Camera-Lidar Sensor Fusion

Purpose

This document outlines the motivation, design, and sample code for the sensor fusion module.

Motivation

This module is responsible for figuring out the class, position, and dimension of the objects (e.g. pedestrians, piers, cars, etc.) in the camera image frame. The module only deals with one camera and one lidar sensor.

Requirements

The sensor fusion node must:

- Receive bounding box topics (that contain class and position of bounding boxes) from the yolo module.
- Receive camera image topics from the camera.
- Receive point cloud topics from the Ouster lidar.
- Output the class, position, and dimension of objects.
- Operate at minimum of 10hz

Code Description

- ## Code Summary

A large portion of the code is based on lidar_camera_calibration. To get started, make sure install dependencies.

```
sudo apt install ros-DISTRO-camera-calibration
```

Also, make sure you have the ROS bag file in 'lidar_camera_calibration/bagfiles' folder. In case you want to run the code with live lidar and camera topics, make sure that you have packages to run the sensors. For Ouster lidar, refer ouster_example for sample codes on how to run the sensor. The code description in this document assumes that you use a bagfile, not live streaming data.

The package contains the following functions :

1) Play a rosbag file
2) Run camera calibration, update the rosbag file with calibrated camera coefficients, and display it
3) Run camera-lidar calibration, and display the point cloud projection on the camera image.
4) Run camera-lidar sensor fusion and display the estimated position & dimension of the objects

5) Misc.: Update the timestamps of specific topics to match them to those of other topics

• Play a Rosbag File

The launch file below will play a rosbag file

```
roslaunch lidar_camera_calibration play_rosbag.launch
or
roslaunch lidar_camera_calibration play_rosbag.launch bagfile:=/path/to/file.bag
```

The above one will play a default rosbag file (you can set it in the launch file.)

• Calibrate a Camera

The camera we use doesn't require the camera calibration since it is already calibrated (i.e. the image topic from the camera is not distorted.) In case you change the camera for the autonomous rover and it requires the camera calibration, follow the instructions at lidar_camera_calibration

• Camera-Lidar Calibration and Display Projection

To perform the sensor fusion, it is necessary to know the correspondence between the camera and the lidar frame. To achieve this, this script will run the camera-lidar calibration using the matplotlib GUI. You can pick corresponding points in the camera and the lidar frames. **Make sure that the timestamps of three topics - camera info, image and point cloud - are aligned.** If they are not, run 'update_timestamp.py' to update the timestamps as described below.

Command lines are the following :

```
roslaunch lidar_camera_calibration play_rosbag.launch
rosrun lidar_camera_calibration calibrate_camera_lidar.py --calibrate
```

Press [ENTER] to launch the GUIs and pick the corresponding points. I used a rectangular plate and selected the four corner points of the plate in both the camera and lidar frames. You may update the point cloud field-of-view to display in 'extract_points_3D' function in 'calibrate_camera_lidar.py'

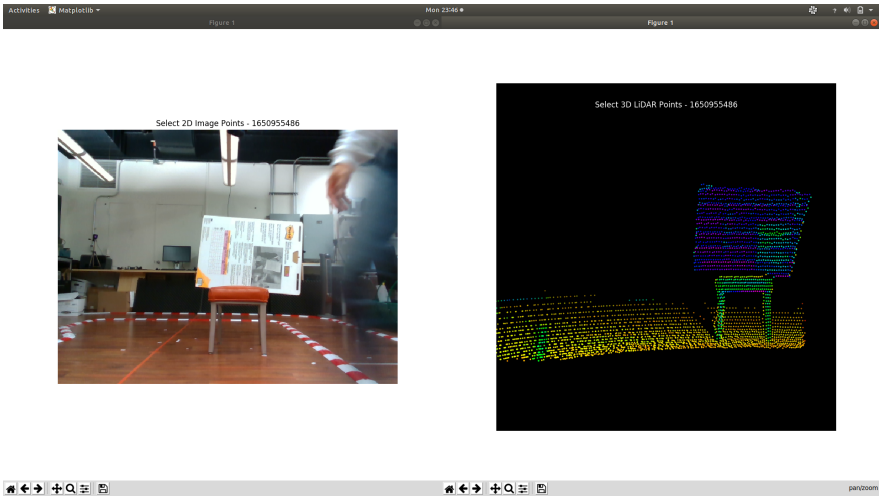

Fig 1. Matplotlib GUIs that help you with the calibration

When you press [ENTER], you will see the two matplotlib GUI like the fig1.

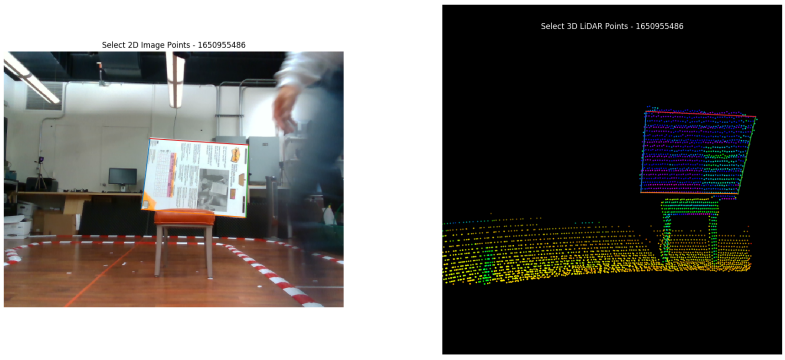

Fig 2. Select corresponding points in two GUIs

Select corresponding points (by click) like the fig2 and close the GUI. To gain enough data for the calibration, repeat the process.

OpenCV's PnP RANSAC + refinement using LM is used to find the rotation and translation transforms between the camera and the LiDAR. Once you have the minimum number of required correspondent points, you will see the numbers like fig3 on the terminal. As you can see, the following calibrated extrinsics are obtained: Euler angles (RPY rad), translation offsets (XYZ m), and rotation matrix.

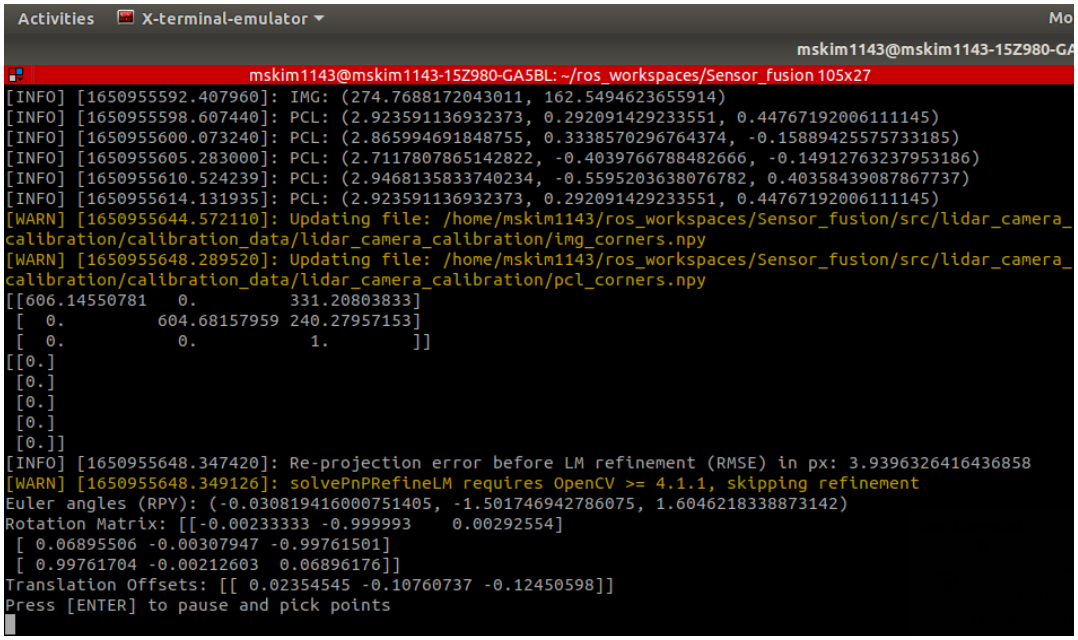

Fig 3. Calibration extrinsic parameters result

Put the Euler angles and Translation offsets at launch files (display_obstacels.launch, display_camera_lidar_calibration.launch) as arguments of "tf2_ros" package in X Y Z Y P R order.

To see the result of the camera-lidar calibration, run the command line below

```
roslaunch lidar_camera_calibration display_camera_lidar_calibration.launch
```

You may update the point cloud field-of-view to display.

- # Camera-Lidar Sensor Fusion

The purpose of this module is to find out the objects' classes, positions, and dimensions near the vehicle. display_obstacles.launch file runs the sensor fusion algorithm to find out the information and displays it on the matplotlib plot figure. The command lines are :

```
roslaunch darknet_ros yolo_v3.launch
roslaunch lidar_camera_calibration display_obstacles.launch
```

The first command runs the object detection module (Yolo) that outputs the topic of the bounding boxes. This may change depending on the progress of the object detection module.

The main nodes in display_obstacles.launch are below :

1) Play_rosbag.launch
   Plays rosbag file. **If you run the launch file with live data streaming from the lidar and camera, delete this and add nodes or launch files to run the camera and lidar.**

2) static_transform_publisher
   Publish the parameters of transformation between the frames.
3) calibrate_camera_lidar.py
   Implement the sensor fusion algorithm. Publish the information (class, position, dimension) of objects. Refer to Obstacles.msg for more details about the form of object information topic.
4) display_obstacles.py
   Based on the object information topic from the above node, draw the estimated position, dimension, and class of each object on the matplotlib plot figure. In the current version, it also draws all of the point clouds in the bounding boxes.
5) image_view
   Shows the image topic on the screen

You can see the result of the sensor fusion module at Sensor Fusion Result
Below is a further explanation of the sensor fusion algorithm. Please read it if you are interested.

**** Sensor Fusion Algorithm ****

From the previous steps and the object detection module, now the system has the projected point cloud and bounding boxes for the objects that indicate the position of each object in a camera image frame. Our goal is to estimate the objects' position and dimensions by processing point cloud data in the bounding boxes. Using all point cloud data in the bounding box to extract the objects' positions and sizes will be like the below figure.

Fig 2.4: All projected point cloud data in the bounding box

In most cases, most bounding boxes contain point cloud data that are not from the corresponding object. For example, in the figure above, there are point clouds on the ground, not the person. The aspects that distract the estimator from the accuracy are the following: *a) point cloud data on the ground, b) point cloud noise and outliers, and c) disparity between the exact boundary of the objects and bounding boxes.*

A filtering algorithm to resolve the problem is like the following; *1) filter the ground, then, for each bounding box, 2) filter the low and high quarter of point clouds in terms of their projected x-coordinate and y-coordinate, respectively, 3) filter the low and high quarters based on their Euclidean distance from the LiDAR, and 4) cluster the point clouds from the closest point cloud data.*

**** Details about the fourth step**: From the filtered point cloud from the third step (set P), 1) add the closest point (point A) to the cluster set (set Q). 2) add point clouds in the set P to the cluster if the distance to point A is less than the threshold. 3) Repeat the second step for newly added point clouds (these are the point A this time) 4) If there is no point cloud newly added, return the cluster.

The a) problem is solved by the 1) step, the c) problem is solved by the 2) step, and the other steps resolve the b) problem. As a final step for the estimation, calculate the objects' positions and dimensions by subtracting the minimum value from the maximum value of the filtered point cloud data. Fig 2.5 is a snapshot of the sensor fusion module, which shows the bounding boxes and projected point cloud in the boxes and estimated the objects' positions and dimensions.

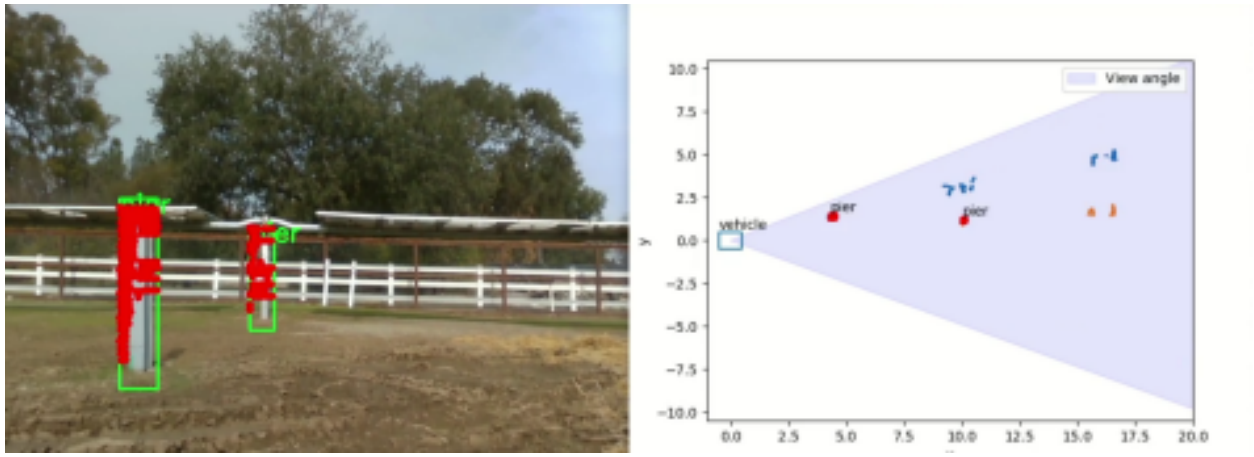

Fig 2.5: Filtering algorithm result. The left image shows the bounding boxes (green) and the corresponding projected point clouds (red dots). The right figure shows the position of the point clouds (blue and orange dots) on the x-y plane (top-down view) and the estimated positions and dimensions of the objects (small red boxes).

As you can see, even if the point clouds in each bounding box have outliers and noise, the algorithm filtered them and extracted the piers' information.

Problems & Future Developments

The algorithm does not run fast enough. Also, since there is no ground truth, the estimation of the algorithm's accuracy has not been implemented.