Part B: Data Analysis PDF

Load and Explore Trajectory Data

Felicia had to do a lot of work transferring the data from rosbag files into csv files so that we could work with them in Jupyter Notebook (since the rosbag files contained so much information, we wouldn't be able to run in all in Jupyter Notebook without it crashing). In order to understand the data, we had to do a lot of research since this is something we haven't really worked with before. We looked through all the columns of the csv files and noticed there were 3 different files containing LiDAR information: ouster points, Livox avia points, and Livox mid360 points.

Bullet points we wrote on all the csv files that simply explain what they are and how they relate to the drone:

Ouster Points:

Sensor: ouster LiDAR (multi-beam spinning LiDAR)

- Digital 3D sensor that uses light to measure distances and creates 3D "point clouds" of the environment

Content: point cloud frames - the actual point cloud is in the binary-encoded data column

Data: each row is one full LiDAR frame

- Need to parse data using the fields and point_step info

Use for: dense 360 degree LiDAR scans (obstacle detection)

Livox Avia Points:

Sensor: Livox Avia (non-repetitive scanning LiDAR)

- 3D data capture

Content: point cloud frames - points column contains a decoded list of per-point data (x,y,z, reflectivity, offset_time)

Data: each row is one frame, but the points field holds the actual list of LiDAR hits - easier to parse than ouster because it's not in raw binary

Use for: smaller field of view, but high precision

Livox Mid-360 Points:

Sensor: Livox Mid-360 (360 degree hybrid solid-state LiDAR)

- Hybrid solid-state: a lidar system that combines a non-rotating, solid-state light source with a smaller, dynamically moving scanning component, in this case a rotating mirror, to direct the laser beam and create a 3D point cloud

Content: same structure as avia - difference is that the point_num is 2016 instead of 2400 (number of points per frame), and the lidar_id is 192 instead of 0

There was also a csv file with VRPN pose:

VRPN Pose:

System: VRPN client node (Holybro motion capture system) - how it collects data from a VR device

Content: rigid-body pose estimates (not LiDAR) - pose estimation is a computer vision technique used to pinpoint the location of specific key points

Data: pose.position(x,y,z) gives us position of the sensor in world frame, pose.orientation (x,y,z,w) gives us the quaternion orientation (represents the orientation of a 3D object in space, encodes a unique axis and angle of rotation) - ground truth of sensor

Long story short, the first 3 give us different LiDAR point clouds from different sensors and the 4th tells us where the sensor was when the scans happened. The ouster points csv file wasn't super helpful though because of all of the information in the "data" column. We would need to parse through that column, where each entry has probably thousands of characters and we decided it wouldn't make sense to do that, so we didn't use the ouster points data. We did however use the VRPN pose data and the Livox mid360 data.

Data and Trajectory Plots with explanation

Originally, we tried to visualize the trajectory just using the small section of data we had grabbed from the dataset, which you can see in Figure 1. This didn't give us very useful information though. Since the LiDAR sensor is grabbing points so close together, there was barely a change in movement and the only visual changes in the graph were due to normal noise (which is why it's just a zig zag trail).

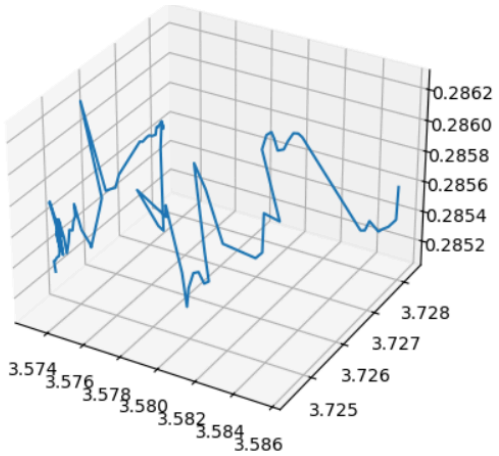

Figure 1: Original plot of trajectory using pose position points

After realizing that the trajectory plot using only a subset of the data is practically useless, we decided to try downloading the entire post data file and then replot it. As you can see in Figures 2 and 3, it now perfectly shows the trajectory of the drone we expected (up and down). There are 2 plots in Figure 3, the first shows the X vs Z coordinates, and the second shows the Y vs Z coordinates. These are helpful because we now know that we have data that accurately represents where the drone is going.

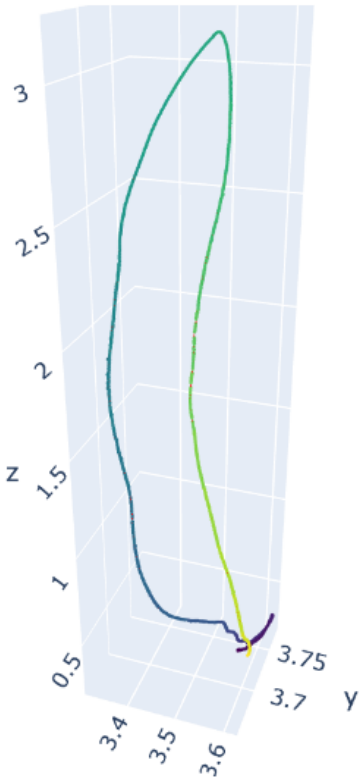

Figure 2: Plot of trajectory using full pose data file

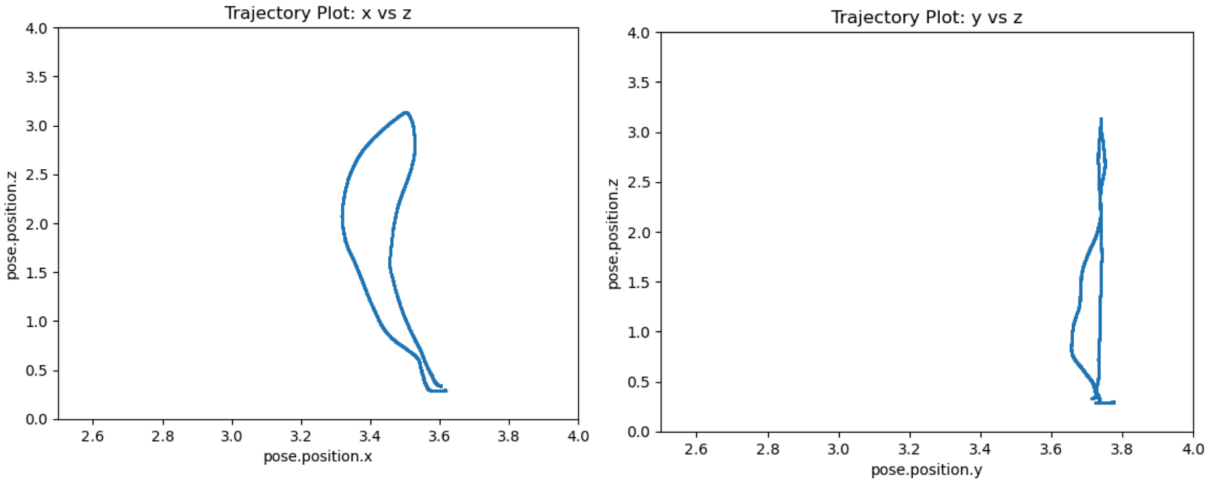

Figure 3: 2D plots of trajectory using full pose data file

We also plotted the 3D point clouds. Those give us information about what the environment that the sensors are picking up looks like. We tried graphing these a couple of different ways, but always ended up with the same block of points with a hole in it, you can see this in Figure 4. After investigating more, this hole could either just be the way the environment is set up, or it could possibly be a blind spot that the sensors are missing. This would be something we want to investigate a little bit more to gain a full understanding of what this means for the modeling.

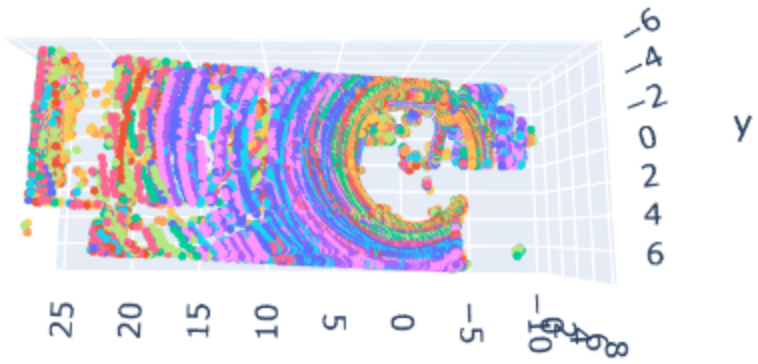

Figure 4: plot of 3D point clouds (not a great representation since it's hard to capture a 3D graph in a screenshot)

Check column types, units, valid ranges, and missing-value patterns
We created Table 1 to easily visualize important information on all the columns in the VRPN pose data. It includes the types, units, and valid ranges of each of the columns. There are no missing values in this dataset, so we didn't have to worry about dealing with that.

Column	Type	Missing_val_count	Unique_Values	Range	Units
Time	<class 'numpy.float64'>	0	3032	(1690899086.439581, 1690899118.0395534)	Seconds
header.seq	<class 'numpy.int64'>	0	3032	(16978, 20144)	Count
header.stamp.secs	<class 'numpy.int64'>	0	33	(1690899086, 1690899118)	Seconds
header.stamp.nsecs	<class 'numpy.int64'>	0	3031	(9456458, 999614419)	Nonoseconds
header.frame_id	<class 'str'>	0	1	(world, world)	Name of Coordinate Frame
pose.position.x	<class 'numpy.float64'>	0	3024	(3.31677794456482, 3.6165285110473633)	meters
pose.position.y	<class 'numpy.float64'>	0	2998	(3.654858112335205, 3.77560043334961)	meters
pose.position.z	<class 'numpy.float64'>	0	3031	(0.2850619852542877, 3.134616613388061)	meters
pose.orientation.x	<class 'numpy.float64'>	0	3032	(-0.0059188581071794, 0.0228217169642448)	X Vector Component
pose.orientation.y	<class 'numpy.float64'>	0	3032	(-0.0257593318819999, 0.0142646925523877)	Y Vector Component
pose.orientation.z	<class 'numpy.float64'>	0	3032	(-0.1681881099939346, 0.0233441442251205)	Z Vector Component
pose.orientation.w	<class 'numpy.float64'>	0	2226	(-0.9999764561653136, -0.9855958819389344)	Cosine of 1/2 Rotation angle

Table 1: Table of the types, missing values, unique units, and range of the pose columns

Document naming conventions; resolve ambiguous fields
The datasets use standard ROS (Robot Operating System) naming conventions. For example, Time is actually split into secs and nsecs, which together make up a ROS timestamp. To make things easier, we combined them into a single Time column in seconds. The frame_id field identifies the reference coordinate frame (always "world" here), which confirms consistency across the data. As mentioned earlier, some fields, like data in the Ouster file, contain binary-encoded point cloud information that would need special ROS parsing tools, so we didn't

use those. In the Livox mid360 dataset, fields such as lidar_id and point_num tell us which sensor the data came from and how many returns were in each scan. These are more like metadata than physical measurements. By writing out these details, we cleared up the confusing parts and nailed down how each field will be used later for modeling.

Propose a feature set for supervised learning (labels: collision/near-miss/normal) and RL
For supervised learning tasks, we are planning on using a mix of continuous and discrete features taken from the pose and LiDAR data:

- Continuous features:
    - Drone position (pose.position.x, y, z) and orientation (pose.orientation.x, y, z, w)
    - LiDAR point cloud coordinates (x, y, z)
    - Motion derivatives: velocities and accelerations (if derived from pose or IMU data)
- Discrete features:
    - reflectivity (intensity of LiDAR return, related to object type/surface)
    - point_num (number of returns per frame)
    - lidar_id (sensor identifier)
    - tag and line fields
- Response variables (labels):
    - collision, near-miss, normal (based on scenario logs or annotations)
    - IMU data (angular_velocity, linear_acceleration) can be used as inputs for RL reward shaping

For reinforcement learning, the continuous state (pose, orientation, recent LiDAR points) provides the drone with spatial awareness, while discrete ensures sensor-specific context. The action space would correspond to drone movement commands. Rewards could be structured around maintaining safe trajectories (penalizing collisions, rewarding smooth paths, etc).

What we learned and what we are wanting to work on
We mostly worked with the dataset where the drone is simply going up and down. We noticed that the X and Y coordinates stay pretty much the same the whole time, so we want to try and use more data from the spiral dataset. Hopefully this will give us more diverse information that we can translate to the model and better teach the drone. We obviously need to complete the model piece so that will be something we will continue to work on. We also are going to work on combining what we found with what Sam and Jacob have been working on.

Contributions
Felicia Huffman and Olivia Gette worked on this part of the project. We worked individually to try to understand the dataset and then came together and shared what we had learned. From there we were able to make more progress by building off of each other's code and helping each other when we got stuck.
Felicia also found a separate dataset to combine with the mid360 dataset (using the timestamps column) so that we had more xyz coordinates to plot. She also downloaded the complete pose data so that we could get a more accurate visualization of the trajectory. Both Olivia and Felicia did similar work regarding the code, we both worked on visualizing the trajectory and plotting the

3D cloud points to understand what that is showing. We both worked through multiple different ways to plot the data in the most useful and informative manner. Felicia also started working on the preprocessing and model fitting code.