LiDAR Bike Data Assessment

Dr Tsai, Mohammadi Mohsen, Quentin Fitte-Rey

March 14, 2025

1 Context

We have a LiDAR bike equipped with multiple sensors that collect a variety of data, which we aim to use for detecting and localizing objects in the environment. The data available to you includes:

- **Images**: Captured from a camera mounted on the bike.
- **LiDAR Point Clouds**: 3D point data from the LiDAR sensor.
- **GPS Data**: Location data with timestamps.
- **IMU Data**: Inertial measurement unit data with timestamps.

Additionally, you have access to a file containing all the necessary data to compute the intrinsic and extrinsic matrices for the camera and LiDAR sensors.

2 Assessment Tasks

The assessment consists of three mandatory steps, to be completed in sequence, and an optional bonus step. Below, each step is detailed with specific instructions.

2.1 Step 1: Reading Sensor Data

Your first task is to write code to read the data from the provided files. The data is structured as follows:

- **Images**: Located in a folder named `images`. Each image is stored as a binary file with this encoding:
  - First 4 bytes: Width of the image (uint32)
  - Next 4 bytes: Height of the image (uint32)
  - Remaining bytes: Raw image data (RGB, uint8)
- **LiDAR Points**: Located in a folder named `lidar`. Each point cloud is stored as a binary file with this structure:

  - Sequence of 16-byte points, each containing:
    * 4 bytes: X coordinate (float32)
    * 4 bytes: Y coordinate (float32)
    * 4 bytes: Z coordinate (float32)
    * 4 bytes: Intensity (float32)

- **IMU Data**: Stored in a JSON file named `imu.json`, containing a list of measurements with timestamps and sensor readings.
- **GPS Data**: Stored in a JSON file named `gps.json`, containing a list of measurements with timestamps and location data.

You need to write functions to read and parse these files into usable formats, such as NumPy arrays for images and LiDAR points, and dictionaries or dataframes for IMU and GPS data.

2.2 Step 2: Synchronizing Sensor Data

The next step is to synchronize the data from all sensors. Since the sensors operate at different frame rates, you must align the data based on timestamps. Here's how to proceed:

- Identify the sensor with the slowest frame rate (likely the camera or LiDAR).
- For each frame from this slowest sensor, find the closest corresponding data from the other sensors based on their timestamps.
- For sensors with slower frame rates, such as GPS, consider interpolating the data to estimate values at the timestamps of the slowest sensor.

Implement a function that retrieves the maximum frame containing all available information. The synchronization will be determined by the slowest sensor, likely between the LiDAR and the camera.

2.3 Step 3: Creating a Lightweight API

For the final mandatory step, create a lightweight API (use whatever python package you prefer) with the following functionality:

1. Allow a user to send the name of a folder containing the sensor data (you only have one video as example). The code should automatically synchronize the data and the API should return a JSON file containing all synchronized information for each frame. For images and LiDAR points, include only the paths to their corresponding files in the JSON.

The JSON output should provide a comprehensive, synchronized dataset for each frame based on the slowest sensor's timestamps.

2.4 Bonus Step: Projecting LiDAR Points onto Images Using the KITTI Dataset

Note: Originally, this bonus step was designed to use the LiDAR bike dataset with its transformation matrices to project LiDAR points onto images. However, since the transformation matrices for our data are not yet ready, we will pivot to using the KITTI dataset instead. The KITTI dataset provides well-calibrated, synchronized LiDAR and camera data, making it a suitable alternative for this task.

As an optional bonus step, you will enhance the API to visualize LiDAR points projected onto images using the KITTI dataset. This involves:

- Adding an API endpoint where a user can send a frame number, with the API returning the path to a newly created image showing the LiDAR points overlaid on the corresponding camera image.
- Using KITTI's calibration matrices to transform 3D LiDAR points into 2D image coordinates.
- Visualizing the projected points on the image.

Data is provided as follows:

- `image_02/data/`: Contains left color camera images.
- `lidar/data/`: Contains LiDAR point clouds in binary format, where each point is stored as four floats: x, y, z, and reflectance.
- `calib/calib_velo_to_cam.txt`: Provides a rotation matrix R and translation vector T from LIDAR to camera 0 coordinates.
- `calib/calib_cam_to_cam.txt`: Contains intrinsic matrices (e.g., K_{02} for camera 2) and transformations between cameras (e.g., R_{02}, T_{02} from camera 0 to camera 2).

In KITTI, the images in image_02 are typically rectified images.

I understand that this bonus step might be more difficult therefore, you can request at any time before the deadline (expect a certain time of response, if you request it sunday at 6p.m. i will not provide it to you) a hint on the step to follow to do all the transformation necessary.

3 Deliverables and Deadline

To complete this assessment, you are required to provide the following:

- **Code**: Submit all code developed for the assessment. The code should be functional and well-structured.

- **Documentation**: Include at least a README file with clear instructions on how to run the code, including any dependencies, setup steps, and usage examples.
- **Live Review**: Your code will be reviewed in a live session. Be prepared to demonstrate and explain its functionality.

You have approximately more than one week to complete this assessment. The deadline for submission is **March 23 at 11:59 PM**. Please send me a ZIP or RAR file via email (qfitterey3@gatech.edu), or provide a GitHub link (please send the link to the exact last commit). Please CC Dr. Tsai (james.tsai@ce.gatech.edu) and Mohsen (mohsen@gatech.edu).