feed is captured at 30 FPS for both depth and RGB. Video data is encoded using the camera's internal processing and sent as raw image frames, which are decoded on the host.

6.2.2. Video Interface for the RGB Subsystem

The video interface for the PS3 Move Eye camera has a resolution at 640x480 pixels. The video feed is captured at 60 frames per second. Video data is encoded using the camera's internal processing and sent as raw image frames, which are decoded on the host.

6.3. Device Peripheral Interface

The USB 3.1 Gen 1 protocol is used for the RealSense camera. The host interacts with these peripherals through the RealSense SDK, which abstracts the low-level protocols and provides high-level APIs for controlling camera parameters and retrieving depth and RGB streams. The communication protocol of MLX90640 Thermal Camera is I2C, which supports I2C high-speed mode (up to 1MHz), and can only be used as a slave device on the I2C bus. The HC-SR04 ultrasonic sensor interfaces with a microcontroller via a digital output TRIG pin for initiating measurements and a digital input ECHO pin for receiving the reflected signal duration, powered by 5V and GND.

GAN Assisted Map Reconstruction for first responders using Sensor Fusion

Christian Jeardoe, Quinn Leboeuf,
Kyle Smejkal, Rufus Tadpatri

FUNCTIONAL SYSTEM REQUIREMENTS

REVISION – 2
4 December 2024

FUNCTIONAL SYSTEM REQUIREMENTS

FOR

GAN Assisted Map Reconstruction for first responders using Sensor Fusion

APPROVED BY:

Rufus Tadpatri *12/04/2024*

Project Leader Date

_____

John Lusher, P.E. Date

_____

T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
1	10/01/2024	GAN Assisted Map Reconstruction		Draft Release
2	12/04/2024	GAN Assisted Map Reconstruction		Final Version

Table of Contents

List of Tables

List of Figures

1. Introduction

1.1. Purpose and Scope

The purpose of this project is to develop a GAN-assisted system for sensor fusion and depth map reconstruction, designed to enhance situational awareness for first responders in critical environments. Leveraging data captured by RealSense, RGB, Infrared, and Ultrasonic sensors the system shall generate accurate, high-quality virtual twins of the environment by reducing noise and filling in missing data, addressing limitations in raw sensor output. The project should encompass the integration of several GAN models and preprocessing pipelines to deliver real-time or near-real-time point cloud generation and enhancement. The scope of this project includes designing and implementing the communication protocols, data processing workflows, and evaluation metrics necessary to ensure the system operates reliably in both controlled and dynamic scenarios. This solution is intended to support applications such as navigation in low-visibility environments, obstacle detection, and spatial mapping, contributing to the safety and efficiency of first responder operations.

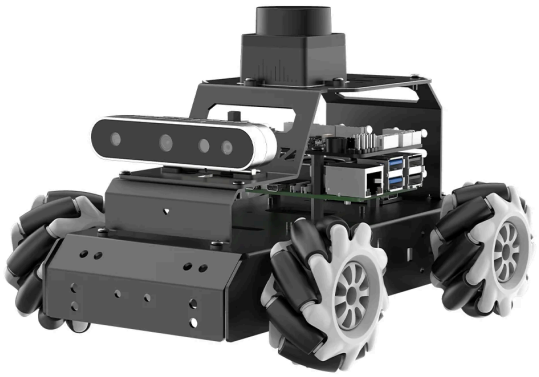

Figure 1: Project Concept Image

1.2. Responsibility and Change Authority

Our team leader, Rufus Tadpatri, is charged with ensuring all specifications of the system are met. Any changes to the specifications or deliverables of the project must be approved by the team leader, Rufus Tadpatri, and our sponsor/TA, Swarnabha Roy.

2. Applicable and Reference Documents

2.1. Applicable Documents

The following documents, of the exact issue and revision shown, form a part of this specification to the extent specified herein:

Table 1: Applicable Documents

Document Number	Revision/Release Date	Document Title
[1]	12/10/2023	Versatile and Scalable 3D RGB Point Cloud Generation from 2D Images in Unsupervised Reconstruction
[2]	2020	Point Cloud Segmentation Using RGB Drone Imagery

2.2. Reference Documents

The following documents are reference documents utilized in the development of this specification. These documents do not form a part of this specification and are not controlled by their reference herein:

Table 2: Reference Documents

Document Number	Revision/Release Date	Document Title
OpenCV-4.5.5	April 2021	OpenCV Documentation v4.5.5
PyTorch-2.0	March 2024	PyTorch Framework Documentation v2.0
IntelRS-SDK-2.0	October 2024	Intel RealSense SDK 2.0
PyRealSense2	October 2024	PyRealSense2 Python Wrapper Documentation
MLX90640 Wiki	July 2024	Waveshare MLX90640-55 Thermal Camera overview

2.3. Order of Precedence

In the event of a conflict between the text of this specification and an applicable document cited herein, the text of this specification takes precedence without any exceptions.

All specifications, standards, exhibits, drawings or other documents that are invoked as “applicable” in this specification are incorporated as cited. All documents that are referred to within an applicable report are considered to be for guidance and information only, except ICDs that have their relevant documents considered to be incorporated as cited.

3. Requirements

3.1. System Definition

The finished product will be a robot that will maneuver around a room/environment. Four individual subsystems with four different sensors will scan the surroundings and generate a 3D point cloud. The four different sensors are an RGB sensor, an infrared sensor, a RealSense sensor, and an ultrasonic sensor.

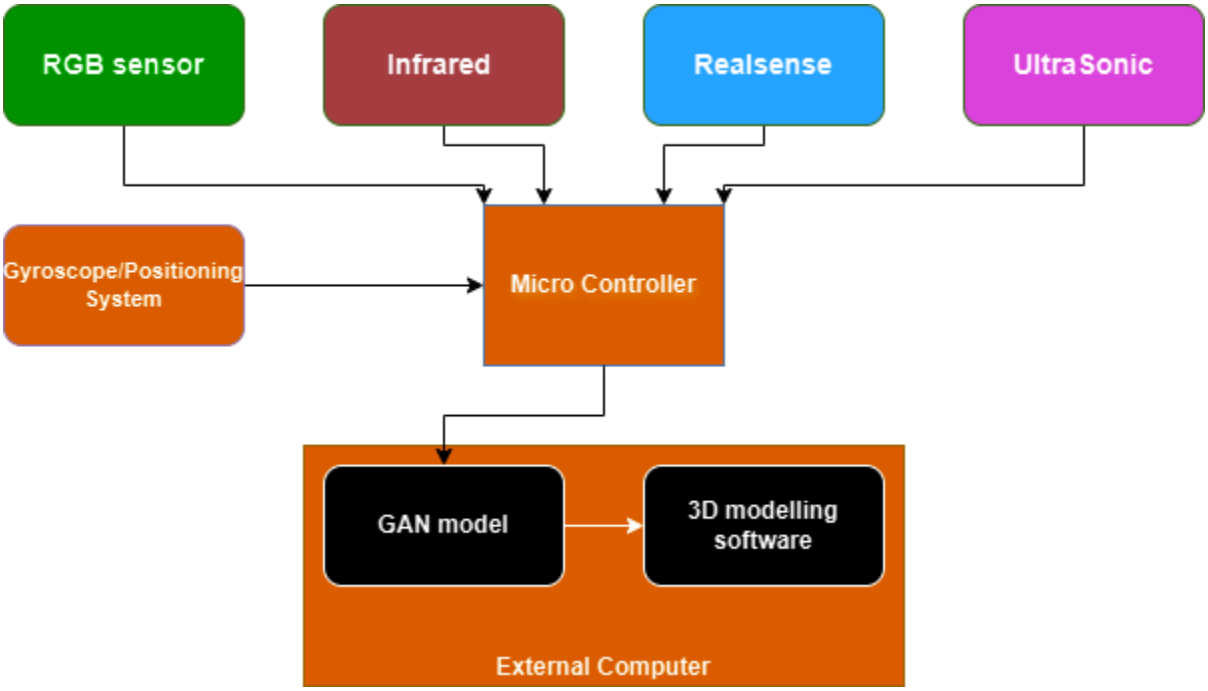

Figure 2. Block Diagram of System

This block diagram illustrates the high-level structure of how each subsystem is interconnected. Each sensor will provide its input to a microcontroller, along with an inertial measurement unit (IMU). The microcontroller will organize the data and transmit it to an external computer. The computer will run each subsystem's individual GAN model, processing the data streams. The final output will then be fed into 3D modeling software to visualize the 3D twin environment being created.

3.2. Characteristics

3.2.1. Functional / Performance Requirements

3.2.1.1. Requirements

3.2.1.1.1. Robot Maneuvering

The robot that will be used to carry the sensors should have accurate navigation for merging of generated point cloud images. The robot must reliably avoid obstacles and be capable of movement on various terrain. Current operable terrains mainly include those found within indoor environments (hardwood, carpet, etc.). Future improved terrain maneuverability could include grass/dirt, jagged rock, and sand.

3.2.1.1.2. Sensor Fusion

Sensors should work together to output highly accurate 3D images of the scanned environment. All sensors should not only work individually, but also all be capable of providing data for 3D image improvement in conjunction with each other. The system should also be capable of understanding when specific sensors are not working properly. This can be a malfunction in the sensor itself, or the sensor is in an environment where it cannot properly collect data (RGB sensor cannot collect data in a dark room). Remaining sensors that are still functional in such situations will still be capable of working together to develop 3D maps without needing the non-functional sensor data.

3.2.1.1.3. 3D Image Output

The system should be capable of creating a 3D map no matter the type of environment or conditions. Whether just 1 or all 4 sensors are working, a 3D image output should always be produced. This image will generate in real time and be improved as more data is collected from the sensors that are moving around some enclosed area.

3.2.1.2. Distance Requirements

The system as a whole can only accurately create a point cloud of objects within 3 meters due to the limitations of each individual sensor.

> *Rationale: The optimal range for the depth sensor of the RealSense camera is .3 to 3 meters. The optimal range for the MLX90640 thermal camera is .5 to 5 meters. The optimal range for the Ultrasonic sensor is around 4 meters.*

3.2.2. Electrical Characteristics

3.2.1.3. Inputs

a. All input data for the GAN model is received through each of the four subsystem sensors.

Rationale: GAN models are developed to accommodate data received specifically from their corresponding sensors

3.2.1.3.1 External Commands

Commands come in from each subsystem's control system, whether being a microcontroller or a laptop device.

Rationale: GAN models are developed to accommodate data received specifically form their corresponding sensors

3.2.1.4. Outputs

3.2.1.4.1 Data Output

Outputs for each subsystem will be a 3D point cloud. The point cloud will open on each laptop as an openCV.

Rationale: Each GAN model should be capable of creating its own point cloud without help from other sensor's data or their corresponding GAN model outputs.

3.2.1.4.2 Raw Video Output

The project shall include a raw video interface to support external recording.

Rationale: Raw video helps ensure the system is working properly and sensor data aligns with corresponding RGB images for the environment.

3.2.2. Failure Propagation

The system shall not allow propagation of faults beyond the project specification interface.

3.2.1.5. Failure Detection, Isolation, and Recovery (FDIR)

Failure detection for each individual system is required for the system to function properly. Each sensor must know when another sensor is inoperable, and accommodate for lack of said sensor data. Failure of sensors is expected based on environmental factors, so recovery is not necessarily needed in these situations. Sensors are individually isolated if failing, and if only one sensor is remaining, the remaining sensor is isolated to be used as the primary data retriever for the GAN model and the corresponding 3D image generation.

4. Support Requirements

In addition to the four sensors, a computer that is powerful enough to render and display 3D graphics is needed for this system. The computer will also be tasked with executing all four GAN models at one time, generating images for each simultaneously.

Appendix A: Acronyms and Abbreviations

FOV	Field of View
GUI	Graphical User Interface
Hz	Hertz
ICD	Interface Control Document
mA	Milliamp
MHz	Megahertz (1,000,000 Hz)
USB	Universal Serial Bus
GAN	Generative Adversarial Network

Appendix B: Definition of Terms

Epoch:
One complete pass of the entire training dataset through the learning algorithm.

GAN Model:
A machine learning model that creates new data that resembles training data.

Cycle Consistency Loss:
A loss function used in GAN training to ensure that transformations between domains (e.g., depth to RGB and back) preserve structural integrity.

Digital Twin:
A virtual representation of a physical object or system, used for simulation, analysis, and visualization of the robot's 3D reconstruction.

Intrinsic Matrix:
A matrix that contains the camera's internal parameters, used to map 3D points into 2D image coordinates.

GAN Assisted Map Reconstruction for first responders using Sensor Fusion

Christian Jeardoe, Quinn Leboeuf,
Kyle Smejkal, Rufus Tadpatri

VALIDATION

REVISION – 1
5 December 2024

Paragraph #	Test Name	Success Criteria	Methodology	Status	Responsible Engineer(s)
3.1	Operation Time	System will run continuously for 10 minutes and simultaneously process data as received	Run program for increasing amounts of time while checking each time if the data is correct	IN PROGRESS	Full Team
3.3.4.2	Detection Range	System will operate with a minimum range of 10 meters	Run program in room with 10 meters of open space to test if the model will render properly	IN PROGRESS	Full Team
2.1.1	GAN Architecture and ML envrionment for RealSense	Basic ML model construction and confirmation of model success.	Construction of a GAN Architeture and ML enviroment for Realsense	TESTED	Christian Jeardoe
2.1.2	GAN Architecture and ML envrionment for RGB	Basic ML model construction and confirmation of model success.	Construction of a GAN Architeture and ML enviroment for RGB	TESTED	Rufus Tadpatri
2.1.3	GAN Architecture and ML envrionment for Infrared	Basic ML model construction and confirmation of model success.	Construction of a GAN Architeture and ML enviroment for Infrared	TESTED	Kyle Smejkal
2.1.4	GAN Architecture and ML envrionment for Ultrasonic	Basic ML model construction and confirmation of model success.	Construction of a GAN Architeture and ML enviroment for Ultrasonic	TESTED	Quinn Leboeuf
3.3.4	Ultrasonic Sensor Controller	Arduino/Rasberry Pi controller fully operate ultrasonic sensor.	Dowload necessary arduino applications and setup environment for operating sensor	TESTED	Quinn Leboeuf
3.3.1	RGB Sensor Operation	Input signals from an RGB sensor to be recieved / usable data	Software on computer executes to run program	TESTED	Rufus Tadpatri
3.3.2	Infrared Sensor Controller	Rasphberry pi controller operate Infrared sensor / Recieve useable data from the sensor	Power light turns green and able to run test program using microcontroller	TESTED	Kyle Smejkal
3.3.2.1	Infrared Controller Computer Connection	Rasphberry pi controller is able to connect and send useable data to the computer running the GAN model	Mircocontoller saves data in directory that GAN model program has access to	TESTED	Kyle Smejkal
3.3.3	Realsense Sensor Operation	RealSense RGB and RGB-D sensors recieved / usable data	Framerate and resolution are checked to ensure they are within specification	TESTED	Christian Jeardoe
2.2.1	GAN Generator/Discriminator for RealSense	GAN generator successfully identifies fake generated data from real data	GAN generator has a near 50% "win rate" against discriminator	TESTED	Christian Jeardoe
2.2.2	GAN Generator/Discriminator for RGB	GAN generator successfully identifies fake generated data from real data	GAN generator has a near 50% "win rate" against discriminator	TESTED	Rufus Tadpatri
2.2.3	GAN Generator/Discriminator for Infrared	GAN generator successfully identifies fake generated data from real data	GAN generator has a near 50% "win rate" against discriminator	TESTED	Kyle Smejkal
2.2.4	GAN Generator/Discriminator for Ultrasonic	GAN generator successfully identifies fake generated data from real data	GAN generator has a near 50% "win rate" against discriminator	TESTED	Quinn Leboeuf
2.3.1	Point Cloud Generation for RealSense	3D point cloud map successfully developed from given input data	Preliminary point cloud is rendered to ensure host is receiving data	TESTED	Christian Jeardoe
2.3.2	Point Cloud Generation for RGB	3D point cloud map successfully developed from given input data	Preliminary point cloud is rendered to ensure host is receiving data	TESTED	Rufus Tadpatri
2.3.3	Point Cloud Generation for Infrared	3D point cloud map successfully developed from given input data	Preliminary point cloud is rendered to ensure host is receiving data	TESTED	Kyle Smejkal
2.3.4	Point Cloud Generation for Ultrasonic	3D point cloud map successfully developed from given input data	Preliminary point cloud is rendered to ensure host is receiving data	TESTED	Quinn Leboeuf
5.1	Full Subsystem Demo for RealSense	RealSense subsystem will fully generate 3D point cloud using images provided by GAN model	Point cloud models will be rendered	TESTED	Christian Jeardoe
5.2	Full Subsystem Demo for RGB	RGB subsystem will fully generate 3D point cloud using images provided by GAN model	Point cloud models will be rendered	TESTED	Rufus Tadpatri
5.3	Full Subsystem Demo for Infrared	Infrared subsystem will fully generate 3D point cloud using images provided by GAN model	Point cloud models will be rendered	TESTED	Kyle Smejkal
5.4	Full Subsystem Demo for Ultrasonic	Ultrasonic subsystem will fully generate 3D point cloud using images provided by GAN model	Point cloud models will be rendered	TESTED	Quinn Leboeuf

GAN Assisted Map Reconstruction for first responders using Sensor Fusion

Christian Jeardoe, Quinn Leboeuf,
Kyle Smejkal, Rufus Tadpatri

TIMELINE

REVISION – 1
5 December 2024

	8/19/2024	8/26/2024	9/2/2024	9/9/2024	9/16/2024	9/23/2024	9/30/2024	10/7/2024	10/14/2024	10/21/2024	10/28/2024	11/4/2024	11/11/2024	11/18/2024	11/25/2024	12/2/2024
Team Formation																
Subsystem Delegation																
ConOps Project Report																
FCR, ICD, Milestones, and Validation Plan																
Project Introduction Presentation																
Ultrasonic Subsystem Introduction Project																
RealSense Subsystem Introduction Project																
RGB Subsystem Introduction Project																
Infrared Subsystem Introduction Project																
Project Update Presentation																
RealSense Image Output																
RealSense Initial GAN Model																
RealSense Initial 3D Point Cloud Generation																
RGB Image Output																
RGB Initial GAN Model																
RGB Initial 3D Point Cloud Generation																
Infrared Image Output																
Infrared Initial GAN Model																
Infrared Initial 3D Point Cloud Generation																
Ultrasonic Initial Output																
Ultrasonic Initial GAN Model																
Ultrasonic Initial 3D Point Cloud Generation																
Final Presentation																
RGB Subsystem Demo																
Infraed Subsystem Demo																
Ultrasonic Subsystem Demo																
RealSense Subsystem Demo																
Final Report																

Not Started
Completed
In Progress
Behind Schedule

GAN Assisted Map Reconstruction for first responders using Sensor Fusion

Christian Jeardoe, Quinn Leboeuf,
Kyle Smejkal, Rufus Tadpatri

SUBSYSTEM REPORTS

SUBSYSTEM REPORTS

FOR

GAN Assisted Map Reconstruction for first responders using Sensor Fusion

APPROVED BY:

Rufus Tadpatri	12/04/2024
Project Leader	Date
John Lusher, P.E.	Date
T/A	Date

Change Record

Rev.	Date	Originator	Approvals	Description
1	12/04/2024	GAN Assisted Map Reconstruction		Final Version

Table of Contents

List of Tables

List of Figures

1. Introduction

Our project, GAN-Assisted Map Reconstruction for First Responders Using Sensor Fusion, is divided into four distinct sensor subsystems: RGB, RealSense, Infrared, and Ultrasonic. Each subsystem is equipped with its own input sensor and dedicated GAN model, each tailored to perform specific functions. Together, these subsystems will provide a continuous stream of data to generate a 3D virtual twin of any environment. All subsystems currently meet their requirements and operate as expected. Moving forward, each subsystem will undergo further refinement and has a well-defined plan for integration to achieve full sensor fusion.

2. RealSense Sensor Subsystem Report (Christian Jeardoe, 129001487)

2.1. Subsystem Introduction

The RealSense sensor subsystem serves as the primary sensor for our project. It captures both RGB and depth map images of the surrounding environment and utilizes a Generative Adversarial Network (GAN) model to enhance the raw depth map by smoothing noise and filling in missing data. Additionally, the subsystem processes the enhanced depth map to generate a detailed 3D point cloud representation, providing accurate spatial information for further analysis and applications.

2.2. Subsystem Details

2.2.1. Subsystem Design

The following is a high level block diagram of this subsystem.

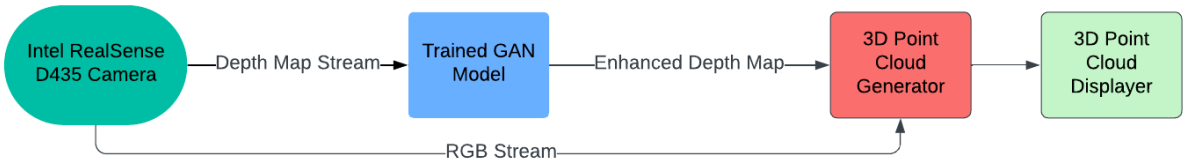

Figure 1: High Level Block Diagram of RealSense Subsystem

2.2.2. Subsystem Camera Specifications

This subsystem starts with the Intel RealSense D435 RGB-D camera. This table shows the specifications for the camera's sensors.

Resolution	1920 x 1080
Frame Rate	30 fps
FOV	69 x 42

Table 1: Specifications for RGB camera

Resolution	1270 x 720
Frame Rate	90 fps

FOV	87 x 58
Depth Accuracy	<2% at 2m
Ideal Range	.3m to 3m

Table 2: Specifications for Depth Sensor

With these specifications it was determined the best specifications for this subsystem was 640x480 resolution, this was done to make the training of the model less computationally intensive and decrease generation time when feeding the trained model a depth map to enhance. Similarly, the video streams are capped at 30 fps to ensure both depth images and RGB images are synchronized.

2.2.3. Subsystem GAN Model Design, Training, and Use

The GAN model for this subsystem was designed based on a Convolutional Neural Network (CNN) architecture with elements of a U-Net like design for the Generator. This type of architecture is often used in GANs for enhancement tasks like this. The Discriminator model is a CNN inspired design as well, used for binary classification.

The GAN model is trained on just over 700 depth map images collected using the RealSense camera. Each of these depth maps are copied and have “masks” placed over them which removes data from the original image. The original image and masked image are then used together in the generator to “teach” it to fill in that missing image data. The theory behind this is that if it can be taught to fill in missing, purposefully removed data, it will be able to fill in the inherent missing data that comes from raw depth map images. The Generator then sends either its own generated image, the “fake” one, or one of the unmasked “real” images to the discriminator where it needs to determine if its “real” or “fake”, using its binary classification. It will do this in a sequence of 20 epochs which take about 3-4 hours to complete. Once it is done, it generates a .PTH file which serves a model state save, meaning it can be loaded into another program to use the trained Generator.

To use the trained Generator, its .PTH file is loaded into another program where the now trained Generator can take in a raw input from the RealSense depth camera and generate an enhanced version of that depth map.

2.2.4. Subsystem 3D Point Cloud Generator and Viewer

The 3D Point Cloud Generator takes in RGB and Depth streams, ensures the pipeline receives both types of frames before moving to the next set, aligns the two frames,

maps the RGB color pixels on to that of the depth map points, and saves/exports the colored point cloud into a .PLY file. This file can then be imported into the 3D point cloud viewer and via OpenCV, be displayed in a 3D environment where the image can be manipulated. The following is an example of a .PLY file that was generated during lab being displayed.

Figure 2: Left, Centered, and Right Viewing Angles of 3D Point Cloud

2.3. Subsystem Validation

Since this subsystem's results are all visual in nature I found it a little difficult to devise ways to measure the validation of this system numerically. However, when it comes to the training of the model, checking the convergence of the Generator and Discriminator can be very useful.

```
C:\Users\jeard\Desktop\School\Fall2024\ECEN403\SubSystemProject\Reallsense>python bestGANmodel.py
Using device: cuda
Epoch 1/20: 100%|
Epoch 1/20, Generator Loss: 58.4501, Discriminator Loss: 22.2563
Epoch 2/20: 100%|
Epoch 2/20, Generator Loss: 47.8473, Discriminator Loss: 50.9038
Epoch 3/20: 100%|
```

Figure 3: Screenshot of Terminal Showing Generator Loss Vs Discriminator Loss

This is the output I have for the training loop for my GAN model, which shows me which device it's using ("cuda" is selected for my GPU), what the current epoch is its percentage complete of that epoch and the loss for both Generator and Discriminator. The goal is for them to converge at 50% each showing they have balanced eachother out.

Other validation checks in place are checking both the RGB and depth cameras when executing my model. This is what that display looks like.

Figure 4: Display of Both Cameras When Running

Finally, the last validation test in place is to check and see the trained Generator's outputs from the raw input image and to compare the two visually.

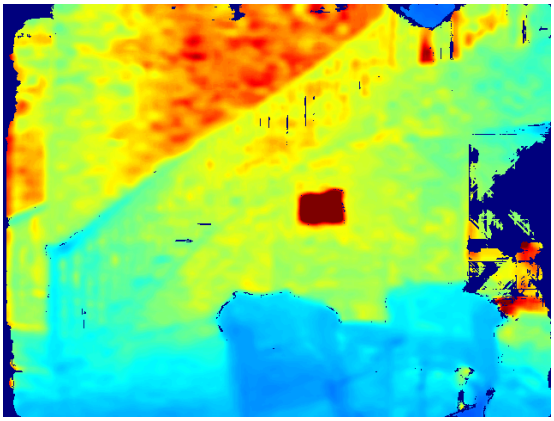

Figure 5: Raw Depth Map Input

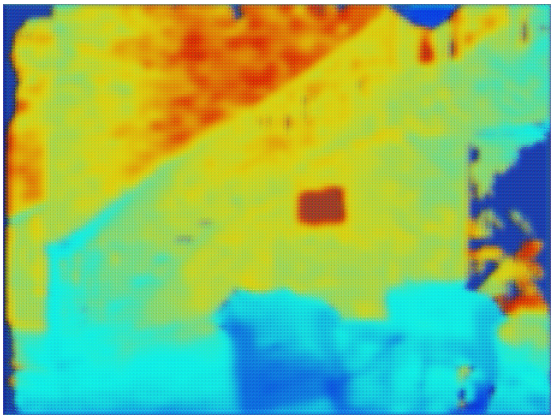

Figure 6: Enhanced Depth Map Output

The solid deep blue is missing data. As you can see the enhanced depth map shows significant improvements from the raw input. In several spots you can see where the GAN has filled in these spots and matched the surrounding area.

2.4. Subsystem Conclusion

All components of this subsystem operate as intended and to an acceptable standard. However, its current performance falls short of the desired standard. To achieve significant improvement, several key refinements are needed: utilizing a higher-quality depth map dataset (one free from pre-existing missing data like the current dataset), increasing the dataset size substantially (from 700 images to an optimal range of 10,000 to 500,000 for effective model training), and leveraging superior computational

resources (as current limitations restrict training to 20 epochs with runtimes of 3–4 hours). With these improvements, the model is expected to perform markedly better. Many of these limitations are slated for resolution or improvement in the next phase of development for ECEN 404.

3. RGB Sensor Subsystem Report (Rufus Tadpatri 830003175)

3.1. Subsystem Introduction

The RGB sensor subsystem is designed to operate in scenarios with high visibility. This subsystem takes in an input image, preprocesses the image using the trained GAN model, and independently generates a 3D point cloud from the image.

A GAN model was developed that converts RGB images to 3D point clouds. RGB data is taken in and preprocessed to optimize input for the GAN. The RGB sensor allows the ability to train the GAN model to convert from RGB to point cloud in order to integrate into the final 3D model.

Currently, there exists a technology that utilizes CNN layers and diffusion-denoising approaches. Convolutional neural networks use several layers to maintain relevant information without causing a storage issue. Our RGB subsystem will use machine learning algorithms to train the model in order to develop a GAN model. This model will be trained to preprocess RGB data; this will help with processing speed as well as the reliability of the model.

3.2. Subsystem Details

The GAN model was developed using StackGAN architecture. StackGAN architecture is a two-stage network, meaning it has two generators and two discriminators. The python program was set up for the GAN model to first be trained on a sample dataset pulled from an online repository. This dataset contained 200 images, and the model was trained on each image for 500 epochs. The more data that could be put through the model, and the more epochs per image, would result in a higher developed and accurate model. Due to time constraints- 500 epochs for 200 images took multiple days to train -the model was not further developed. Moving forward, continuing to train the model on more data and for longer periods of time is an option to continue to improve the product. After the model was trained, a second python program, which references the GAN model, was developed to function as the user interface. The program opens the RGB sensor viewer via CL-Eye Test program, takes in the input image, then preprocesses the image and puts it into the trained GAN model. From this, the GAN model is able to generate a 3D point cloud, which opens on the laptop using OpenCV for the user to view.

Further steps to be taken to improve the product includes increased training of the GAN model, both on images and epochs, as well as aligning the sensor with others on the project. For example, concurrent use with the depth sensor alone will allow for a more space-aware 3D point cloud.

In the next stages of the project, for subsystem integration, the GAN model built to support the RGB sensor will work directly with the other three sensors used on the project.

3.3. Subsystem Validation

Figure 7 below shows the input image that can be put into the model. Figure 8 and figure 9 show what the 3D point cloud looks like after being put through the subsystem.

Figure 7: Input image into RGB Subsystem

Figure 8: Output 3D Point Cloud

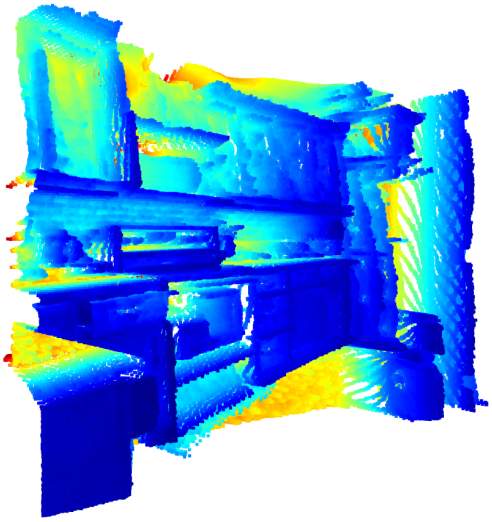

Figure 9: Output 3D Point Cloud Alternate Angle

As seen in figures 8 and 9, from taking in an input image, the program will output a 3D rendering that can be rotated and navigated to see multiple angles of the environment.

3.4. Subsystem Conclusion

The RGB subsystem functions as intended. It correctly takes in an input image from an RGB sensor and generates a 3D point cloud accurately and in a timely manner. The GAN is also set up for subsystem integration. Moving forward, the RGB subsystem is fully completed per the individual subsystem deliverable and is ready for the next stages of the project.

4. Thermal Sensor Subsystem Report (Kyle Smejkal 627002694)

4.1. Subsystem Introduction

The thermal camera subsystem captures thermal image data to show areas of high heat for the first responders. In order to make a usable point cloud from this data, a depth map is needed. In the final form of this project, the depth data from the Realsense camera or Ultrasonic sensor can be used for that depth map. For this report, the depth data from the Realsense camera was used. The thermal image data and depth data are then used to train two separate GAN models, one for each data type. These GAN models are based on Convolutional Neural Networks and are used to generate images based on the input sensor data, which can be used to make a point cloud.

4.2. Capturing Thermal Data

The thermal imaging data is captured using a MLX90640-55 thermal camera connected to a Raspberry PI 3 model B v1.2 via the I2C interface. **Figure 10** shows the pinout of connecting the camera to the Raspberry PI. **Table 3** shows the specifications of the camera.

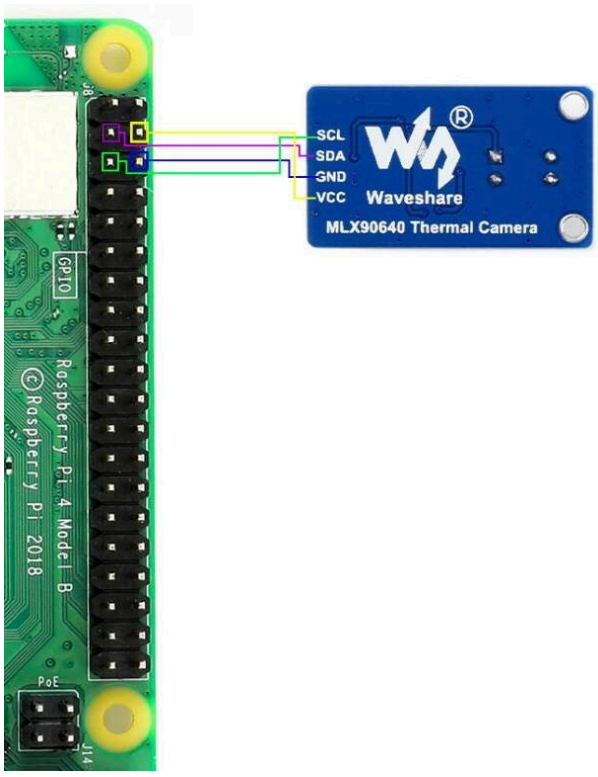

Figure 10: Pinout of Thermal Camera connection

Resolution	32 x 24
FOV	55° x 35°
Target Temperature (Temperature range of sensor)	-40℃ ~ 300℃
Refresh Rate	0.5Hz~64Hz (Programmable)

Table 3: MLX90640 Thermal Camera Specifications

The thermal data is captured in a 32x24 array that is then resized to 640x480 to match the size of the depth image from the realsense camera and saved as a '.PNG'. The image is then moved over to the host device for training the GAN model.

4.3. Training the GAN model

Before training the GAN model the images need to be formatted correctly. This is done by changing the depth and thermal images into 1 color channel 640x480 pixel images. The thermal images are outputted from its sensor as only having 1 channel, so no change is needed. The depth image comes in as an 640x480 RGB image with blue colors being closer to the camera and red meaning it is further away from the camera, so only the blue channel is saved to represent the depth image. Lastly, is the GAN architecture. **Figure 11** shows the architecture for the Discriminator and **Figure 12** shows the architecture for the generator.

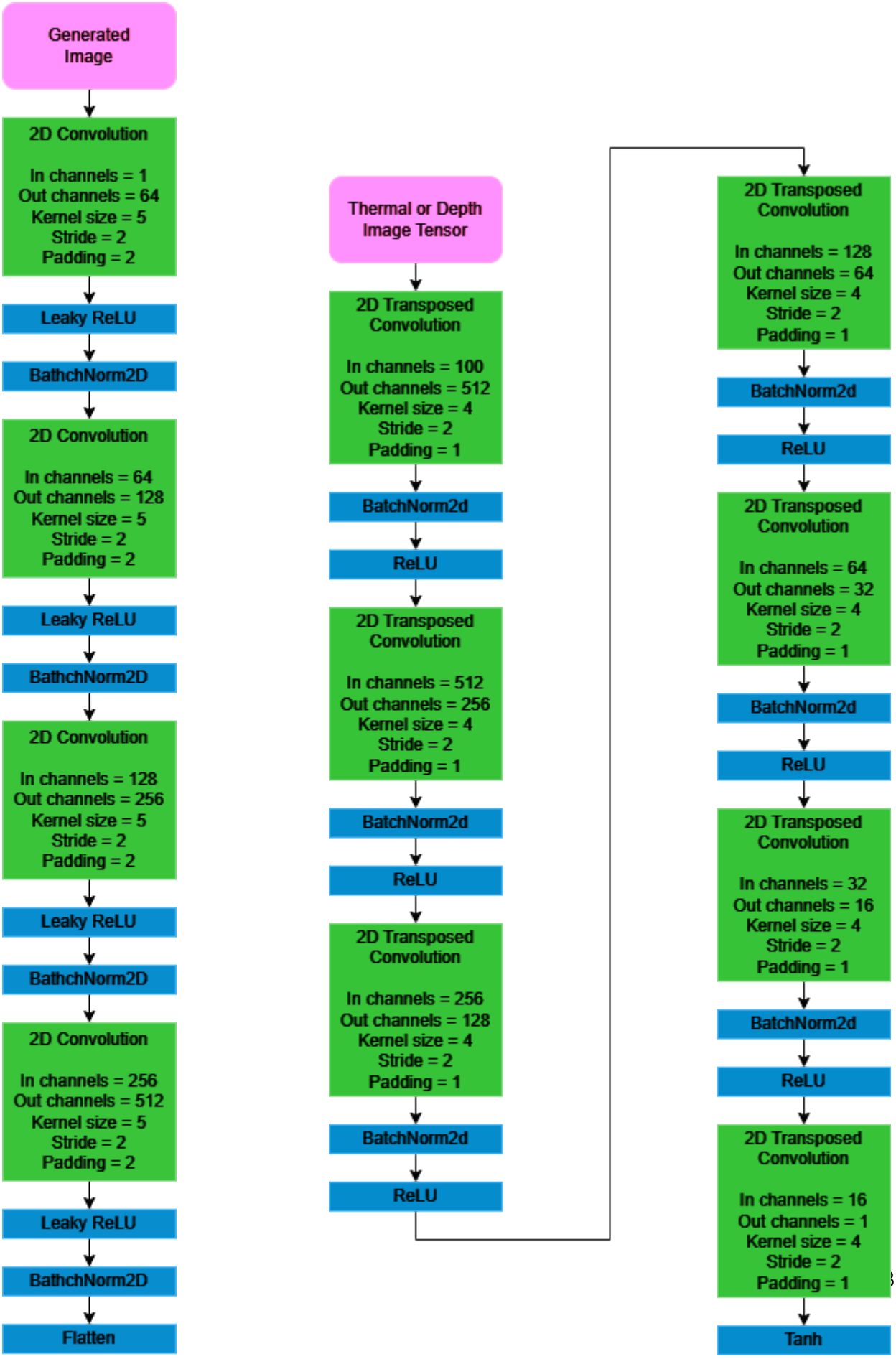
Generated Image
2D Convolution
In channels = 1
Out channels = 64
Kernel size = 5
Stride = 2
Padding = 2
Leaky ReLU
BathchNorm2D
2D Convolution
In channels = 64
Out channels = 128
Kernel size = 5
Stride = 2
Padding = 2
Leaky ReLU
BathchNorm2D
2D Convolution
In channels = 128
Out channels = 256
Kernel size = 5
Stride = 2
Padding = 2
Leaky ReLU
BathchNorm2D
2D Convolution
In channels = 256
Out channels = 512
Kernel size = 5
Stride = 2
Padding = 2
Leaky ReLU
BathchNorm2D
Flatten
Thermal or Depth Image Tensor
2D Transposed Convolution
In channels = 100
Out channels = 512
Kernel size = 4
Stride = 2
Padding = 1
BatchNorm2d
ReLU
2D Transposed Convolution
In channels = 512
Out channels = 256
Kernel size = 4
Stride = 2
Padding = 1
BatchNorm2d
ReLU
2D Transposed Convolution
In channels = 256
Out channels = 128
Kernel size = 4
Stride = 2
Padding = 1
BatchNorm2d
ReLU
2D Transposed Convolution
In channels = 128
Out channels = 64
Kernel size = 4
Stride = 2
Padding = 1
BatchNorm2d
ReLU
2D Transposed Convolution
In channels = 64
Out channels = 32
Kernel size = 4
Stride = 2
Padding = 1
BatchNorm2d
ReLU
2D Transposed Convolution
In channels = 32
Out channels = 16
Kernel size = 4
Stride = 2
Padding = 1
BatchNorm2d
ReLU
2D Transposed Convolution
In channels = 16
Out channels = 1
Kernel size = 4
Stride = 2
Padding = 1
Tanh

Figure 11: Discriminator Architecture **Figure 12: Generator Architecture**

Now using the above architecture, two separate GAN models are trained. One uses depth images to train and the other uses thermal images. During the training process, the loss function used at all steps is the BCEWithLogitsLoss function from pytorch. This function combines the Sigmoid and BCEloss functions into one. There are three steps each epoch during the training process. First, the discriminator trains on the images provided. (Only one type of image is trained on at a time) Next, the discriminator trains using images generated by the generator. Lastly, the generator is trained using the discriminator as a judge. At the end of each epoch, the loss of the discriminator and generator are outputted to show the progress of the training process.

4.4. Point Cloud Generation

The point cloud generation is done by mapping the thermal image into 3d space using the generated depth image to map it along the z-direction. This is done using the camera intrinsics matrix to line up the two images. **Figure 13** is an example of this. The image in **Figure 13** can be thought of as the thermal image and the scene as the depth image.

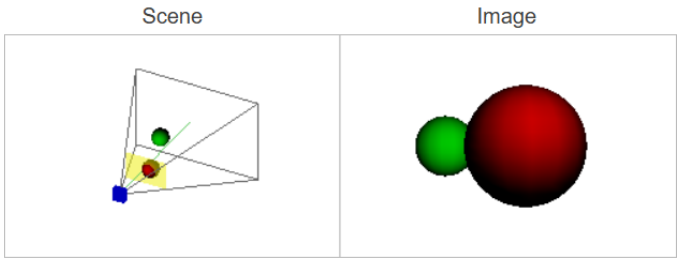

Figure 13: Camera Intrinsics Example

The point cloud is generated by going pixel by pixel through the thermal image and assigning it a depth value based on the depth image. **Figure 14** and **Figure 15** show an example of a thermal image and the depth image respectively. Note that the Realsense camera has a wider FOV then the MLX90640, so before the point cloud is calculated, the depth image has to be processed to account for this difference. **Figure 16** and **Figure 17** show what the resulting generated point cloud looks like from two different angles.

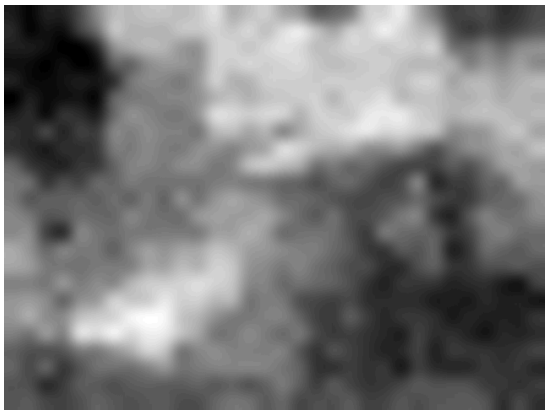

Figure 14: Thermal Image Example

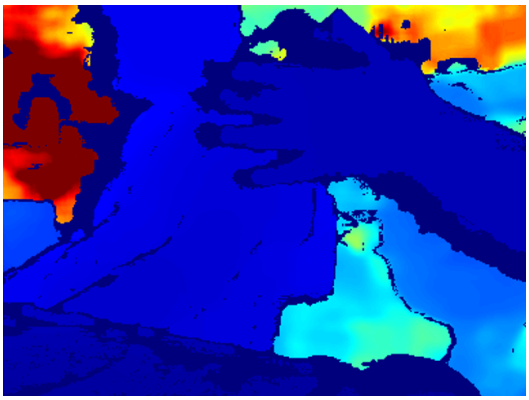

Figure 15:Depth Image example

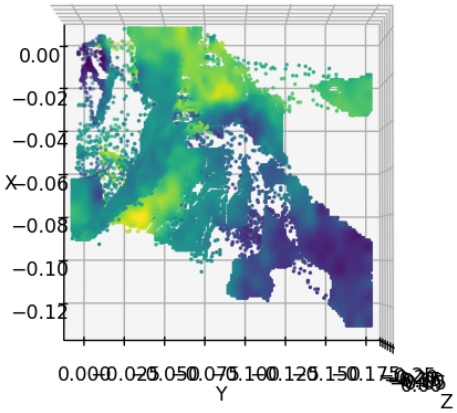

Figure 16: Point Cloud Example

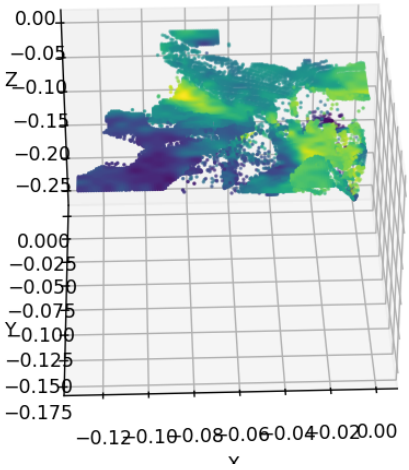

Figure 17: Different angle of Point Cloud

4.5. Subsystem Validation and Conclusion

The thermal camera subsystem works as intended as shown in the previous figures, but is held back by the camera that is currently being used and the amount of training time/data that the model has had. The thermal camera being used is a very low resolution, so when the GAN model trains on the low resolution images, it outputs a very noisy image where the details of the image are not distinguishable at all. This leads to a point cloud that isn't useful. The model has also only been trained on just over 500 images for about 10 hours in total. To improve this subsystem, a better camera with higher resolution and more training time/data are needed.

5. Ultrasonic Sensor Subsystem Report

(Quinn LeBoeuf, 529007084)

5.1. Subsystem Introduction

The Ultrasonic Sensor Subsystem relies on leveraging sparse and noisy distance measurements captured by the ultrasonic sensor and enhancing this data with a GAN model to produce accurate 3D point clouds. The GAN model for this subsystem is designed to use processed output measurements from the ultrasonic sensor. This processed data involves ultrasonic distance measurements that will then be used to develop a depth map from ultrasonic sensor data. Processed data from here are fed into the GAN model to further create an improved depth map and thus infer an RGB image, which is used to develop a 3D point cloud. This point cloud generated will later be merged together with other sensor data to form one unified view of some scanned environment, while also being used as a backup considering the project is designed for emergency situations, in which other sensors may not properly work depending on environmental hazards.

5.2. Data Retrieval and Processing

Data retrieval for this subsystem occurs through the ultrasonic sensor (HC-SR04) and the corresponding microcontroller (Arduino Uno). From this sensor, distance measurements up to 4 meters can be read. Original minimum distance requirements were 10 meters, however this proved to not be possible given budget constraints based on how data will need to be processed. A 60 millisecond delay between measurements was decided for maximum accuracy without causing overlapping measurement issues during sensor movement and rotation. Ultrasonic sensor measurements will still need conversion to depth map data to provide an image for GAN model training, as well as input data. This conversion will occur by using multiple ultrasonic sensors which will use phased array analysis to create a noisy depth map of some environment. Temporary online datasets have been used to test and train the GAN model for operation based on predicted ultrasonic sensor scans. More discussion of this topic is provided in the subsystem validation.

5.3. GAN Model Structure and Training

The GAN model used in this system is designed to enhance depth maps and generate inferred RGB images, ultimately enabling the creation of 3D point clouds. The architecture consists of two key components: the Generators (U-Net style) and Discriminators (PatchGAN style), trained using a combination of adversarial and cycle consistency losses. Training begins by first preprocessing the paired depth and RGB images from the dataset through resizing and normalization transformations. The model has two generators: one generates RGB images from depth maps, and the other cleans the noisy depth maps. Generators are trained on L1 losses between generated outputs and their respective targets, while Discriminators classify between real and fake images using adversarial losses. These two objectives ensure that the generators produce realistic and accurate outputs. At each epoch, the model performs an alternative update of generator and discriminator parameters by using the Adam optimizer. The cycle consistency losses ensure that the depth to RGB and vice-versa transformations preserve the

content structure of the input, while the adversarial losses direct the generators in producing visually compelling output.

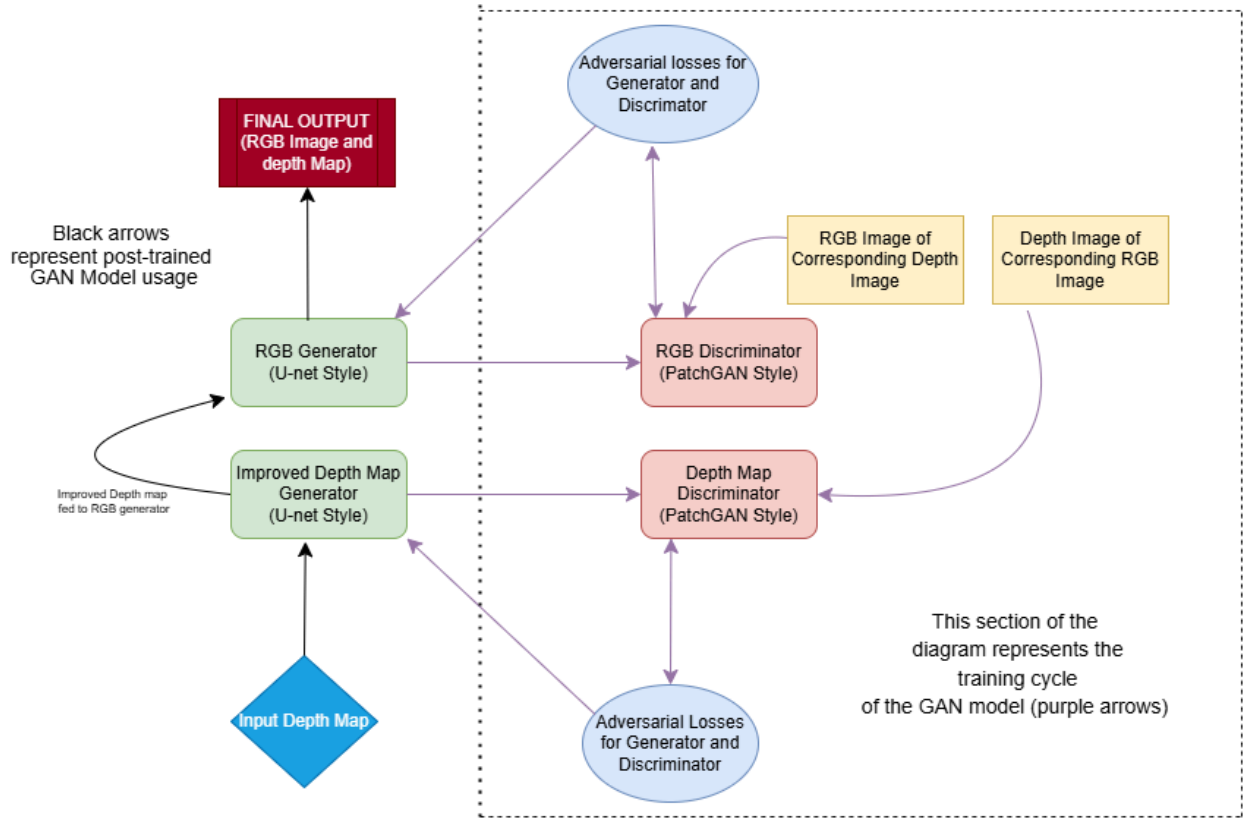

Figure 18: GAN model training and usage implementation

The model was trained for 50 epochs while logging Generator and Discriminator losses at regular intervals. This training ensures the GAN model is capable of properly enhancing depth maps from the ultrasonic sensor as well as generating an RGB image, which is needed for creating a point cloud image. Figure 18 shows the input/output architecture of the model, and highlights the training architecture for proper image generation

5.4. Subsystem Validation

5.4.1. Data Retrieval Validation

The ultrasonic sensor data collection was validated by comparing the measured distances against known reference values, demonstrating consistent accuracy within an acceptable error margin. This confirms that the sensor is functioning properly and providing reliable input for further processing by the GAN model. While the sensor data collection validation has been executed, it is still not processed properly for the GAN model to use. This is a problem that was realized halfway through the project, however focus has been on the machine learning model and its performance. A 3D ultrasonic scanner design has been conceptualized as the solution to

this issue, but requires many outside implementations (PCB design, Phased array analysis, etc.) that will have to be worked on before, as well as going into, ECEN 404.

5.4.2. GAN Model Training Validation

During Training, the model outputted the losses of both the generators and discriminators during each epoch. Generator losses decrease while maintaining a moderate value indicating the improvement of the generator during training. The discriminator loss values also decrease towards more balanced levels. Both these occurrences prove the model is learning properly and the data is not converging (or being overtrained). Figure 19 shows an image of a training test to make sure the loss functions were working correctly.

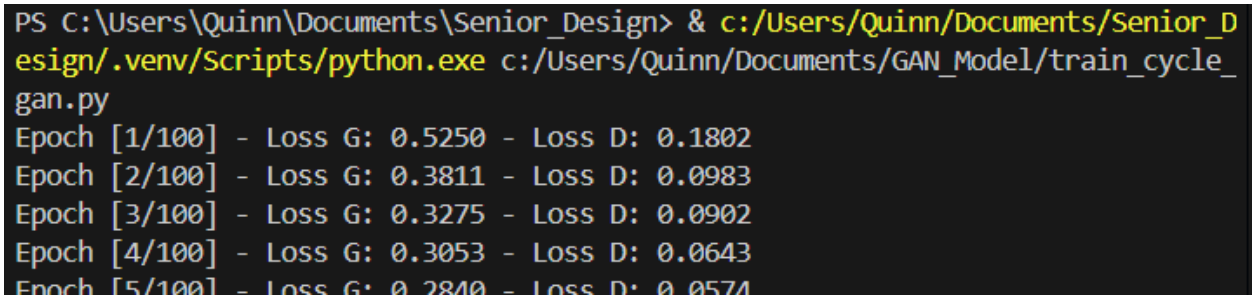

Figure 19: Generator and Discriminator Losses During Training Test

5.4.3. GAN Model Operation Validation

The operation of the GAN Model was validated using visual output results. In figure 20, we can see multiple scenarios where the model takes in depth image data (the depth image data is based on the ground truth RGB image), improves the depth image, generates an RGB image from said depth image, and then a corresponding point cloud is generated. Possible improvements for environmental generation accuracy include merging point clouds from multiple angles to build a complete environment (notice the point cloud cannot understand there is empty space behind the table unless another image is taken from a separate angle). More training with a larger dataset could prove to be useful given more time and improved GPU performance.

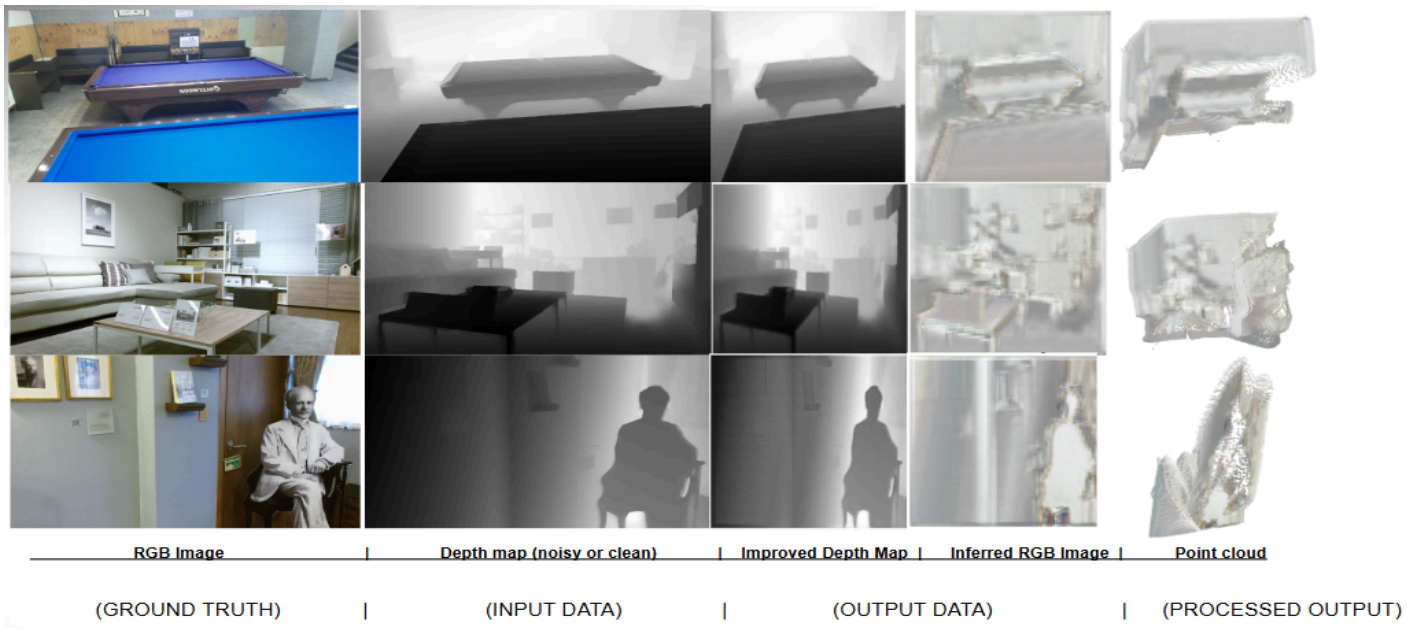

Figure 20: Trained GAN Model Input and Output Results

5.5. Subsystem Conclusion

The Ultrasonic Sensor Subsystem uses machine learning (GAN model) to reconstruct sparse/noisy ultrasonic depth data into high-resolution 3D reconstructions. By using a U-Net-based generator for depth map enhancement and RGB inference, along with a PatchGAN discriminator for realistic output validation, the system ensures accurate and visually coherent results. The integration of adversarial and cycle consistency losses further refines the model performance, which enables the detailed creation of 3D point clouds. This subsystem will play a key role in backup environment reconstruction, in which this sensor can collect and process data where others cannot, enabling critical applications of environmental reconstruction for first responder operations.

GAN Assisted Map Reconstruction for first responders using Sensor Fusion

Christian Jeardoe, Quinn Leboeuf,
Kyle Smejkal, Rufus Tadpatri

INTEGRATED SYSTEM REPORT

REVISION – 1
25 April 2025

INTEGRATED SYSTEM REPORT
FOR
GAN Assisted Map Reconstruction for first responders using Sensor Fusion

APPROVED BY:

Rufus Tadpatri	4/25/2025
Project Leader	Date
John Lusher, P.E.	Date
T/A	Date

Change Record

Rev.	Date	Originator	Approvals	Description
1	4/25/2025	GAN Assisted Map Reconstruction		Final Version

Table of Contents

List of Tables

List of Figures

1. Overview

For this project, our team was tasked with developing multiple generative adversarial network (GAN) models designed to enhance and allow for 3D map reconstruction through advanced sensor fusion techniques. Each GAN model was specifically tailored to address unique challenges associated with various sensors within the system, optimizing their contributions to the overall mapping accuracy. Throughout the duration of the project, we successfully implemented these GAN models, enabling effective real-time fusion of sensor data. The culmination of our efforts resulted in the creation of a robust, real-time 3D map reconstruction program. This solution not only integrates multiple sensor inputs seamlessly but also leverages GAN-generated enhancements to improve the precision and completeness of the generated maps.

1.1. Major Changes During Integration

1.1.1. Ultrasonic Subsystem Switch to LiDAR

After major deliberation with the sponsor as well as instructors, it was decided at the beginning of the second semester to change the ‘ultrasonic subsystem’ to the ‘LiDAR subsystem’. This was mainly due to the inability to create depth maps from single ultrasonic sensor data. The only solutions found to create the necessary depth maps were discussed and found to be inefficient for our sponsors project requirements. Switching to LiDAR allowed us to get 3D point cloud data, which also required a complete change and retraining of the associated GAN model.

1.1.2. Infrared Subsystem Change

The original MLX90640-D55 had a resolution of 32x24 and was found to not adequately resolve infrared object data past 5 meters. The low resolution would end up negatively impacting the quality of the output of the GAN model. After discussion with the sponsor, we determined that the minimum distance required for the subsystem to be able to resolve infrared data was 10 meters. In order to solve these issues, we changed the sensor to a FLIR Lepton 2.5. This new sensor has a resolution of 80x60, which improved the quality of the output of the GAN model and was able to resolve infrared object data well past 10 meters. The new sensor is able to be plugged directly into the computer running the program rather than being captured on a raspberry pi and transferred to the computer running the program via ethernet or USB, so the raspberry pi is no longer needed.

1.1.3. Scanning Requirements and Demo Prototype

The original sponsor requirements involved a controlled mobile robot that could move around a room and scan environments. During the integration process, it was found that we needed a robotic device that could bear the weight of two laptop devices which ran the Unity application and controlled the LiDAR drivers and ROS system. Due to major budget constraints, it was discussed to be not possible to make or order a device to meet these requirements. After sponsor approval, we decided on using a flagpole type device to mount the sensors and center them with the IMU. All devices would be connected to the necessary computers which were

separate from the physical prototype. The “sensor flag pole” could then be spun 360 degrees to scan any 3D space.

2. Execution

At the beginning of this semester, we undertook the significant task of refactoring the data handling and transmission pipeline for the point cloud data produced by each subsystem. This was essential due to the project's requirement to operate in real-time, a departure from our previous approach at the conclusion of 403, where each subsystem individually demonstrated static GAN-processed point cloud renderings optimized for single-frame visualization. To achieve real-time functionality, we transitioned from our original method, which involved iterating over individual points and exporting them into file formats such as .ply or .obj, to packaging the data into JSON format for rapid and efficient communication.

This modification enabled integration with Unity via WebSocket connections hosted on localhost. Specifically, our system architecture consisted of a Python-based server responsible for collecting sensor data, processing it through the appropriate GAN model, packaging the resultant point cloud data into JSON format, and transmitting it through a WebSocket connection. Concurrently, a script implemented in Unity was developed to listen on the designated localhost port, retrieve the incoming JSON data, unpack the serialized information representing a single frame of point cloud data, and render it within the Unity environment in real time. This modification, while lengthy in its development time, was crucial for real time rendering. Only the LiDAR system differed from this approach as it had its own laptop which hosted the python server. For this, we used an ethernet connection and connected via IP addresses rather than a localhost, as shown in figure 4. Nevertheless, the end result was the same.

3. Validation

3.1. IMU Accuracy Validation

One of the many parts of our validation plan was ensuring accurate rotational measurements of the Inertial Measurement Unit (IMU). This was validated by running Unity with only the IMU and physically rotating it at 90 degree angles and checking Unity’s internal degree measurement system to compare them against each other.

Table 1: IMU Physical Angle vs Unity Measured Angle

Physical IMU Rotation (degrees)	0	90	180	270	360
Unity IMU GameObject Rotation (degrees)	0.012	90.573	180.646	271.302	361.835

We observed that the IMU exhibited minimal drift, typically remaining within two degrees under standard operating conditions. The two primary factors affecting angular deviation were rotational speed and operational duration. While extended operational periods could lead to noticeable cumulative drift, within the ten-minute validation period we established, duration was not a significant contributor to measurement inaccuracies. In contrast, rotational speed had a substantial effect; rapid rotations caused the IMU to either overcompensate or undercompensate, resulting in discrepancies between the physically measured angle and the angle recorded in Unity.

3.2. Sensor and System Validations

We meticulously executed our validation plan, ensuring each test adhered strictly to its defined success criteria and methodology. Through multiple tests and demonstrations, we confirmed our model's capability to capture, package, transmit, and render each subsystem's generated 3D point cloud in Unity in real time. Verification was achieved primarily through reviewing data logs indicating successful data transmission and by observing dedicated Unity GameObject bins created dynamically at runtime for each subsystem.

Detection range validation involved positioning sensors precisely 10 meters from an object or wall, as specified in our validation criteria, and verifying correct rendering within Unity. Operational stability was assessed over a ten-minute interval by simultaneously running all four Python servers alongside Unity, closely monitoring for any interruptions or crashes. During testing, we observed that the program consistently operated for approximately 23 minutes before Unity encountered crashes due to storage limitations, far exceeding our ten minute requirement. Which could have been extended with an addition of external storage for the hosting laptop.

For comprehensive project validation, we introduced environmental variations impacting sensor operation, particularly reduced lighting conditions and complex geometrical configurations. In lower lighting conditions, system performance was generally stable, although the RGB sensor exhibited significant degradation. In contrast, environments with increased geometric complexity slightly diminished the overall performance of all systems, yet these reductions remained within acceptable error margins.

Lastly, the "Null Data" feature set outlined in our validation plan, introduced in our project's final semester, was ultimately not implemented. Although we developed several functional prototypes, none reached the requisite maturity for the final demonstration and were consequently omitted due to time constraints.

4. System Performance

4.1. RealSense Integrated Performance

The RealSense subsystem performed as expected after integration. It exhibited several of the expected performance hiccups noted at the end of 403 where the GAN model, in an attempt to

fill missing data points in the depthmap, tended to “over-smoothen” and blur what was once defined edges of corners and other objects. However, in this effort, it also decreased the variance of depth distances making for smoother flat surfaces and a reduction in the degree to which outliers were straying from the mean distance.

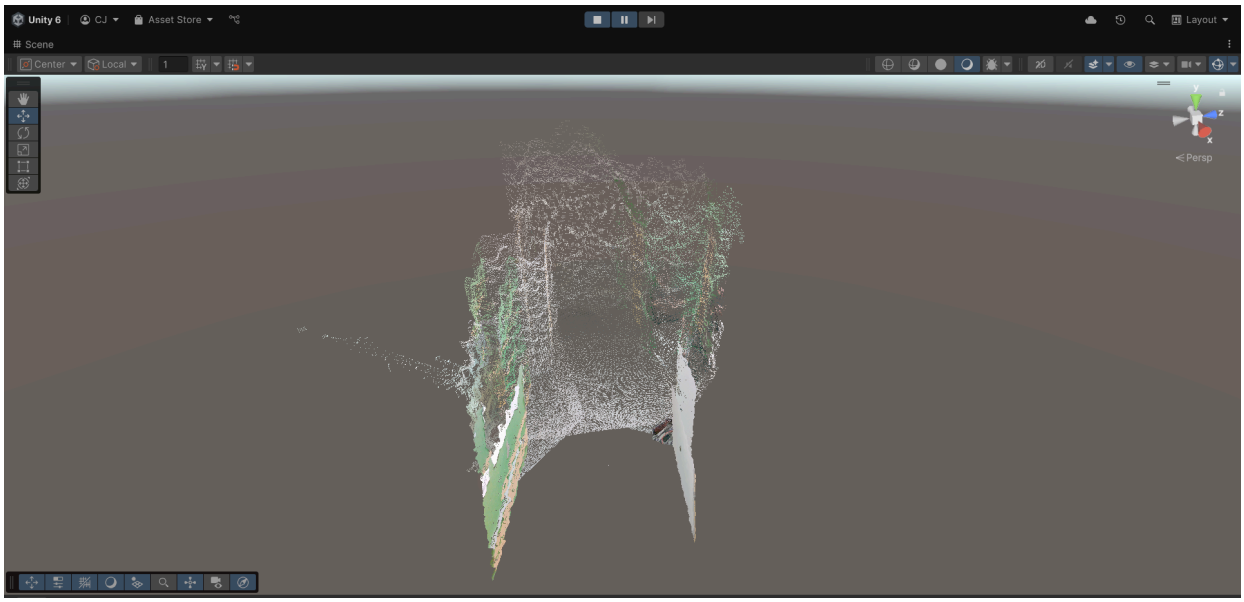

Figure 1: RealSense Scan of Hallway Environment Shown in Figure 5

4.2. Infrared Integrated Performance

With the new sensor being able to be plugged directly into the computer running the Python server, the integration of the infrared subsystem with the other subsystems was much simpler. The GAN model performance improved significantly due to higher-resolution training data, although training the GAN model took much longer. After integration with the other subsystems in Unity, the subsystem performed as expected and was able to output a point cloud into Unity alongside other subsystems.

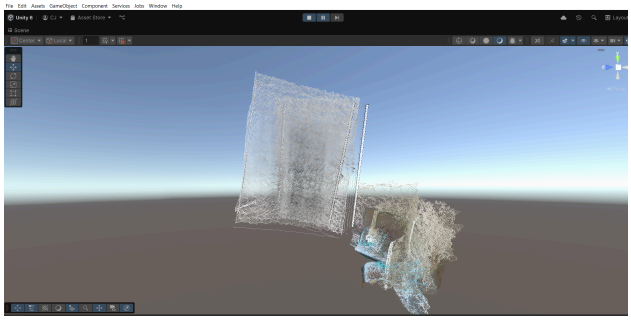

Figure 2: Infrared Subsystem Alongside other Subsystems in Unity

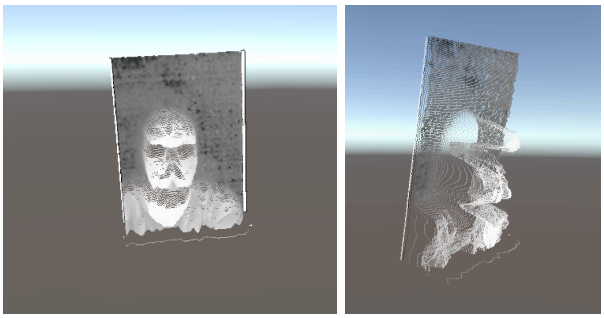

Figure 3: Infrared Point Cloud in Unity

4.2.1. Infrared Unity Visualization

The point cloud was initially saved as an .obj file; however, to enable visualization within Unity, it was necessary to reformat the data as a JSON file for transmission to the Unity application via the webhook. A scaling and sampling variable were also added to assist in the visualization in Unity. The scaling variable was used to make sure the size of the point cloud would be similar to the size of the point clouds of other subsystems. The sampling variable was used to equally remove data points from the point cloud, so that the program running or unity wouldn't crash. This allowed for the fine tuning between performance and quality.

4.3. LiDAR Integrated Performance

After full integration with Unity and the IMU, the LiDAR was able to fully scan 360 degree environments, improve point cloud density through use of the newly trained GAN model, and fully render in Unity along with the other sensors. While the LiDAR has its own integrated IMU, the IMU on the prototype was necessary to align point clouds with other sensors rendered data, as well as properly layer each individual scanned point cloud received from the LiDAR properly as the prototype was spun.

4.3.1. USB-C Tethered Data Pipeline

Unlike the other sensor devices, the LiDAR required an ROS environment and managing of drivers to operate. These applications required ~10 GB of space, which was not available on the device running Unity, and the device was at its computational limit when scanning and integrating point clouds in Unity (very slow and sometimes crashed). This required the drivers and ROS to be run on a separate system. To concurrently send system data from computer A (operates LiDAR) to computer B (performs Unity rendering), we set up a USB-C-to-Ethernet connection, hosted an on-device websocket server on computer A, which then computer B would connect and pull .JSON point cloud snapshots every few seconds to render real time in Unity.

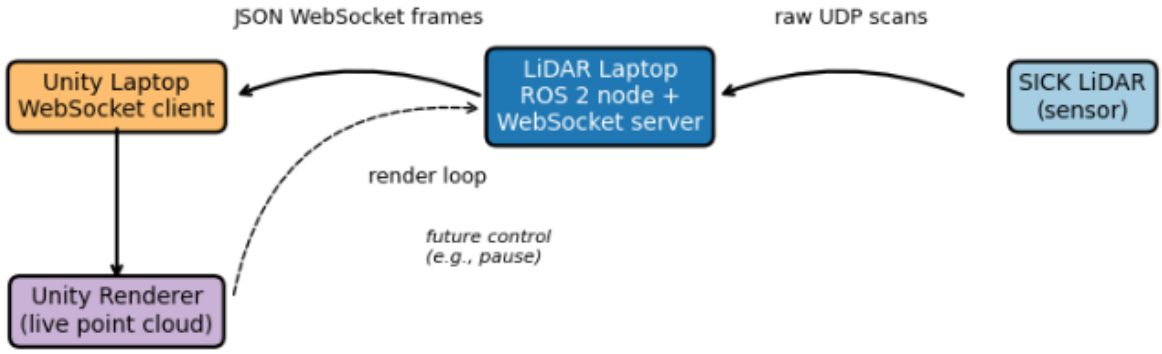

Figure 4: USB-C Tethering System

4.3.2. LiDAR Unity Visualization

Figures 5, 6, and 7 show the complete integrated LiDAR scan individually from the other sensors. Having the LiDAR work on its own allows the prototype to operate in environments that would visually impair some of the other sensors. This is specifically useful for environments with little to no lighting.

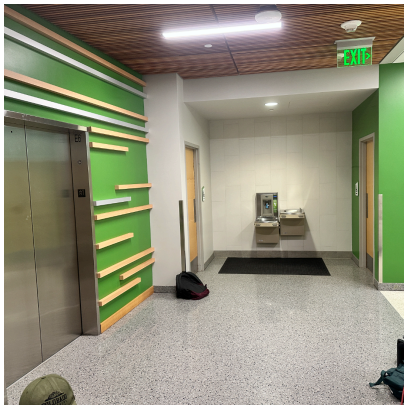

Figure 5: Ground Truth of Hallway Environment

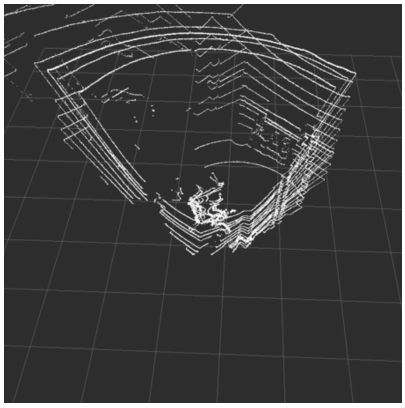

Figure 6: LiDAR Unaltered Scan

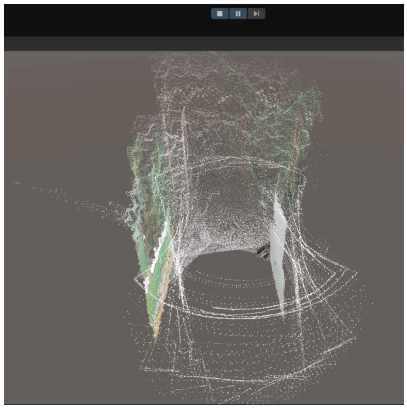

Figure 7: Improved Scan (with Realsense)

4.4. RGB Integrated Performance

4.4.1. RGB Value Color Correction

Upon the conclusion of last semester, the RGB subsystem produced a 3D point cloud without maintaining the initial RGB value from input. The final output of the script would show a 3D environment with the colors showing the heat intensities as opposed to the rgb values showing true colors. The gan model was trained by identifying heat values corresponding to distances from the sample dataset that could be used to identify distances when given an input image from the RGB sensor. This semester, I updated the program to maintain the RGB values by creating a separate data variable that stores the RGB values that get inserted back into the 3D point cloud after being rendered from the color intensities.

Figure 8: Ground Truth Image of Kitchen Environment

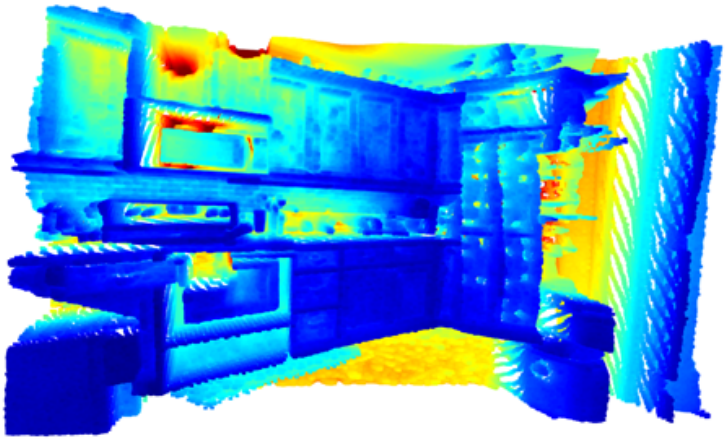

Figure 9: Output Point Cloud from Subsystem at the end of 403

Figure 10: Output Point Cloud from Subsystem at the end of 404

Figure 11: Alternate Angle of Output Point Cloud from Subsystem at the end of 404

As shown in figures 10 and 11, the RGB color values were maintained after modifications to the subsystem script.

4.4.2. RGB Unity Visualization

In order for the RGB subsystem to be properly integrated, the program needed to output as a .obj file. For a large period of time, the program outputted a .ply file. This was a feasible output to demonstrate a 3D point cloud on my personal device, but was not feasible when putting the script into unity. There was a major scaling issue in unity due to the parameters of the sensor size, and changing the output from a .ply to a .obj file amended this. Once the program was outputting as a .obj, the subsystem was able to work properly into unity and be integrated alongside the other subsystems.

Figure 12: RGB Subsystem Output in Unity

Figure 13: Alternate Angle of RGB Subsystem Output in Unity

As shown in figures 12 and 13, outputting the point cloud as a .obj file successfully allowed the script to run without issue within unity.

4.5. Combined System Performance

4.5.1. Combined Unity Visualization

As shown in previous sections, each sensor was capable of individually developing their own 3D point cloud which was then integrated into Unity. The combined scan Figure 14 shows all four sensor scans scaled and layered in Unity. This provides a fairly dense point cloud, with

each individual scan representing different data that the other relative sensors cannot collect.This provides us with full sensor fusion capabilities so the prototype can work in various environments. If one or more sensors fail, others can take their place, giving the device the advantage of scanning various hazardous environments.

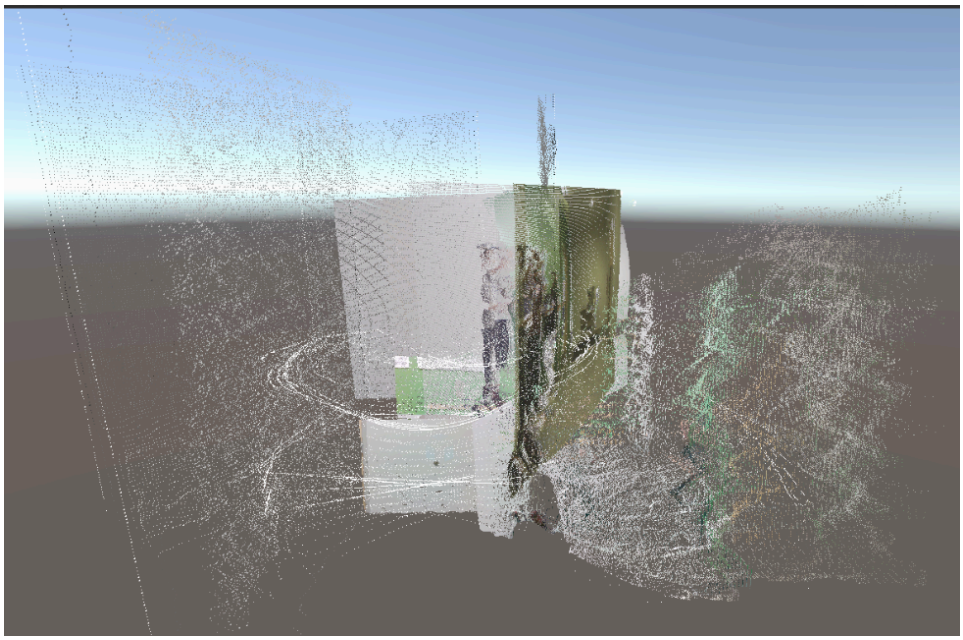

Figure 14: Full System Output of Hallway Environment

While the image may appear messy, within the Unity program, the user would have the capability to navigate around the environment and have a much cleaner view of the render. Because of the full sensor fusion capabilities, completing a scan in this manner is the optimal way for the four sensors to work together and create the output in Figure 14.

5. Conclusions

5.1. Limitations

5.1.1. RGB/Infrared Dataset and Training Limitations

When training the GAN models, the biggest limitations the team came across were time and available datasets. Because the first half of this project was constrained to roughly a 3 month time period, the amount of time that the team could spend training the models was extremely limited. Ideally, we could have spent almost 6 months alone training the models on close to 50,000 images; instead, each sensor used only about 500 to 1000 images to train the models. The reason the dataset was so limited is there are no publicly available datasets that matched our needs, so we had to create the datasets ourselves. This led to an imperfect, though still fully functional, model that has a lot of generalization. Furthermore, the capabilities of the RGB sensor and the infrared sensor led to point clouds that did not display accurate depth data. This is because these two sensors only collect 2D data, so creating a 3D point cloud posed a serious issue. Thanks to efforts from other team members in training the GAN models, we were able to

work around this to an extent. However, on their own, while the subsystems can determine depth difference between objects, they cannot accurately determine the real distance separating them with precision.

5.1.2. Realsense/LiDAR Dataset and Training Limitations

The Realsense and LiDAR sensors were already capable of creating point cloud images alone, so the focus in these systems were point cloud smoothing (Realsense) and densification (LiDAR). As mentioned in the above section, training was extremely limited for each relative GAN model. For the LiDAR point cloud data, this caused point density improvement to be relatively minimal. The GAN model would also layer points for flat objects (such as walls) slightly off from the Z axis, causing flat surfaces to appear incorrectly textured. For the Realsense, the main limitation stemmed from using user created data from the sensor itself as the “best case” scenario image data, then removing even more data to use as the less accurate set. This created one obvious issue, neither of the images in the dataset were “perfect” representations of the ground truth. Due to this the GAN model was unable to correct for major swaths of artifacting and could only really correct minor instances of inaccuracies.

5.1.3. IMU Limitations

One of the major limitations for our project was the role for which the IMU was supposed to play. While it did exceedingly well at mapping the rotations of our sensors and accurately transforming their rotations in Unity it failed to reliably map their x, y, and z linear movements. This issue is one commonly noted with single IMU 3D position tracking. Due to the nature of how one gets position data via double integration of the IMU’s accelerometer’s data, any slight noise in linear acceleration data compounds quickly leading to wild inaccuracies in position after just a few moments. Due to this limitation, it was decided that the IMU would be used for only rotational angle data capturing and that the sensors would be spun along a single controlled axis for 360 degree environment scanning. Locking the IMU like this came with its own set of issues as well. With this setup, any slight tilt of the sensor apparatus would lead to misalignment of future point cloud frames leading to gaps and overlapping.

5.1.4. Processing Power Limitations

During this project one of the continuous limitations that arose time and time again was that of computational power. First, in 403 with GAN model training, running a dataset of 1000 images through 50 epochs would take anywhere from 8 - 16 hours depending on the hardware specifications of the individual team member’s computer. This, while more of a time rather than an operational hindrance, resulted in slowed efforts to improve GAN model performance. The other and far more serious computational limit was that of the laptop responsible for hosting Unity and three of the four python servers. During initial testing of the RealSense sensors integration into Unity, it was discovered that the transmission of full fidelity, sub-second, JSON packages resulted in an immediate freeze and crash of Unity. To combat this, transmission delays and downsampling factors were added to each python server to allow for precise tuning of data sending. Each sensor ended up with a four times downsample and anywhere from a three to five second delay, with most of the variation in time due to correcting for concurrency in rendering time to have inputs synchronized.

5.2. Impacts

The main goal of this system was to provide first responders with a visual of environments in emergency situations to prevent injury and death of said responders. Impacts of the system include:

- First responder safety and effectiveness: by delivering a quality real-time 3D reconstruction of hazardous environments, first responders can assess structural damage, locate victims, and identify dangers without physically entering the situation. This reduces risk of exposure, improves decision-making, and helps overall mission outcomes.
- Exploration of inaccessible environments: In situations such as cave rescues, or sea floor mapping, access and visibility prove to be difficult. Integrating sensor data through GAN enhancement yields complete models of complex spaces providing researchers and emergency responders with reliable information to make assessments of various situations.

5.3. Other Applications

While this project was essentially designed to keep first responders safe in various environments, the final product can be utilized in various applications. It can create quality environment models which can be developed for autonomous vehicles, infrastructure inspection, and construction surveying. It could also create rapid production of 3D maps to provide remote monitoring of facilities, warehouses, or other structures enabling faster decision making and assessments in various scenarios.