B.Tech.-Project

on

SLAM FOR DRONES

(Project Code: D12)

Submitted by

PAVAN SAI TEJA (2017ME10583)

SHASHANK (2017ME10572)

CHAITANYA RAM (2017ME10576)

Supervised by

Prof. Subir Kumar Saha

DEPARTMENT OF MECHANICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY, DELHI

AUG 2020

ACKNOWLEDGEMENT

We thank our project supervisor Prof. S.K. SAHA, Dept of Mechanical Engineering, Indian Institute of Technology, Delhi for his valuable comments and suggestions that greatly helped in improving the quality of work.

We thank our Btech Project Co-ordinator Prof R.K. Pandey, Dept of Mechanical Engineering, IIT Delhi for his painstaking efforts to guide us throughout our research work.

We thank Prof. Sunil Jha, Dept of Mechanical Engineering, Indian Institute of Technology, Delhi for his valuable advice and help during our research work.

We thank all our professors for their valuable comments after reviewing our work, special thanks to Prof.J.P.Khatait, Dept of Mechanical Engineering, Indian Institute of Technology, Delhi

We wish to extend our special thanks to all our colleagues and friends who helped directly or indirectly to complete our research work.

We extend our thanks to our parents and all our family members for their unceasing encouragement and support at home who wished us a lot to complete this work.

Shashank Dammalapati
Chaitanya Ram
Pavan Kota

ABSTRACT

Developing and testing algorithms for autonomous vehicles in real world is an expensive and slow process. In this project we will present a new simulator (AirSim) built on Unreal Engine that provides realistic simulations for implementing Autonomous algorithms. [5] AirSim is an open-source cross-platform simulator for autonomous vehicles It supports hardware-in-the loop (HITL) and software-in-the-loop (SITL) with popular different fight controllers. The simulator is designed to be extensible to accommodate new types of vehicles, sensors, and software protocols. The modular design of AirSim enables various components to be easily usable independently in other projects and 3D environments. With the wide application of LIDAR applications in many fields, the LIDAR based SLAM technologies has also developed rapidly. We demonstrate LIDAR based SLAM algorithms by first configuring a drone with LIDAR sensor and map the 3D simulator environment in different 3D environments/scenes.

TABLE OF CONTENTS

LIST OF FIGURES & TABLES

CHAPTER 1

Introduction

1.1 Background

The Future is coming, and we cannot stop it now. We will be having flying machines all above us very soon transporting goods and even people. We must design our machines so that we stay save from these flying machines because they will be in the open environment unlike a Metro where if any, the damage will be restricted to the metro rail and not to the others moving on the road or staying in their home; or the roadways where enough research has been done on the safety measures, now there is a considerable amount of automation already in our cars and vehicles. The problem with flying vehicles like drones is that they are intrinsically very unstable, and if there is any catastrophic failure, they become much more insecure and unstable. To solve this problem, we must bring Autonomy to the drones -The most important part of this will be the awareness of the surroundings. This can be achieved by Simultaneous Localization and Mapping. This will allow the drones to map the environment continuously and help itself navigate through the obstacles. Also, SLAM will help us map places for general purpose usage like mapping architecture, art structure.

Figure1: Drone @ Hockey Ground, IITD

1.2 Problem Overview

To test out these SLAM algorithms, simulation is a better alternative than real world testing considering cost, speed, and ease of implementation. Currently, this is a non-trivial task as simulated perception, environments and actuators are often simplistic and lack the richness or diversity of the real world. For example, for robots that aim to use computer vision in outdoor environments, it may be important to model real-world complex objects such as trees, roads, lakes, electric poles and houses along with rendering that includes finer details such as soft shadows, specular reflections, diffused inter-reflections and so on. Similarly, it is important to develop more accurate models of system dynamics so that simulated behaviour closely mimics the real-world. AirSim is an open-source platform that aims to narrow the gap between simulation and reality to aid development of autonomous vehicles. The platform seeks to positively influence development and testing of autonomous algorithms

Most robotics researchers work on ROS for its community support, ROS runs on Ubuntu machines. There is no option for researchers to work in a windows environment for developing and testing their algorithms. AirSim is an attempt, but there are no clear libraries for implement SLAM. We bridged this gap by creating a data retrieving algorithm that can put the lidar data into a format that the MATLAB navigation toolbox can use to implement Simultaneous Localization and Mapping.[1]

Figure2: Generated Map (Pink) overlayed on the actual map built in the simulator.

CHAPTER 2

Literature Survey

For developing SLAM using simulation, there is a vast collection of simulators, addressing all of them exhaustively is beyond the scope of this paper. A few popular selections that are close to our work are mentioned here.

1. For research work, Gazebo was one of the most popular simulation platforms. It has a modular nature that allows various physics engines, sensor models to be used and 3D worlds to be built. Gazebo goes beyond rigid monolithic body vehicles and can be used with link-and-joint architecture to mimic more general robots, such as complex manipulator arms or biped robots. While Gazebo is fairly feature rich it has been difficult to create large scale complex visually rich environments [1]

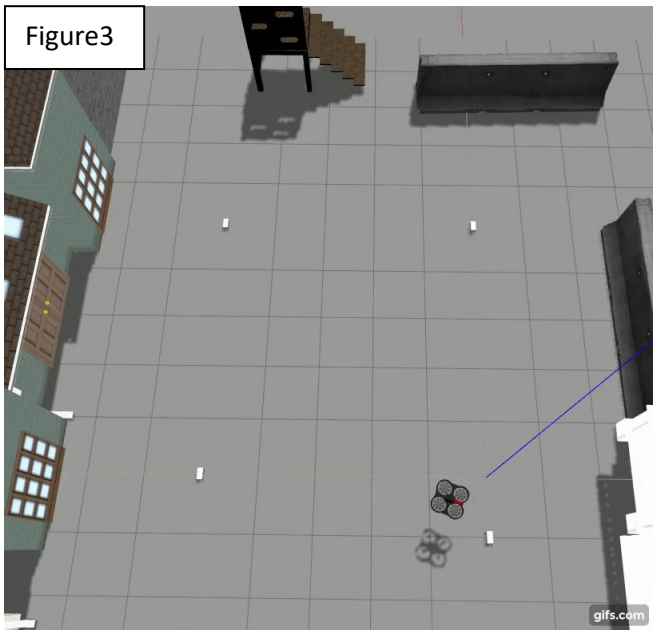


Figure 3: Drone in Gazebo Environment [11]

2. AirSim: For autonomous vehicles that are closer to the real world, high-fidelity visual and physical simulation has lagged various advances in rendering techniques created by platforms such as Unreal Engine or Unity.

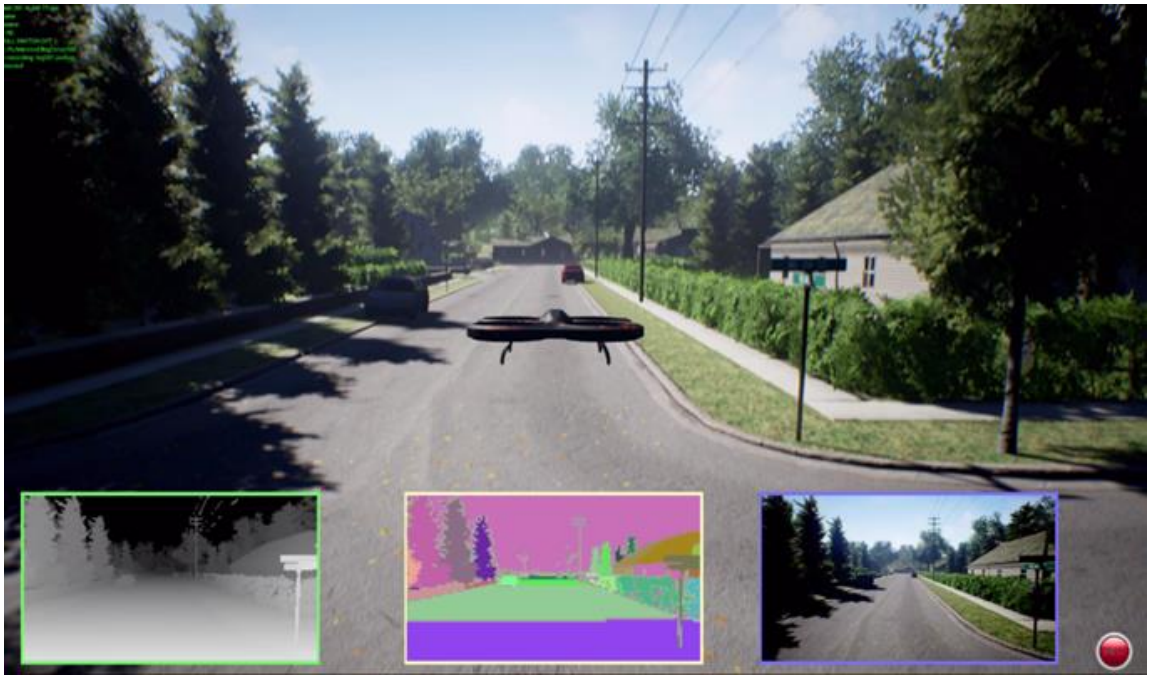


Figure 4: Drone in AirSim simulator [10]

3. Other notable efforts include Hector, which focuses primarily on tight integration with popular ROS and Gazebo middleware. Similarly, Rotors [3] offers a modular framework for designing micro aerial vehicles and creating control and state estimation algorithms that can be tested in simulator simulators.. Finally, jMavSim [4] is easy to use simulator that was designed with a goal of testing PX4 firmware and devices.

Figure 5: Pixhawk Computer and Mission Planner Ground Control Station, [12]

We settled to use AirSim for its rich API support for interaction between Companion computer and the Simulation engine. This simulation uses a quadcopter equipped with various sensors and cameras. The possible sensors and cameras in our Microsoft AirSim simulated drone are. [5]

• inertial measurement unit (IMU) • global positioning system (GPS) • barometer • one dimensional distance sensor • 2D-lidar • 3D-lidar • regular camera

For plotting we used:

1. Matplotlib _ for quick verifications
2. MATLAB _ for plotting the Pose Graph

Matplotlib is a python library used to plot data. This library gives us functionality like MATLAB and is easily accessible just by importing the module. But, for implementing SLAM we need more advanced plotting software.

MATLAB is a combination of a desktop environment designed for iterative processes of analysis and design and a programming language that can handle objects of the array type, such as LiDAR data. The Navigation Toolbox that can run Pose Graph Optimization on Lidar data is also available from MATLAB.

Most robotics researchers work on ROS for its community support, ROS runs on Ubuntu machines. There is no option for researchers to work in a windows environment for developing and testing their algorithms. AirSim is an attempt, but there are no clear libraries for implement SLAM. We bridged this gap by creating a data retrieving algorithm that can put the lidar data into a format that the MATLAB navigation toolbox can use to implement

CHAPTER - 3

PROJECT OBJECTIVES AND WORK PLAN

Problem Definition/Motivation: Currently to implement a SLAM algorithm for autonomous navigation in real world, we must fetch different sensors like Lidar, Vision sensors, etc. But for regular university researchers it becomes impossible as cost of these sensors are way beyond their budget.

In our project, we ventured to find a 3D environment simulator with configurable spaces, robots, and sensors for testing algorithms in any user defined environments. This allows us to verify and debug correctness of our algorithms for SLAM in different situations/environments. This reconfigurability of spaces and environments is not easily possible in real world testing. Simulation makes testing cycle much easier.

OBJECTIVES OF THE WORK

Simultaneous Localization and Mapping (SLAM) in robotics is the computational problem of creating or updating a map of an unknown world using sensors while keeping track of the position of the robot inside it at the same time.

The objective of this work is to develop LIDAR based SLAM algorithms that estimate the trajectory of the robot and create a 3D occupancy map of the environment from the 3D Lidar point clouds and previously estimated trajectory.

Using LIDAR point cloud data obtained from 3D simulator, we try to obtain coordinates of the (environment surface feature points) obstacles with respect to drone frame.

We then implement suitable LIDAR based SLAM algorithms to map the environment simultaneously while the robot is moving.

In our project, we are using Microsoft's AirSim project in Unreal Engine to simulate the drone and the environment, which supports hardware-in-the-loop (HITL) and software-in-the-loop (SITL) with various combat controllers, our method makes testing algorithms cheaper and faster

METHODOLOGY

- To simulate 3D environment, we install (latest) Unreal Engine 4.25.4 and build Microsoft's AirSim project that works with Unreal Engine (>4.24.0)
- We setup Python Clients to communicate with Simulation objects from the command line.
- We use Lidar data API to fetch point cloud data from the Unreal Engine simulation environment and map this data using an external python script.
- We implement SLAM algorithms to map the environment simultaneously while the robot is moving.
- We demonstrate the repeatability of our algorithms by applying SLAM in various user defined environments.

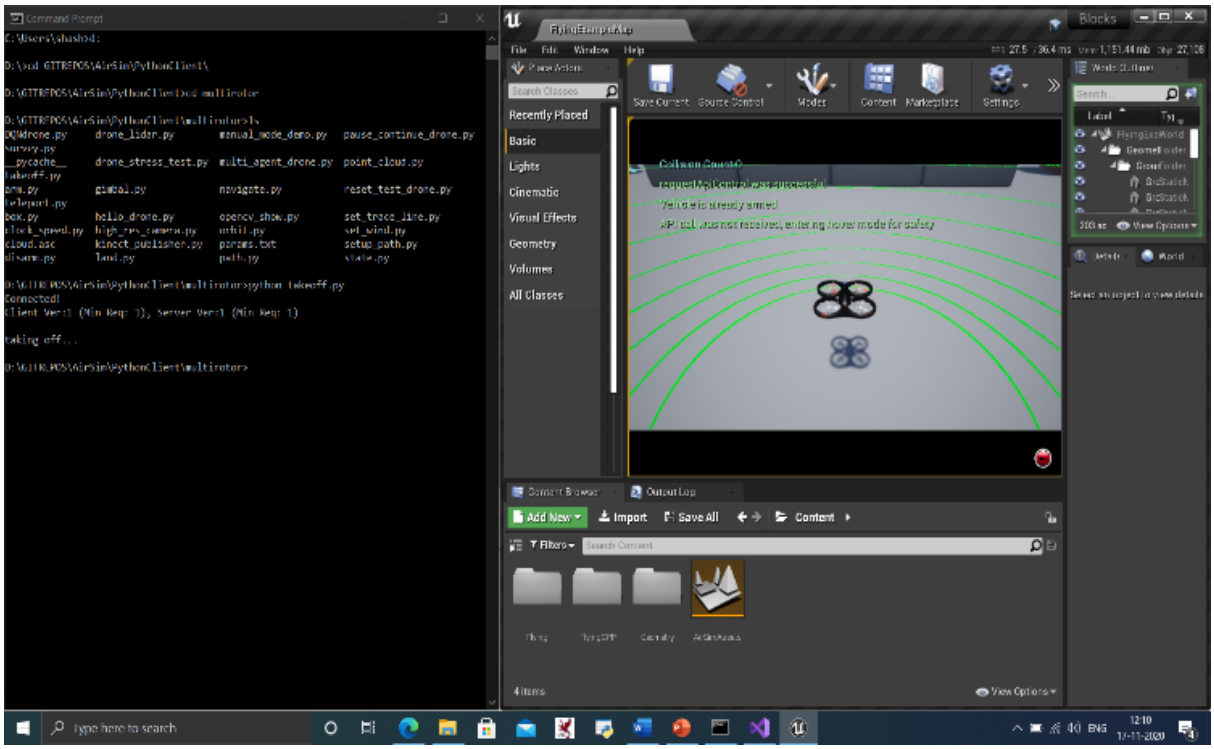


Figure 6: CMD Prompt on left | 3D simulator on the right

Table1

TASK	Week 1-2	Week 3-4	Week 5-6	Week 7-8	Week 9-10	Week 11-12	Week 13-14
Literature Review about SLAM							
Evaluating Ros, Blender+Matlab, Microsoft AirSim							
Setting up AirSim Environment and collecting Data							
Plotting point cloud data and designing projectile of the drone for SLAM							
Studying Lidar SLAM and their implementation							
Testing our final code with multiple 3D environments							
Report Making							

GANTT CHART _ Planned

Table2

TASK	Week 1-2	Week 3-4	Week 5-6	Week 7-8	Week 9-10	Week 11-12	Week 13-14
Literature Review about SLAM							
Evaluating Ros, Blender+Matlab, Microsoft AirSim							
Setting up AirSim Environment and collecting Data							
Plotting point cloud data and designing projectile of the drone for SLAM							
Studying Lidar SLAM and their implementation							
Testing our final code with multiple 3D environments							
Report Making							

GANTT CHART _ Corrected

CHAPTER 4

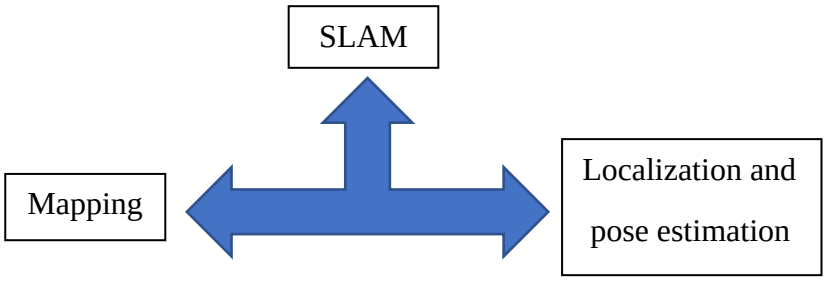
WORK PROGRESS

Relevant Theory (SLAM):

- Simultaneous localization and mapping refer to the navigation of mobile robots, constructing a map of the unknown world in the absence of external referencing systems such as GPS, and simultaneously localizing within this map (SLAM).

- Learning maps under robot pose uncertainty is also often referred to as simultaneous localization and mapping.

- It is an essential skill for navigation of the mobile robots in an unknown environment.



- We use "smoothing approaches" to solve the full SLAM problem (unlike "filtering approaches) to estimate the full robot trajectory and the map of the environment as the robot moves. Usually, they rely on minimization methods of least-square error.

- SLAM uses “mapping algorithms” to build a map and “Localization and pose estimation algorithms” to localize your vehicle in that map at the same time.

- We use lidar- SLAM for implementation, that processes lidar scans to iteratively build a map and optimizes your estimates of robot poses.

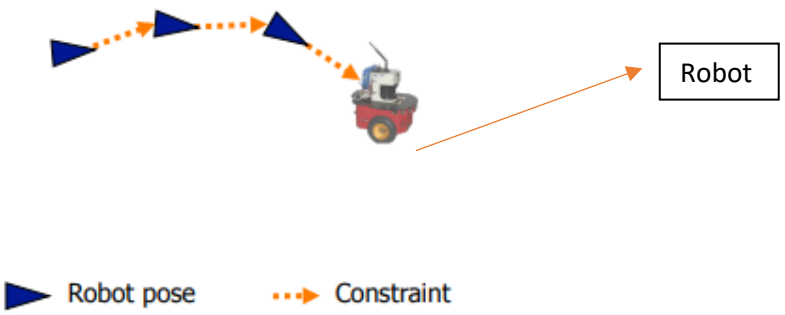


Figure 7 : Relative Poses of a robot moving in a ceratin path and contraints [13]

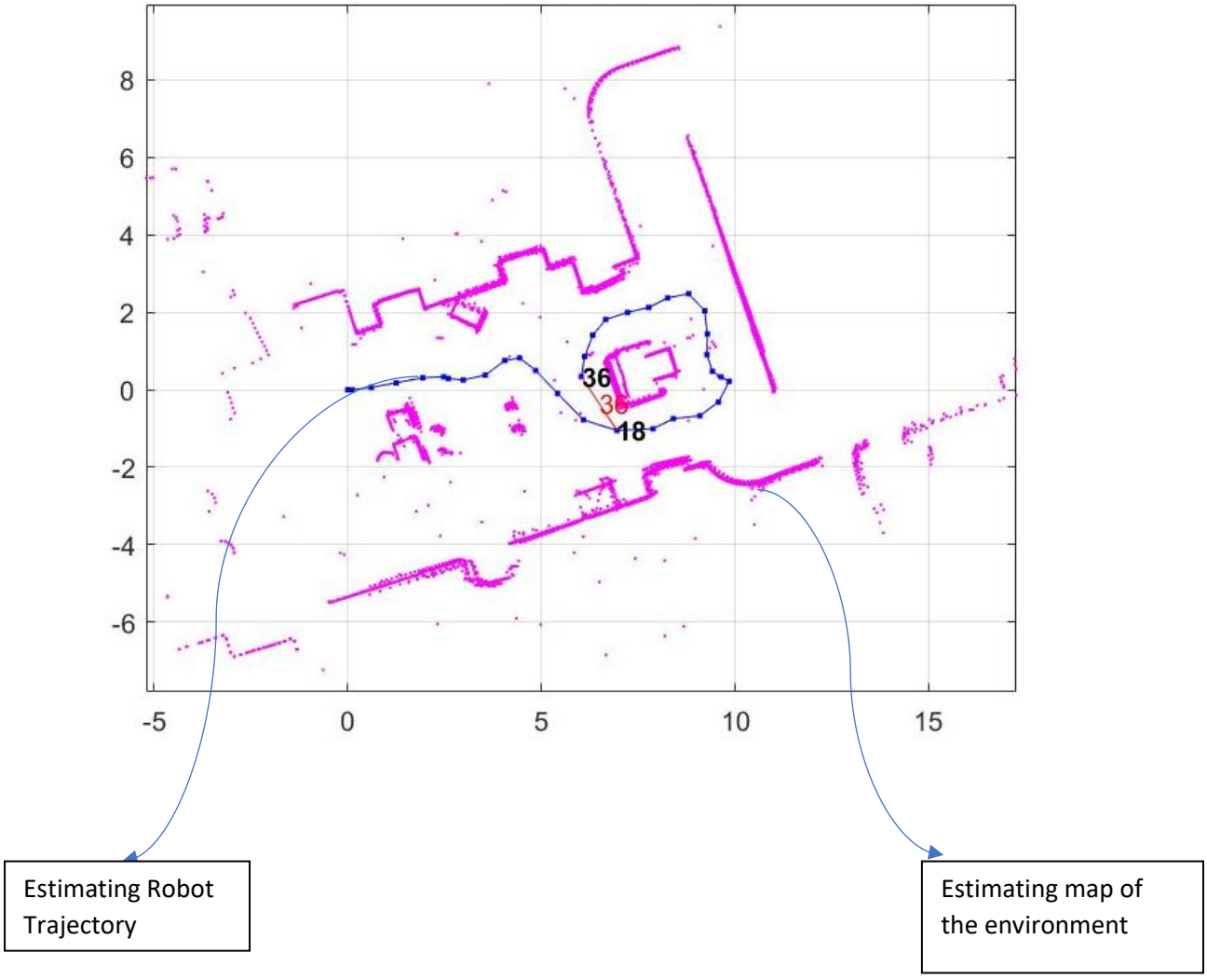


Figure 8: Explain how robot is estimating and optimizing its poses iteratively and understands the map of the environment and localize its position in the map.

Steps in lidar – SLAM Implementation

1. Localization and Pose Estimation Algorithms

(Orients your vehicle in your environment)

1.1 Localization Algorithm

(a) We use "Scan Matching" to estimate the robot pose in a known map using lidar sensor readings.
(b) Matches two laser scans using Normal Distribution Transform (NDT) Algorithm.
(c) The goal of the scan matching is to find the relative pose (or transform) between two robot positions where the scans were taken.
(d) The scans can be aligned based on the overlapping features.

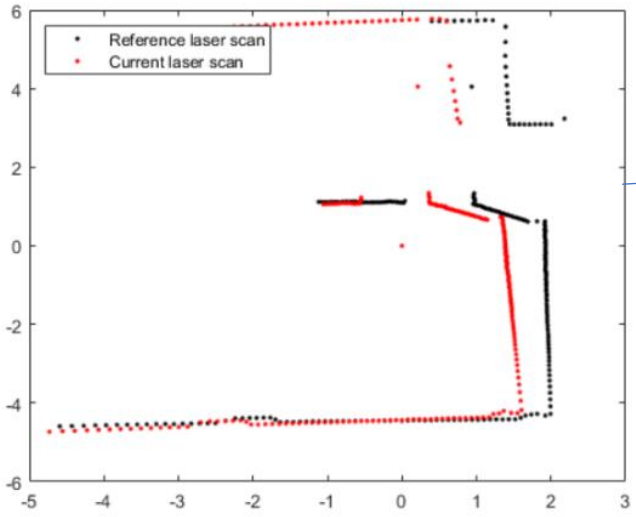


Figure 9: Before Transform

(Original Scans)

Notice there are translational and rotational offsets, but some features still match

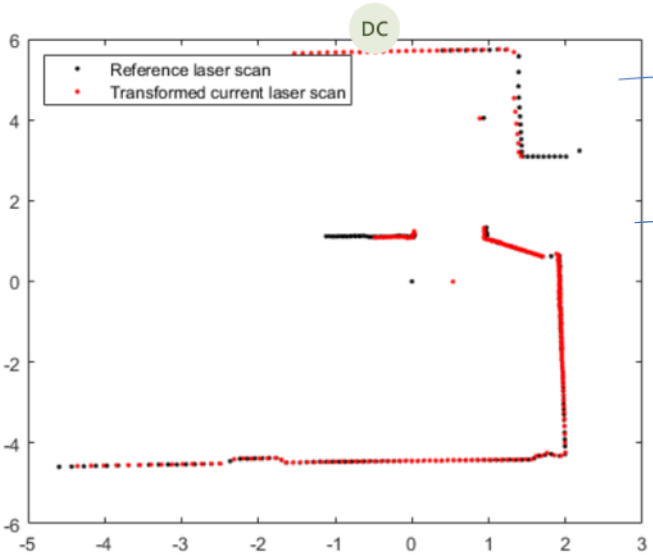


Figure 10: After Transform

This picture shows how scan matching algorithm align the overlapping features of the lidar scan and estimating robot relative pose between those two-robot position.

1.2 Pose Graph Optimization Algorithm (Graph – Based SLAM)

(a) Construct a “Pose Graph” contains nodes connected by edges. These nodes are robot poses and edges are constraints that define the relative pose (found using lidar scan matching) between the nodes and uncertainty on that measurement.

(b) The “pose graph” iteratively track your estimated poses with relative pose edge constraints. That is when a robot moves to another point, the pose graph creates a new node and new edge constraint correspondingly.

(c) Adding an edge (or constraint) between two existing nodes in the pose Graph creates a “loop closure edges” in the graph. The pose graph checks for these loop closure edges to optimize the Pose Graph.

(d) The “Pose Graph Optimization” is simply:
- Constructing the graph from the raw (lidar)sensor measurements and next is,
- Determining the most likely configuration of the poses given the edges(constraints) of the graph (Pose graph optimization algorithm)

(e) The pose graph constructed is heavily dependent on sensor.

(f) We implement “OptimizePoseGraph (pose Graph object)” function that modifies the nodes to account for the uncertainty and improve
the overall graph.

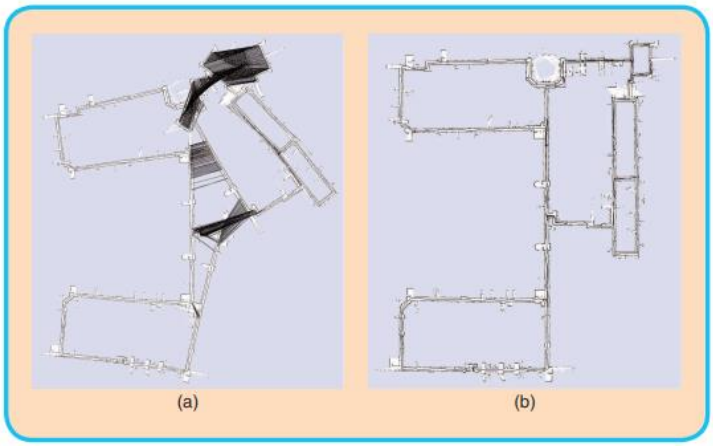


Figure 11: Pose-graph example | dataset recorded (left) and (right) after optimization. [7]

2. Mapping Algorithms

Occupancy maps are used to represent obstacles in an environment and define limits of the world.

- Illustration of Pose Graph optimization in MATLAB

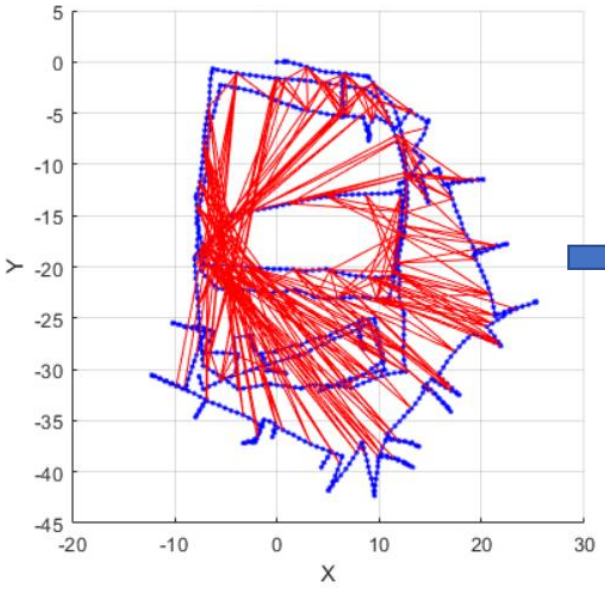


Fig 12: Original Pose Graph

Red lines indicate loop closure edges identified in the data set

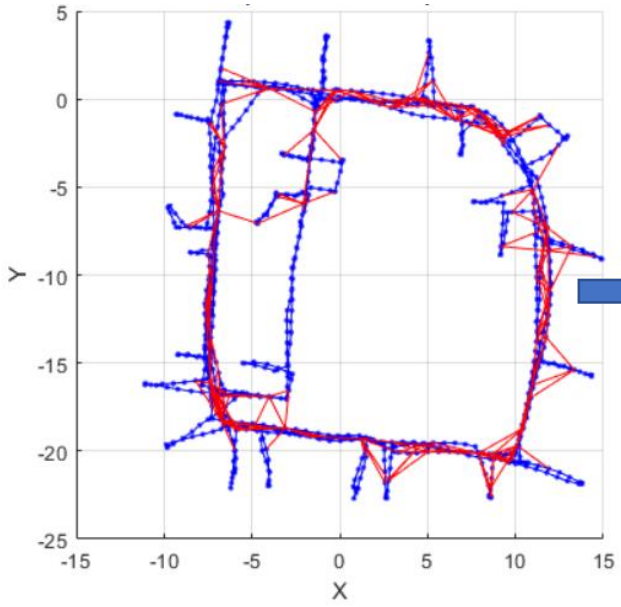


Fig 13: Updated Optimized Pose Graph

- Therefore, from the above example we can appreciate how pose graph optimization algorithm updates the robot poses and maps the environment almost correctly.

- Lidar-SLAM (Lidar based simultaneous localization and mapping) is built around the optimization of a 2D pose graph

Description of the problem/case study

For Autonomous Navigation, we need

1. SLAM
2. Navigation
3. Motion Planning

In this project we are only addressing SLAM (Mapping and Localization). SLAM is the first step for building autonomous robots because without awareness of the environment robots can neither Navigate nor plan its trajectory

What is SLAM (Simultaneous Localization and Mapping)?

- Computing the robot's poses and the map of the environment at the same time
- Localization: estimating the robot's location
- Mapping: building a map
- SLAM: building a map and localizing the robot simultaneously

We are implementing the SLAM in Windows Environment using AirSim and MATLAB.

Exp. Setup

Components:

1. Windows 10 _ computer
2. Unreal Engine Simulator
3. AirSim Plugin for Unreal Engine
4. Python | Programming IDE
5. MATLAB

Procedure: We are simulating a drone and its environment in Unreal Engine's AirSim project

1. Download Unreal Engine
2. Build AirSim Project plugin
3. Setup API control for Companion computer and the Simulation Engine
4. Fetch Sensor Data using API Layer as shown in the Figure [3].
5. Adding LiDAR sensor to the drone
6. Import Matplotlib module to Python development and use it plot the data fetched from simulated Lidar sensor from the Unreal Engine's AirSim project environment.
7. Import Lidar data into Excel sheet for building the pose graph and the map
8. Setup MATLAB code for creating the pose graph and the map

Unreal Engine Setup

- Download Epic Launcher
- Install Unreal Engine (Unreal Engine is Open Source and Free)
- Unreal Engine >= 4.25 is recommended for our project for its support for AirSim plugin

AirSim Setup

- Clone AirSim repository to your computer using the command

```
git clone https://github.com/Microsoft/AirSim.git
```

- It is recommended to clone the package to drive other than D or E to avoid build errors
- Build the AirSim project using the command

```
build.cmd
```

- This command will create plugins for the Unreal Engine Simulator

API Setup

- AirSim provides us with pre-programmed python APIs to retrieve data from the simulated drone programmatically.
- For this we need to have the python packages “msgpack”, “airsim”, “numpy”, “opencv-python”
- Install the libraries using the following commands in the command prompt.

```
pip install msgpack-rpc-python
pip install numpy
pip install airsim
pip install opencv-python
```

Fetching API data

- Using API is easy, we can simply get required drone data and send instructions using python commands. For example

`client.takeoffAsync().join()` : Makes the drone takeoff from the ground

`client.moveToPositionAsync(n,e,d,v).join()` : Moves the drone to the specified location with the speed (v)

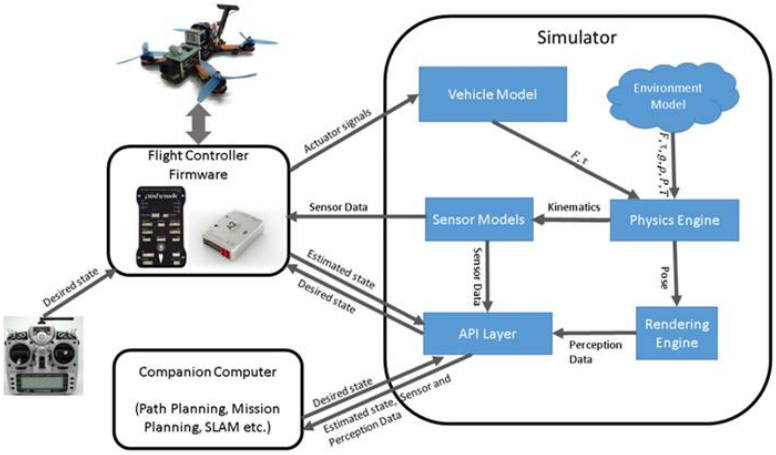


Figure 14 _ AIRSIM ARCHITECTURE

Adding LiDAR sensor to the drone

- By default, the drone is not equipped with a LiDAR scanner, we have to add the sensor through the 'settings.json' file that is located in the Documents/AirSim folder.

Matplotlib Setup

- We used Matplotlib to visualize the lidar data directly. This helped us reduce the development time by eliminating porting of data to MATLAB to visualize.
- Install MATPLOTLIB using the command

  `pip install matplotlib`
- After installing the required packages and setting up the APIs, the below given code should plot the retrieved LiDAR data from the simulation

```
points = self.parse_lidarData(lidarData)
print('lidar', str(i+1), ' points: ', points)
state = self.client.getMultirotorState()
print("state: %s" % pprint.pformat(state))
print('total lidar points in lidar range ' + str(len(points)))
x = points[:,0]
y = points[:,1]
z = points[:,2]

ax = plt.axes(projection='3d')
ax.scatter3D(X, Y, Z, color = "green")
#ax.plot_trisurf(X, Y, Z,cmap='viridis', edgecolor='none')
plt.show()
```

Importing LiDAR data to excel

- In this project we are working on 2D LiDAR scan data.
- We are collecting 50 to 100 lidar scans depending upon the size of the map and the resolution required.
- LiDAR settings:

```
"LidarSensor1": {
  "DataFrame": "SensorLocalFrame",
  "DrawDebugPoints": true,
  "Enabled": true,
  "HorizontalFOVEnd": 180,
  "HorizontalFOVStart": -180,
  "NumberOfChannels": 1,
  "Pitch": 0,
  "PointsPerSecond": 5000,
  "Roll": 0,
  "RotationsPerSecond": 10,
  "SensorType": 6,
  "VerticalFOVLower": 0,
  "VerticalFOVUpper": 0,
  "X": 0,
  "Y": 0,
  "Yaw": 0,
  "Z": -1
}
```

- LiDAR data is retrieved in a 2D Array format, handling that kind of a data can be done using pandas and write the data frame to Excel
- Install the required libraries using the command

```
pip install pandas
pip install xlsxwriter
```

- Now write the LiDAR data into an Excel Workbook with each lidar scan in a new excel sheet using the below code.

```
# The below code opens a new Excel workbook with the name 'scan.xlsx'
writer = pd.ExcelWriter('scans.xlsx', engine = 'xlsxwriter')
df = pd.DataFrame([x,y])
# The below code opens a new Excel sheet in the above workbook with the name 'sheet_scan(number)'
df.to_excel(writer, sheet_name = 'sheet_scan(number)'])
writer.save()
```

MATLAB for SLAM

- MATLAB Navigation Toolbox has algorithms built in for motion planning, SLAM, and navigation.
- We used the toolbox to import our LiDAR data and ran SLAM algorithm
- This algorithm has two main parts: Pose Graph and Map
- Run the Slam Algorithm to build pose graph and map

Results and Discussion

- AirSim (3D Environment) Installation – Complete
- Added Lidar to the drone as shown in the Figure 15

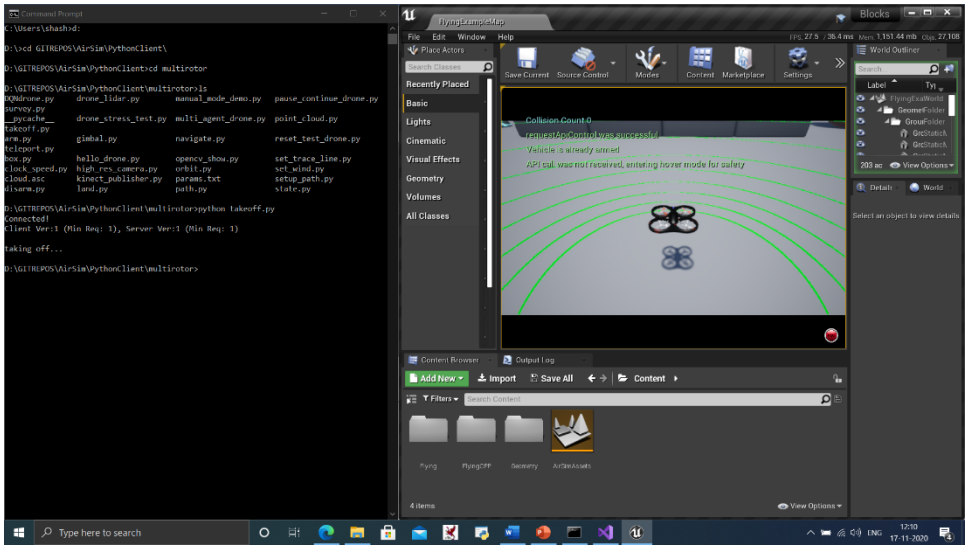


Figure 15: Drone with Lidar Sensor

- Constructed multiple environments for testing the lidar SLAM as shown in Figure 16

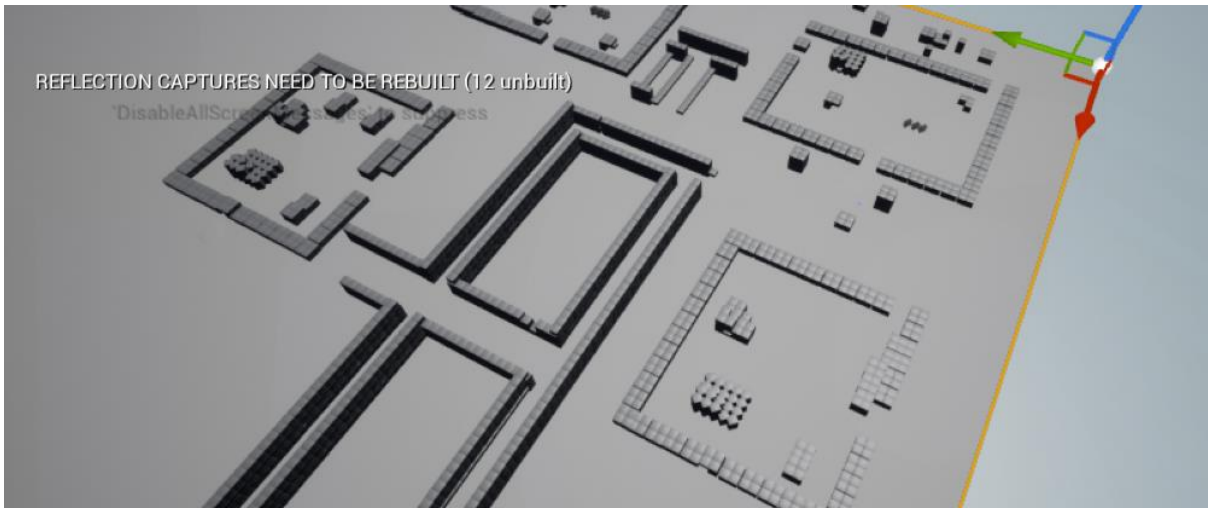


Figure 16: Construction of multiple environments in AirSim for testing

- Retrieved LiDAR data from simulation into the Excel sheets as shown in the Figure

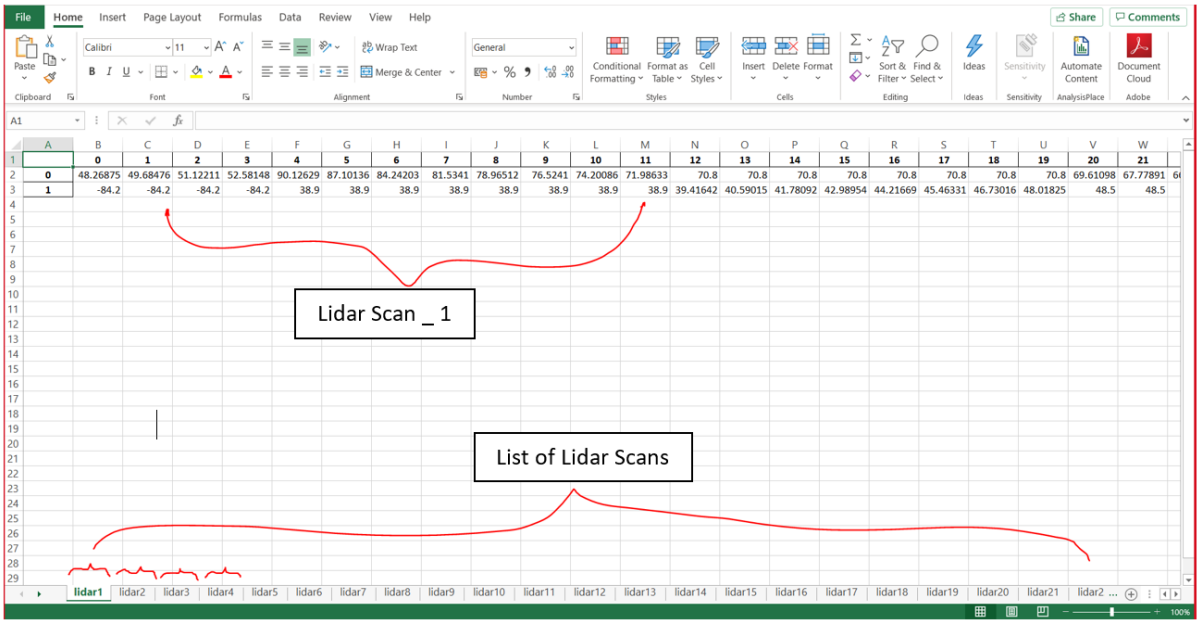


Figure 17: Lidar Data collected into Excel workbook with each Sheet holding each Lidar Scan

- Imported Excel LiDAR data into file format suitable for MATLAB SLAM Toolbox.
- Performed Pose Graph Optimization and generated map of the world as shown in the Figure

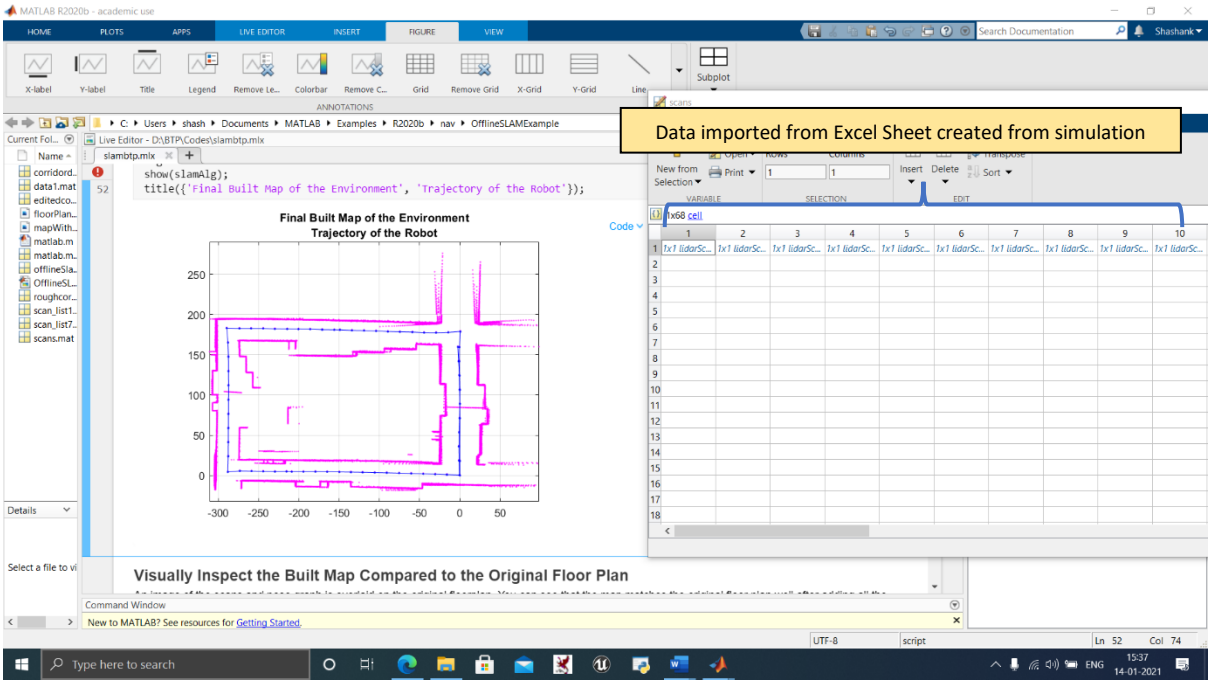
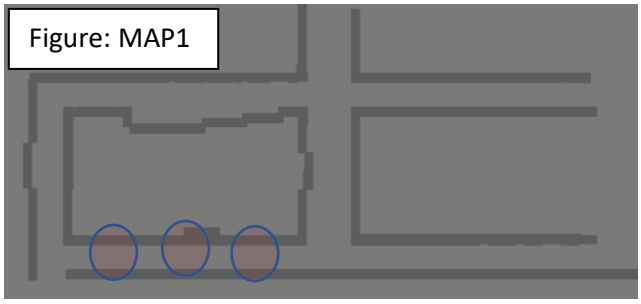


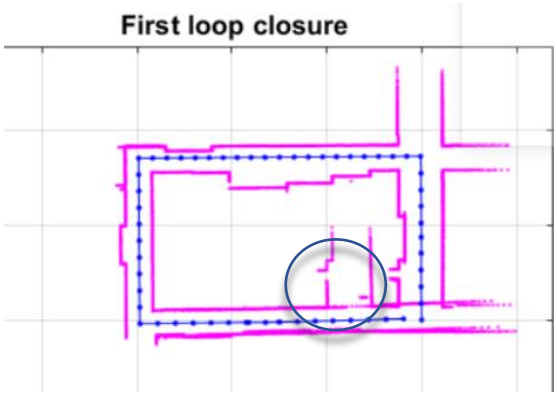
Figure 18: Pose Graph Optimization and Mapping performed in MATLAB

Below, we are attaching the maps and SLAM visualizations done using MATLAB for the environments we created in the AirSim.

EXAMPLE 1

In the below example, although the SLAM was implemeted, we got a ghost map on the botton as shown in the overlayed figure below.





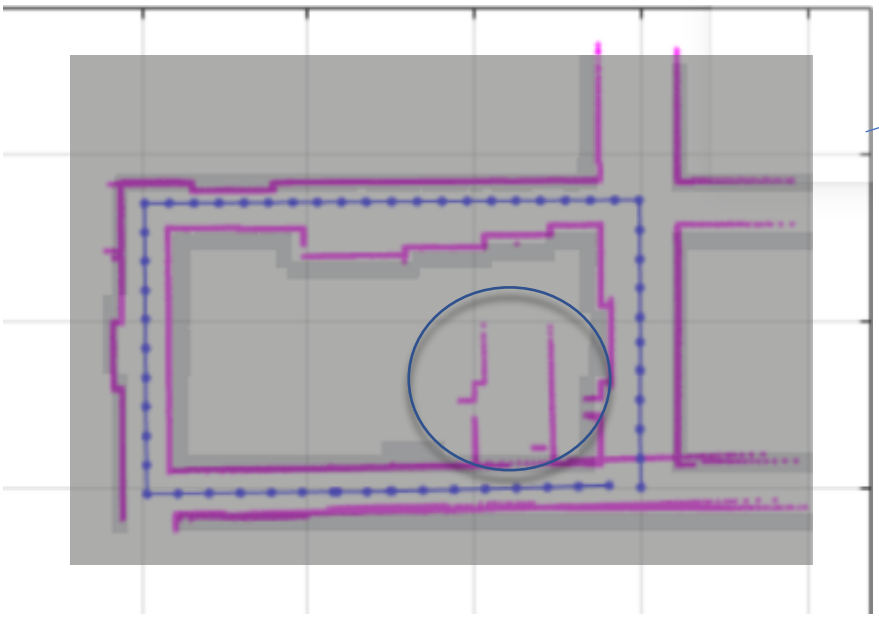


Figure 19: Example 1

Overlaying of Pose Graph and map onto the original Environment

Example 2 (Link to the Map building process: Video 1.mp4 - Google Drive)

In the following example, we changed the map a little bit from the above example. In this case we didn't get any ghost images in the map built. We then continued to build the Occupancy Grid Map.

Figure 20: Example 2

Overlaying of Pose Graph and map onto the original Environment

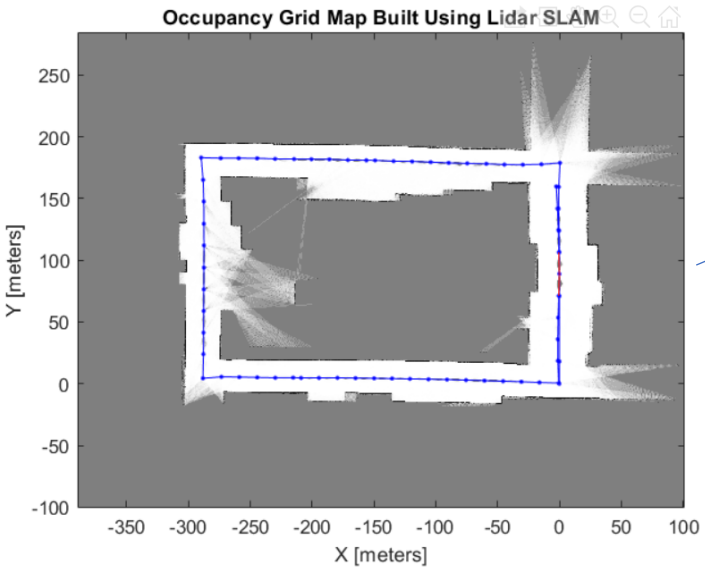


Figure 21: Example 2

Occupancy Grid Map.

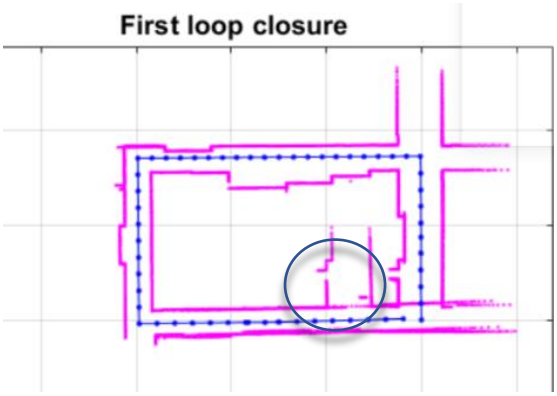
This occupancy grid map shows all the available motion space in White.

Discussion:

The problem with not having direct odometry data is clearly visible in the Example 1 shown above. In (Figure: MAP1), when the robot is moving in the bottom corridor, the robots gets the same LiDAR data at each position located by the red circles due to similarity of the micro-local environment.

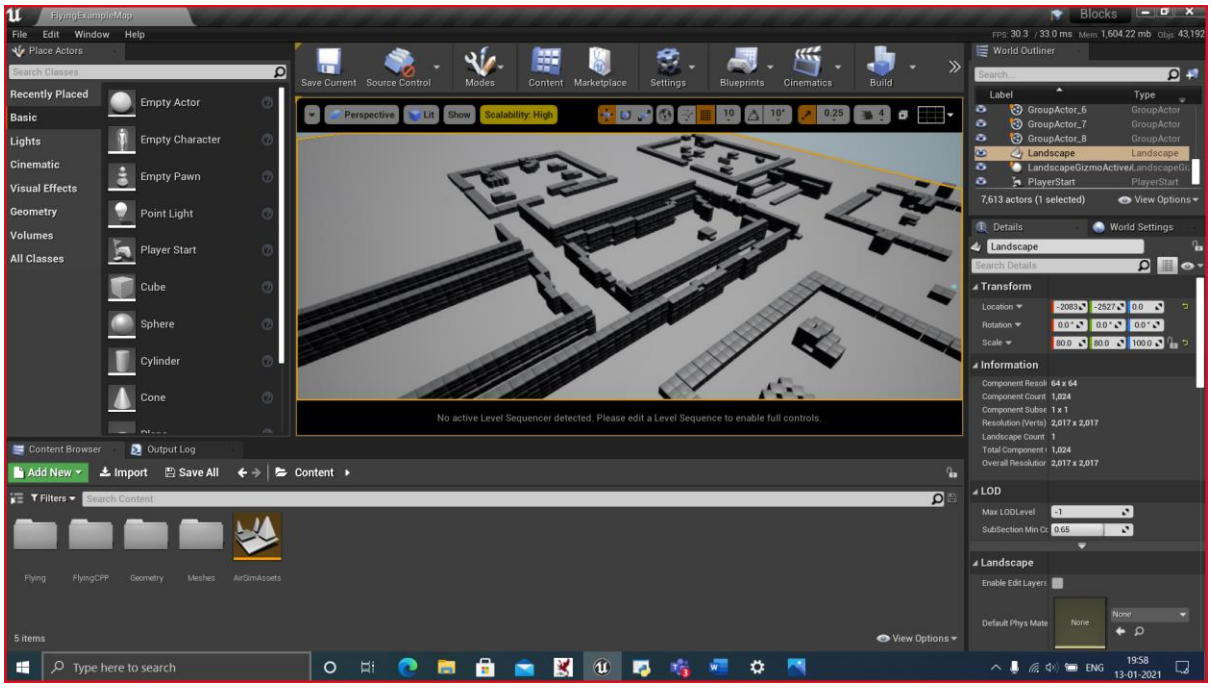
This will have a negative impact on the NDT algorithm that estimates the relative pose between two consecutive LiDAR scans. Due to this, map is built incorrectly as shown below.

Figure: MAP1



We eliminated this problem in Example 2 by modifying the map to eliminate confusion to the NDT algorithm. Ideally, I think we need to use sensor fusion techniques using IMU/GPS data to better estimate the relative poses between individual scans.

MAP OF THE ENVIRONMENT SHOWN IN THE ABOVE EXAMPLES



Most robotics researchers work on ROS for its community support, ROS runs on Ubuntu machines. There is no option for researchers to work in a windows environment for developing and testing their algorithms. AirSim is an attempt, but there are no clear libraries for implement SLAM. We bridged this gap by creating a data retrieving algorithm that can put the lidar data into a format that the MATLAB navigation toolbox can use to implement

CONCLUSION AND FUTURE WORK

After the environment setup, the task at our hands was to figure out a way to extract lidar data perfectly and to understand and implement suitable SLAM algorithm for Lidar data in the simulation. An example would be Graph Based SLAM algorithm.

We implemented the SLAM algorithms to map the environment simultaneously while the robot is moving. We demonstrated the repeatability of our SLAM implementation in various user defined environments.

- Our deliverable is a set of programs that can help implement SLAM in Windows using AirSim and MATLAB which previously was not possible.
- Our deliverable outputs a file like ‘rosbag’ which contains lidar data retrieved from the simulation.

In our project we got ghost images in the Pose Graph/Map built in MATLAB. To eliminate this problem, we can extend our project to take extra IMU data and calculate the odometry in addition with the NDT algorithm that was used in this project.

After the completion of SLAM, we can focus on Navigation, Motion Planning and Trajectory Planning for a complete autonomous package.

Now that we are familiar with SLAM and various simulations, we can develop robust, adaptive, learning and context driven autonomous robotic systems and algorithms without having the sensors and robots using simulations. In case of drones, there is a clear advantage in continuing the research using AIRSIM because in real world, A drone flight lasts less than 10 minutes before its battery goes off. Simulations allows us to test algorithms easily without battery constraints and cost constraints.

We can now focus on developing algorithms that can bringing robotic systems into dynamic and lively human environment without having to risk damage or costs.

REFERENCES

1. Koenig, N., Howard, A.: Design and use paradigms for gazebo, an open-source multi-robot simulator. In: IROS

2. Meyer, J., Sendobry, A., Kohlbrecher, S., Klingauf, U., Von Stryk, O.: Comprehensive simulation of quadrotor uavs using ros and gazebo. In: SIMPAR, pp. 400–411. Springer (2012)

3. Furrer, F., Burri, M., Achtelik, M., Siegwart, R.: Rotorsa modular gazebo mav simulator framework. In: Robot Operating System (ROS), pp. 595–625. Springer (2016)

4. K.S. Chong and L. Kleeman. Feature-based mapping in real, large scale environments using an ultrasonic array. International Journal Robotics Research, 18(1)

5. M. Csorba and H.F. Durrant-Whyte. A new approach to simultaneous localisation and map building. In Proceedings of SPIE Aerosense, Orlando.

6. Cadena, Cesar & Carlone, Luca & Carrillo, Henry & Latif, Yasir & Scaramuzza, Davide & Neira, Jose & Reid, Ian & Leonard, John. (2016). Simultaneous Localization And Mapping: Present, Future, and the Robust-Perception Age. IEEE Transactions on Robotics. 32. 10.1109/TRO.2016.2624754.

7. Biber, P., and W. Strasser. "The Normal Distributions Transform: A New Approach to Laser Scan Matching." *Intelligent Robots and Systems Proceedings*. 2003.

8. Magnusson, Martin. "The Three-Dimensional Normal-Distributions Transform -- an Efficient Representation for Registration, Surface Analysis, and Loop Detection." PhD Dissertation. Örebro University, School of Science and Technology, 2009.

9. Grisetti, G., R. Kummerle, C. Stachniss, and W. Burgard. "A Tutorial on Graph-Based SLAM." *IEEE Intelligent Transportation Systems Magazine*. Vol. 2, No. 4, 2010, pp. 31–43. doi:10.1109/mits.2010.939925.

10. Bailey, T., & Durrant-Whyte, H. (2006). Simultaneous localization and mapping (SLAM): Part II. IEEE Robotics & Automation Magazine, 13(3), 108-117. https://doi.org/10.1109/mra.2006.1678144

11. Fuentes-Pacheco, J., Ruiz-Ascencio, J., & Rendón-Mancha, J. M. (2012). Visual simultaneous localization and mapping: A survey. *Artificial Intelligence Review*, *43*(1), 55-81.

12. (2020). LiDARsim: Realistic LiDAR simulation by leveraging the real world. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. https://doi.org/10.1109/cvpr42600.2020.01118

13. Home - AirSim (microsoft.github.io)

14. autonomous-navigation · GitHub Topics

15. STorM32 Gimbal Controller — Copter documentation (ardupilot.org)

16. https://in.mathworks.com/products/navigation.html