Final Project

EECE5554 Robot Sensing and Navigation

Yichu Yang

*This is an indivial project

Content involved in Slides

PART 1. Lidar Localization and Mapping with MATLAB

- 1.1 Implementing ICP registration
  - In the Iterative Closest Point or, the reference, or target, is kept fixed, while the other one, the source, is transformed to best match the reference. (wiki*)
  - The main task of ICP is to find a transformation to minimize a cost function:

$$f(R,T) = \frac{1}{N_p}\sum_{i=1}^{N_p} \left| p_t^i - R \cdot p_s^i - T \right|^2$$

in which, p_s is point from source point cloud and p_t is the closest point to p_s in the target point cloud, which can be found using KD-Tree or other methods.

*https://en.wikipedia.org/wiki/Iterative_closest_point

ICP registration in MATLAB

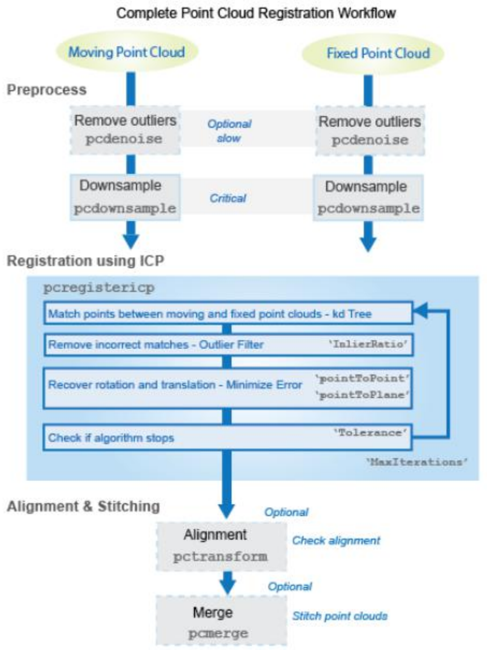


Fig.a is a typical workflow of ICP registration in MATLAB, it is very simple and the functions are well packed. One important step is downsampling of the original point cloud, otherwise we will get much longer calculation time and larger error.

We can feed the previous transformation matrix(affine3D) as an initial guess to allow faster convergence since the movement of vehicle is supposed to be continuous.

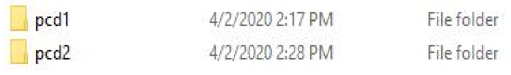
*https://www.mathworks.com/help/releases/R2019b/vision/ref/pcregistericp.html

Data Set Information

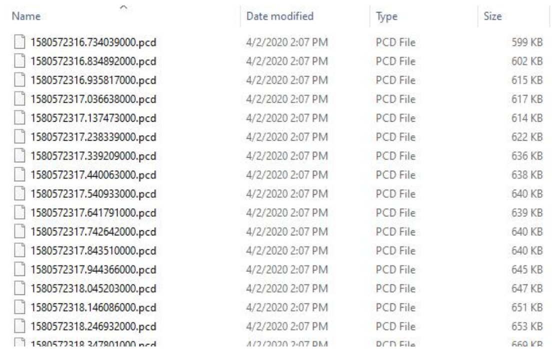
- **morning_stereo_rgb_ir_lidar_gps.bag** **length: 602s**
- **containing topics:**

• /camera_array/cam0/camera_info	5252 msgs (not used)
• /camera_array/cam0/image_raw	5252 msgs (not used)
• /camera_array/cam1/camera_info	5252 msgs (not used)
• /camera_array/cam1/image_raw	5252 msgs (not used)
• /flir_boson/camera_info	36120 msgs (not used)
• /flir_boson/image_raw	36120 msgs (not used)
• /imu/imu	60237 msgs
• /ns1/velodyne_points	5973 msgs
• /ns2/velodyne_points	5972 msgs
• /tf	15086 msgs (not used)
• /tf_static	1 msg
• /vehicle/gps/fix	597 msgs

These two are lidar topics in the structure of sensor_msgs/PointCloud2, they are converted into pcd files which can be found under the work source directory inside ./pcd1 and ./pcd2.



pcd1	4/2/2020 2:17 PM	File folder
pcd2	4/2/2020 2:28 PM	File folder

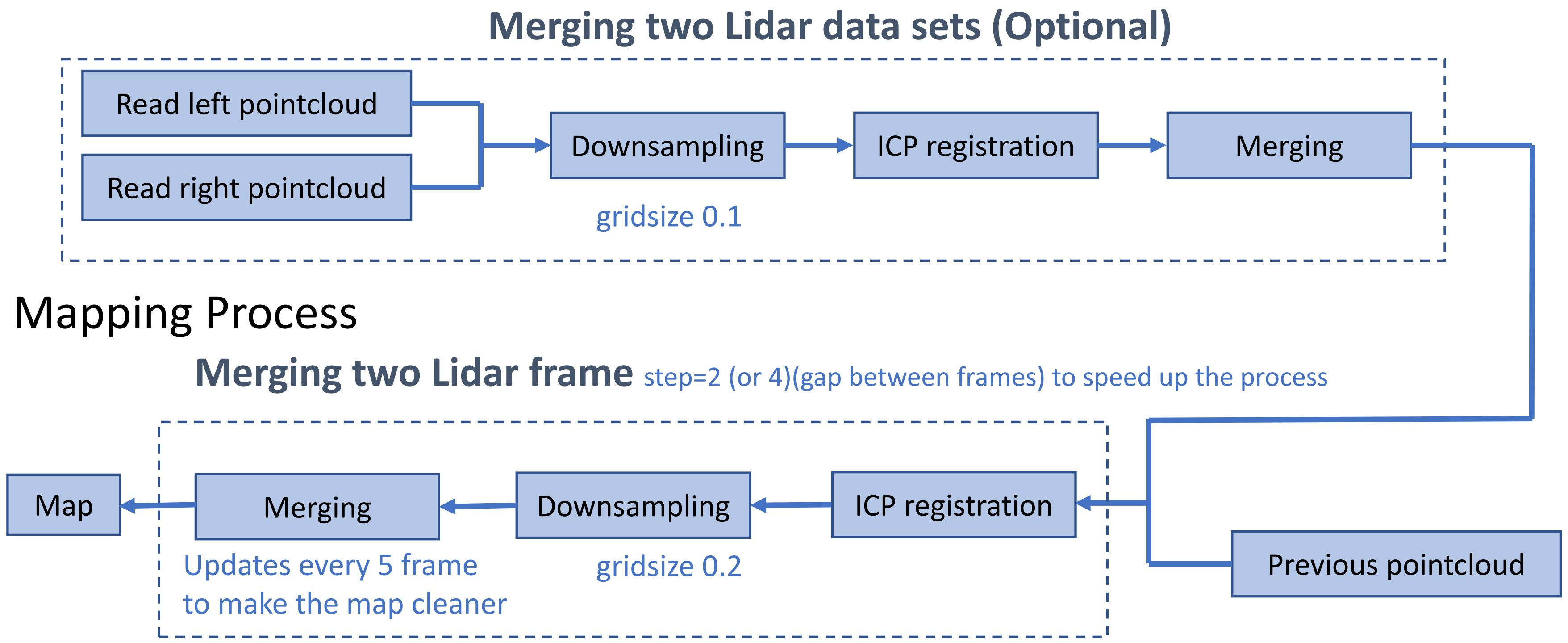


Name	Date modified	Type	Size
1580572316.734039000.pcd	4/2/2020 2:07 PM	PCD File	599 KB
1580572316.834892000.pcd	4/2/2020 2:07 PM	PCD File	602 KB
1580572316.935817000.pcd	4/2/2020 2:07 PM	PCD File	615 KB
1580572317.036638000.pcd	4/2/2020 2:07 PM	PCD File	617 KB
1580572317.137473000.pcd	4/2/2020 2:07 PM	PCD File	614 KB
1580572317.238339000.pcd	4/2/2020 2:07 PM	PCD File	622 KB
1580572317.339209000.pcd	4/2/2020 2:07 PM	PCD File	636 KB
1580572317.440063000.pcd	4/2/2020 2:07 PM	PCD File	638 KB
1580572317.540933000.pcd	4/2/2020 2:07 PM	PCD File	640 KB
1580572317.641791000.pcd	4/2/2020 2:07 PM	PCD File	639 KB
1580572317.742642000.pcd	4/2/2020 2:07 PM	PCD File	640 KB
1580572317.843510000.pcd	4/2/2020 2:07 PM	PCD File	640 KB
1580572317.944366000.pcd	4/2/2020 2:07 PM	PCD File	645 KB
1580572318.045203000.pcd	4/2/2020 2:07 PM	PCD File	647 KB
1580572318.146086000.pcd	4/2/2020 2:07 PM	PCD File	651 KB
1580572318.246932000.pcd	4/2/2020 2:07 PM	PCD File	653 KB
1580572318.347801000.pcd	4/2/2020 2:07 PM	PCD File	660 KB

Algorithm work flow:

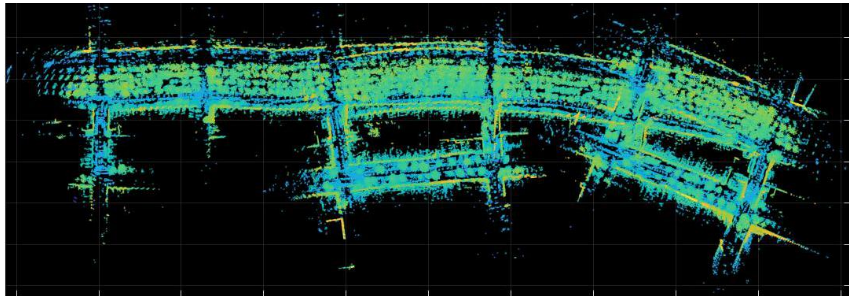
Localization Process(using only lidar)

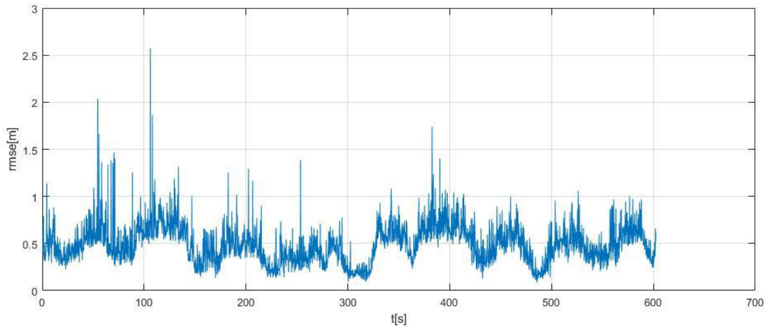
Mapping Process



1.2 Results & Problem analysis

1.2.1 Results of ICP only:


Final Map using ICP only (step=2)



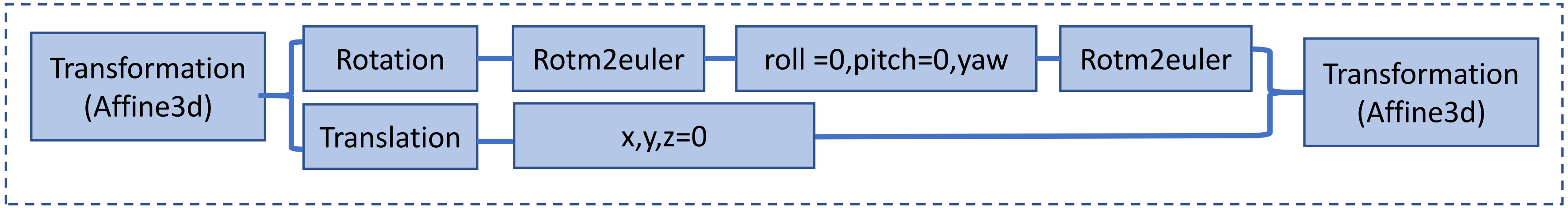
RMSE

As you can see the final map using ICP only is still heavily influenced by yaw drifting. (Because when doing merging the transformation matrix we use is the accumulated transformation matrix, which is updated using accumTform = affine3d(tform.T * accumTform.T);) Yet the RMSE is not changing a lot (around 0.6m), which indicates the large deviation mainly comes from error accumulation.

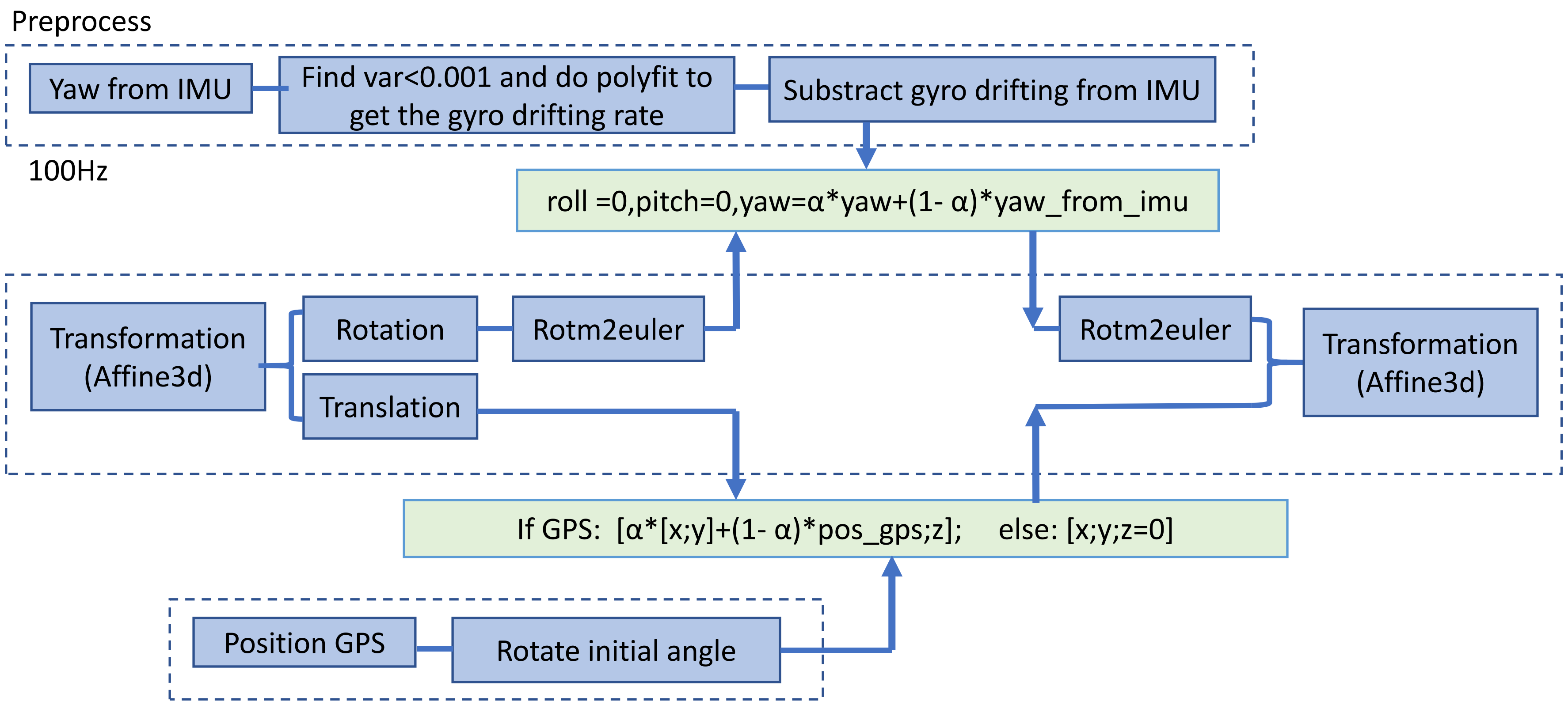
To solve that issue we some help of extra sensors: GPS and IMU. By merging these sensors we can improve the results.

- Main Problems:
  - Map generated using ICP only will drift as the error from every merged frame accumulates: rotation error and translation error
  - roll and pitch angle drifting will cause the map to curve up.
  - Yaw error will cause deviation of close loop street path.
- Solution:
  - assuming small roll and pitch angle, small z translation: manually eliminate possible cause of error out of the algorithm. In this case, no significant height change is detected, so we assume we are only moving on a flat plane.

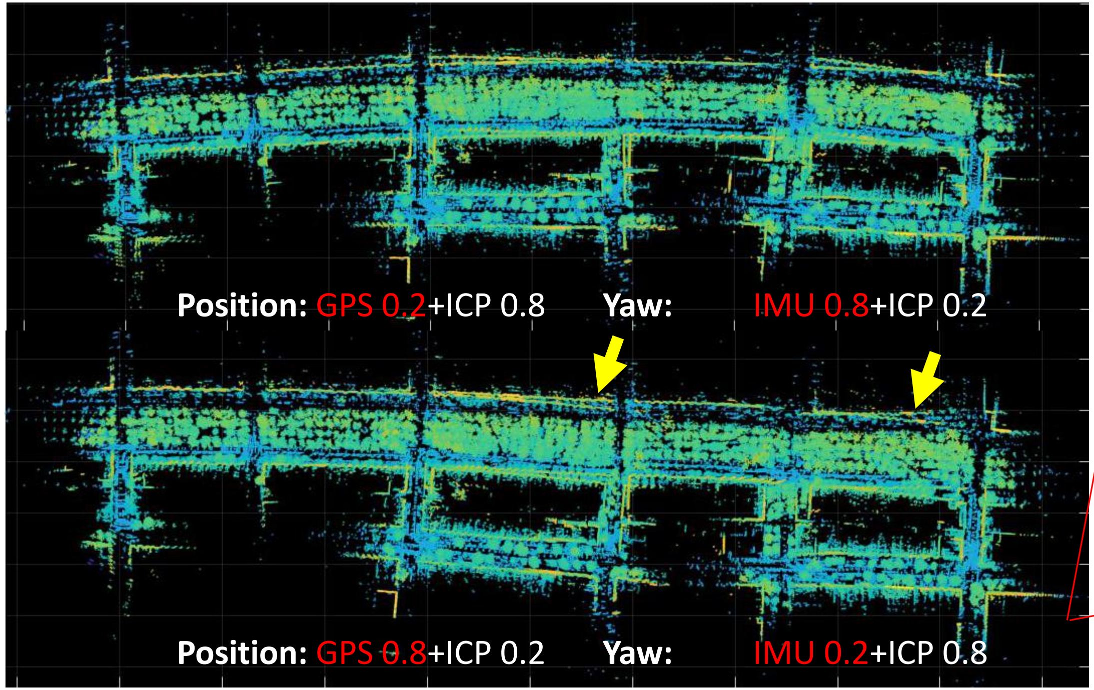
Error Compensation

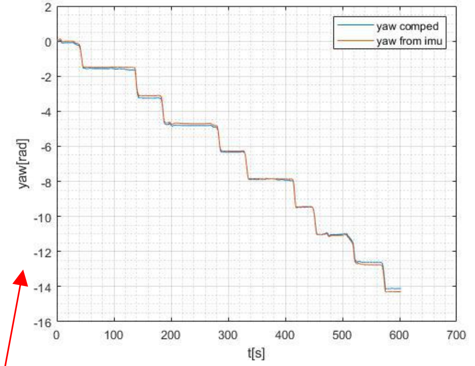


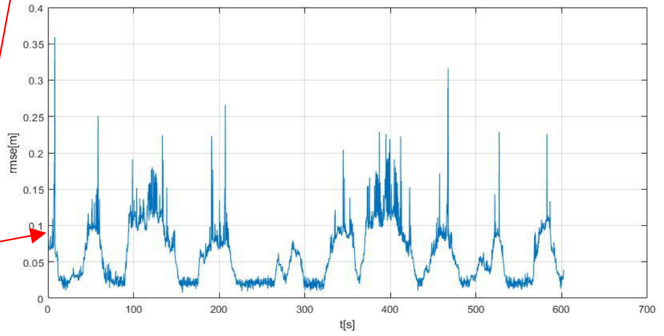
Sensor Fusion: 1st order complementary filter


Preprocess
Yaw from IMU
Find var<0.001 and do polyfit to get the gyro drifting rate
Substract gyro drifting from IMU
100Hz
roll =0,pitch=0,yaw=α*yaw+(1- α)*yaw_from_imu
Transformation (Affine3d)
Rotation
Rotm2euler
Translation
Rotm2euler
Transformation (Affine3d)
If GPS: [α*[x;y]+(1- α)*pos_gps;z]; else: [x;y;z=0]
Position GPS
Rotate initial angle

1.2.2 Results of ICP+IMU+GPS:







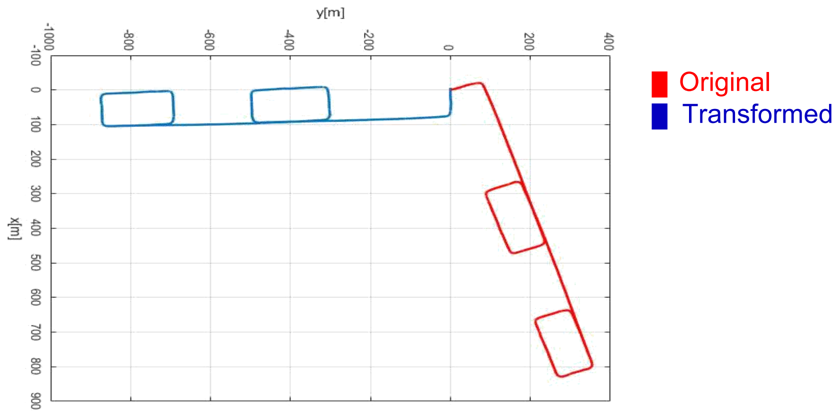
As you can see the final map using ICP +IMU+GPS with complementary parameters as 0.8 and 0.2 gives significantly better results. The RMSE is also much better (only 0.08m) comparing with ICP only algorithm (0.6m). But there's still some mismatching in the final map because we don't have a true value of orientation(like magnetometer), fusion of ICP and IMU still has some accumulated error (yellow arrow).

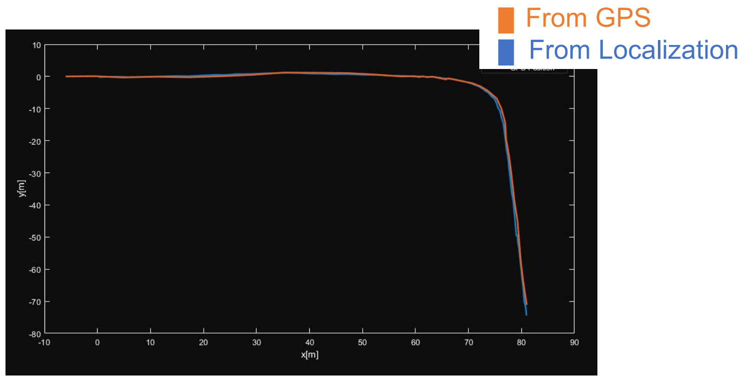
1.3 Exploring methods to improve the mapping results

Problem 1. GPS Coordinate Mismatch:

GPS is projected using UTM coordinate (real world coordinate),$\mathbf{U}_{\mathbf{w}}$, yet trajectory from IMU and Lidar odometry is in local coordinate,$\mathbf{U}_{\mathbf{imu}}$, (origin is at initial start point and x-axis direction is the initial vehicle heading). Thus we don't have a transformation between $\mathbf{U}_{\mathbf{w}}$ and $\mathbf{U}_{\mathbf{imu}}$, so if we directly fuse these two, the map will be distorted.

To solve this issue, we select the first 200s data and generate the map seperately, then we do polyfit of the trajectory then we can get a rough translation and rotation offsets on both axis. Then we can either adjust the initial yaw estimation of the vehicle or we can tranform the gps trajectory from gps frame to local frame. As you obser





Problem 2. Lidar Distortion

- This is the major reason of accumulation of error in ICP. One Lidar scan takes some time, during that time the vehicle is moving, so it will introduce some distortion.
- From Fig.a below we know the lidar scan method is like a vertical line sweeping around 355-365deg.
- Thus there exists a time delay between line to line, causing the distortion. I assume the scan is performed at constant angular velocity, and we only compensate for translation error.
- Yet the velocity estimation maybe inaccurate and no rotation correction involve so this method still leaves some distortion.

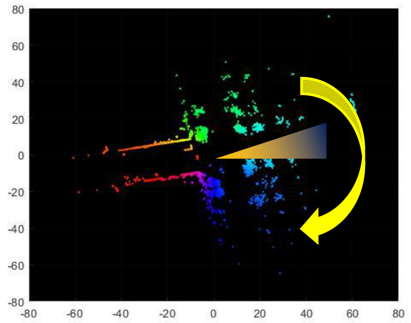


Fig.a Lidar scan sequence

Raw Point Cloud

initial angle =atan(y0,x0)

For every point:

1. Calculate the angle between current point and initial point
2. Linearly interpolate the position drift

Repackage the point cloud

undistorted point cloud

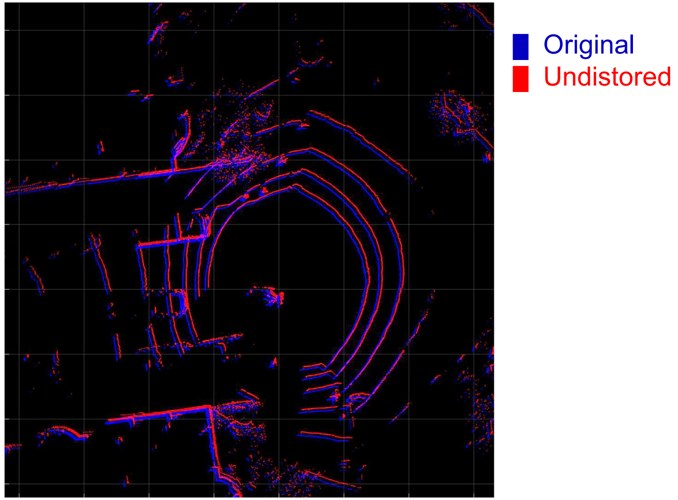
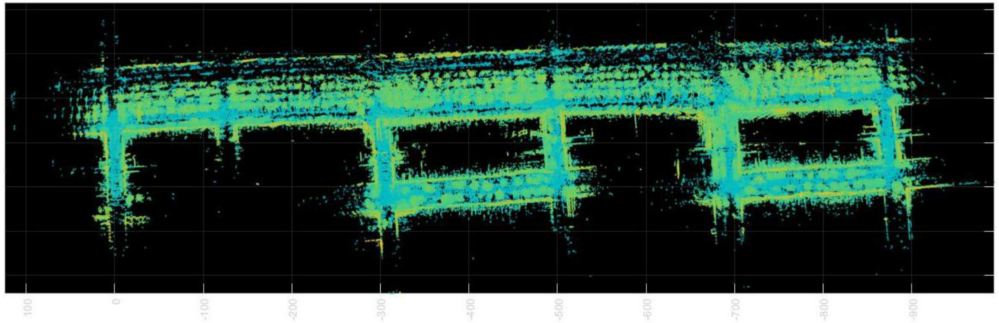
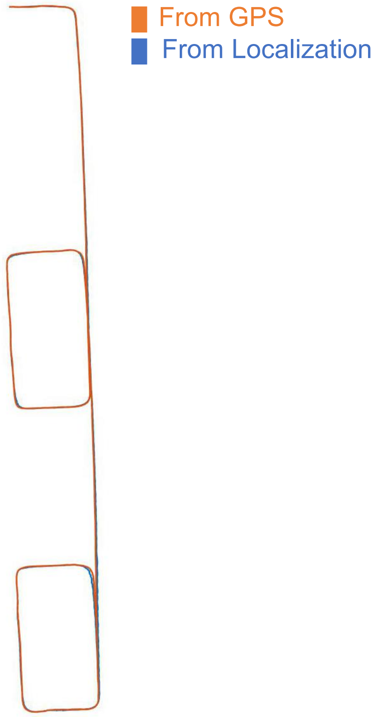


Fig.b Undistorted Lidar Scan

Final Result





- The parameter used for this mapping results is 0.8 and 0.4, high yaw compensation ratio is to avoid yaw deviation from lidar distortion, low position compensation ratio is to avoid low frequency and high noise gps influence on the trajectory.
- And as you can see from the figure to the right the trajectory we got from localization is very close to the GPS trajectory.
- Lidar distortion correction and GPS initial position adjustment really helped to improve the results.

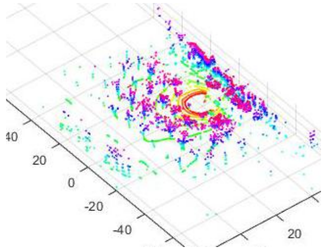
1.4* Accelerate the algorithm using PCL in C++

The algorithm is implemented in MATLAB, and all points in the pointcloud is used when doing ICP, which will cost a lot of computation time even though the cloud points are downsampled. This will slow down the whole algorithm. To accelerate the algorithm, in order to get to a real-time behavior, I took as reference the feature-extraction method used for pointcloud registration in LOAM-SLAM.

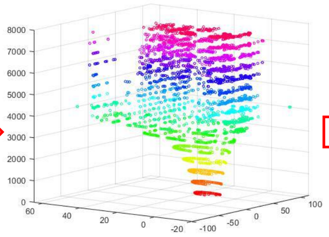
In localization and mapping processing, it's the visualization of the map that spends the most time, if without visualization of the map, each frame takes about 0.5 sec in MATLAB. With visualization, the time will be 4 sec.

So the same algorithm is then implemented in C++ with PCL library again.

The feature extraction procedure is shown below:



$\rightarrow$



$\rightarrow$

calculate curvature of each point on each line

$$dx = \sum_{j=i-5}^{j=i+5}(x_j - x_i)$$

$$dy = \sum_{j=i-5}^{j=i+5}(y_j - y_i)$$

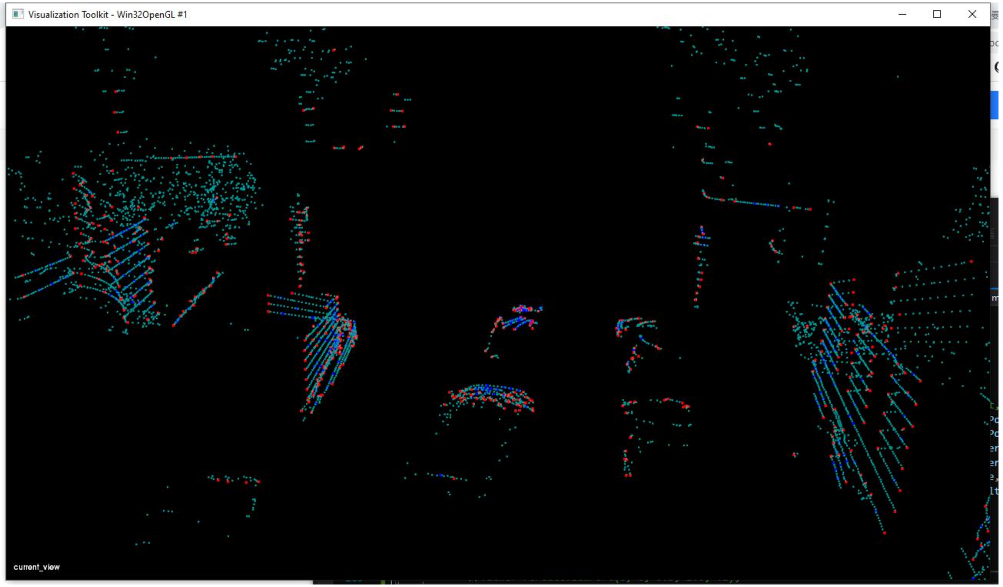
$$dz = \sum_{j=i-5}^{j=i+5}(z_j - z_i)$$

curvature is defined as $\mathrm{curv} = \mathrm{dx}^2 + \mathrm{dy}^2 + \mathrm{dz}^2$

$\rightarrow$

corner points

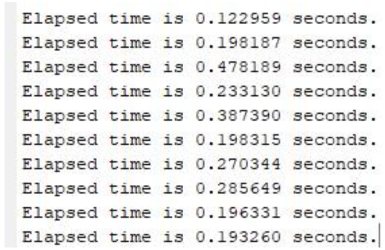
flat surf points



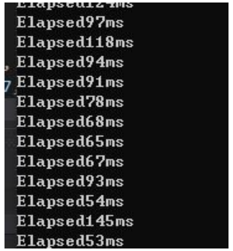
red points: edges
blue points: faces

So after this feature extraction the points density is significantly lower, but points with important messages(corner, surface) are kept. Then we can use these points for ICP registration, we should get similar results but with much faster response.

As you can see from the screen capture, after implementing feature extraction, the average time used for icp registration is around 60-70ms which is significantly less than 200ms on MATLAB.



```
Elapsed time is 0.122959 seconds.
Elapsed time is 0.198187 seconds.
Elapsed time is 0.478189 seconds.
Elapsed time is 0.233130 seconds.
Elapsed time is 0.387390 seconds.
Elapsed time is 0.198315 seconds.
Elapsed time is 0.270344 seconds.
Elapsed time is 0.285649 seconds.
Elapsed time is 0.196331 seconds.
Elapsed time is 0.193260 seconds.
```



```
Elapsed124ms
Elapsed97ms
Elapsed118ms
Elapsed94ms
Elapsed91ms
Elapsed78ms
Elapsed68ms
Elapsed65ms
Elapsed67ms
Elapsed93ms
Elapsed54ms
Elapsed145ms
Elapsed53ms
```

Part2. How to get a cleaner dense Map?

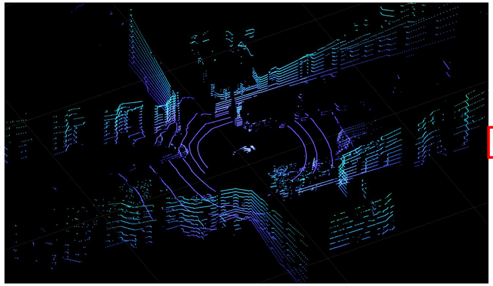
Why do I come up with this idea?

Normally, we would like a dense map, which can be easily converted into 3d meshes later. Yet due to many issues such as moving objects, matching errors, we have to lower the map update rate to get an accurate map with less noise. In my case the algorithm I'm using comes up with much larger error comparing to those uses complicated filters, thus I need a lower mapping update rate. In this case it's set to every 5 frames with 2 step interval (which means it updates every 1 sec) , the downsampling grid size is 0.4m to speed up the visualization, and the sensor is a 16-line velodyne sensor. So all these factors cause the result to be less satisfying, the map is not as dense as I want.

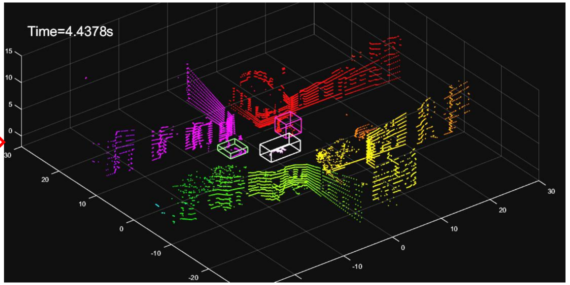
So can I get a dense map with low noise? The answer is yes but we need to seperate those regions of interest out of the original point cloud and then register that into the final map.

From previous results we can see that the map is very noisy, there're a lot of objects other than buildings and roads on the map, such as passing-by vehicles and pedestrians.

To get rid of these objects, we need to first cluster the point clouds, then try to seperate these objects.



Ground Extraction



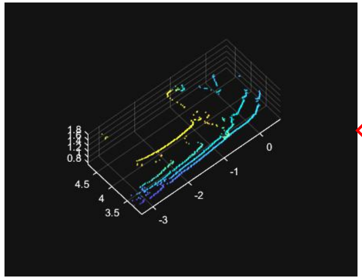
Euclidean Clustering

Use Principal Component Analysis(PCA) to get two main direction from clustered points

Dimension conditions based on a normal car: length around 5, width around 2m, height around 2m

Marked out with bounding box

Meet Dimension Condition: Car

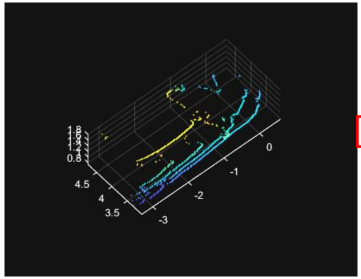


buildings

Cluster Size dimension condition: point number>2000 and dimension is significantly larger

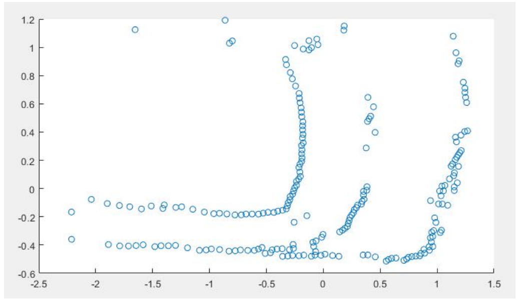
Mapping

Recognizing a car using PCA method:

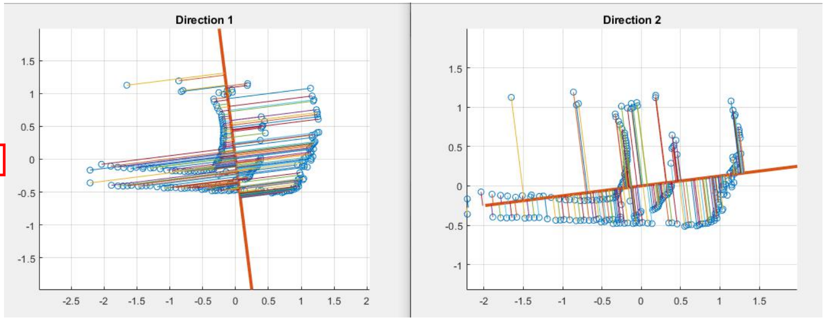


birds eye view

substract mean value

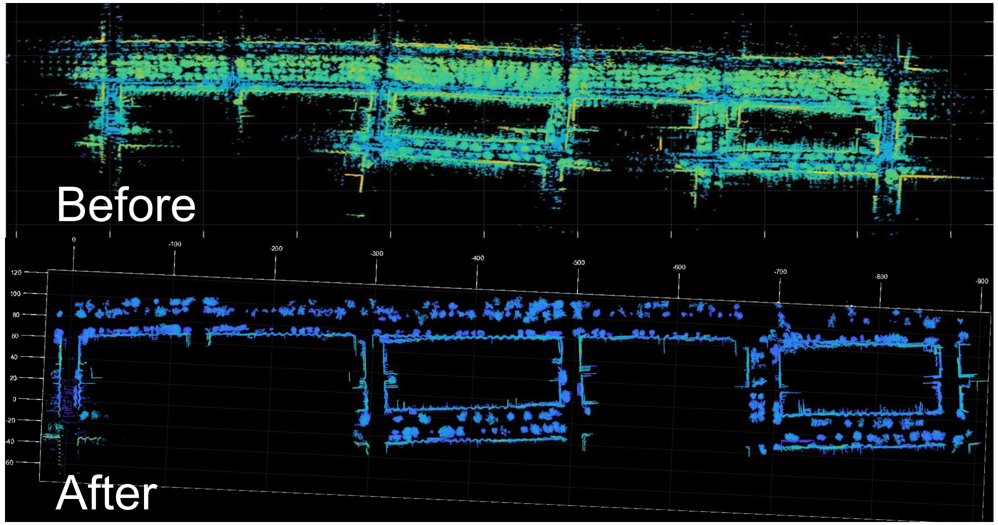


X=Position*Position';
[U,V]=eig(X);
u,v are the two main direction we are looking for, they represents the pose of the object of interest.



check if the bounding box with respect to these two major directions is within the range limits, and if it meets other conditions (such as density, because trees around the same size normally have lower dfensity) if so it is recognized as a car. Same method can be used on other things.

- As you can see from the figure below, the map is much cleaner, all the edges from buildings are preserved, while the ground, vehicles and pedestrians are removed.
- Some trees are recognized as buildings and are left on the map, but comparing with the original one, the quality is significantly better.
- Also, this method is faster using C++ version than MATLAB version.



Conclusion and Future targets

- In this final project I explored doing Lidar SLAM in MATLAB and C++ and got a reasonably satisfying result with sensor fusion with a complementory filter.
- I explored map reconstruction by extracting buildings out of it.
- I also explored doing all these things in ROS, yet because I am using ubuntu inside VMware, so rviz visualization is not working properly, but using ROS can allow me for much more efficient source code development. So in the future I will try to keep learning and implenting robotics algorithm on ROS.

Finally

- Git-Lab Link of my project:
  - https://gitlab.com/yang.yich/eece5554_roboticssensing.git

For MATLAB demostration just run **final_proj.m** under /src/MATLAB

For C++ demostration, install **Final Project Yichu Yang.msi** under /src/C++/setup