

STUDENT ID: D11103003 桂學文

DATE: 2024/12/10

In this exercise, I will try to implement a localization model.

The input is GPS and IMU data.

The output is a 3-dim pose(x, y, z)

I will use the Kitti-tracking-GPS IMU dataset.

The code is in:

https://github.com/wennycooper/AIOT_Exercise_3

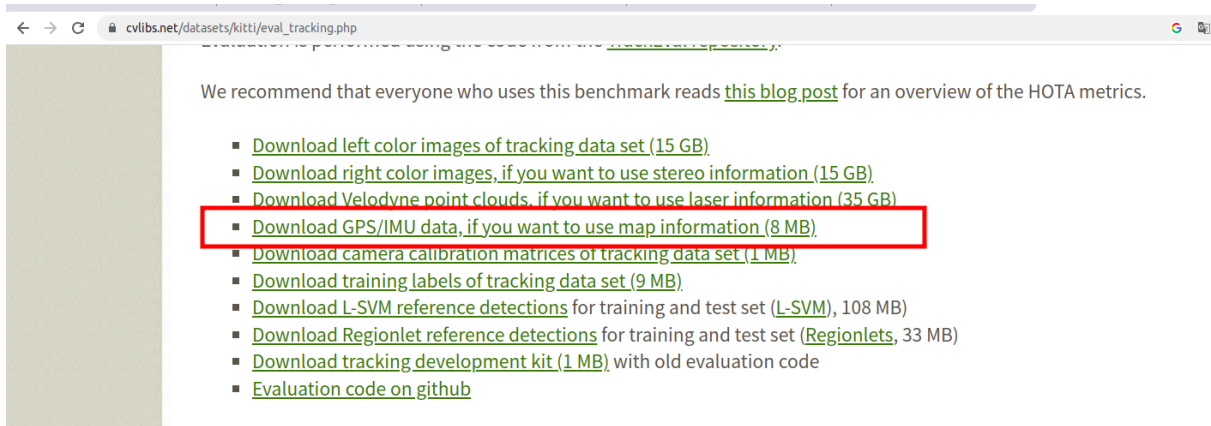
Here are the overall steps.

1. Load the Kitti-Track GPS/IMU dataset (30-dim)
2. Normalise the dataset
3. Construct the model
4. Train the model
5. Inference the result
6. Evaluate
7. Conclusion
8. Other attempts

Here is the details:

1. Load Kitti-Track dataset

In the Kitti-Track dataset, they provide 21 sequences of training set and 29 sequences of testing set. It is downloaded from Kitti official site as below.



Fig, 1 Download KITTI Track dataset

The dataset is in OXTS format which contains 30-dim GPS and IMU information. In the last 7 fields, there seems to be not much information about the localization. So I will use the 23 fields.

GPS/IMU 3D localization unit

=====

The GPS/IMU information is given in a single small text file which is written for each synchronized frame. Each text file contains 30 values which are:

- lat: latitude of the oxts-unit (deg)
- lon: longitude of the oxts-unit (deg)
- alt: altitude of the oxts-unit (m)
- roll: roll angle (rad), 0 = level, positive = left side up (-pi..pi)
- pitch: pitch angle (rad), 0 = level, positive = front down (-pi/2..pi/2)
- yaw: heading (rad), 0 = east, positive = counter clockwise (-pi..pi)
- vn: velocity towards north (m/s)
- ve: velocity towards east (m/s)
- vf: forward velocity, i.e. parallel to earth-surface (m/s)
- vl: leftward velocity, i.e. parallel to earth-surface (m/s)
- vu: upward velocity, i.e. perpendicular to earth-surface (m/s)
- ax: acceleration in x, i.e. in direction of vehicle front (m/s^2)
- ay: acceleration in y, i.e. in direction of vehicle left (m/s^2)
- az: acceleration in z, i.e. in direction of vehicle top (m/s^2)
- af: forward acceleration (m/s^2)
- al: leftward acceleration (m/s^2)
- au: upward acceleration (m/s^2)
- wx: angular rate around x (rad/s)
- wy: angular rate around y (rad/s)
- wz: angular rate around z (rad/s)

- wf: angular rate around forward axis (rad/s)
- wl: angular rate around leftward axis (rad/s)
- wu: angular rate around upward axis (rad/s)
- posacc: velocity accuracy (north/east in m)
- velacc: velocity accuracy (north/east in m/s)
- navstat: navigation status
- numsats: number of satellites tracked by primary GPS receiver
- posmode: position mode of primary GPS receiver
- velmode: velocity mode of primary GPS receiver
- orimode: orientation mode of primary GPS receiver

Fig, 2 OXTS format

They didn't provide the ground_truth pose. So I use lon,lat,plt to UTM conversion to obtain the ground_truth pose.

[https://en.wikipedia.org/wiki/Universal_Transverse_Mercator_coordinate_system#From_latitude,_longitude_\(%CF%86,%CE%BB\)_to_UTM_coordinates_\(E,_N\)](https://en.wikipedia.org/wiki/Universal_Transverse_Mercator_coordinate_system#From_latitude,_longitude_(%CF%86,%CE%BB)_to_UTM_coordinates_(E,_N))

It is implemented in:

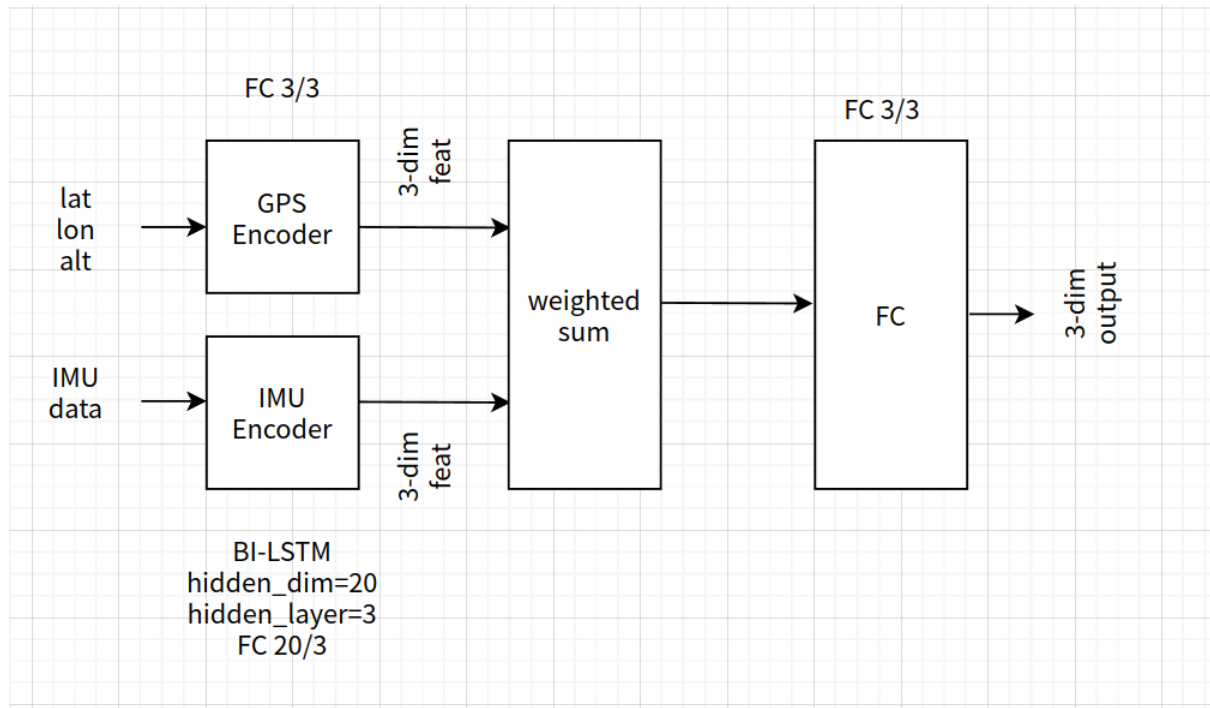
- **load_oxts.py**
compute GT pose and save dataset to pickle file

2. Normalise the dataset

- Normalized_data [lon, lat, alt, r, p, y ...] = (data - mean) / std #(N, 23)
- normalized_pose [r,p,y] = (pose - mean_pose) / std_pose #(N, 3)

3. Construct the model

The model structure is as below. We leveraged the PointNet to solve the number points varying problem.



The main idea is, the pose can be roughly computed in linear to obtain the x,y,z pose. So I will just use a 3x3 Fully-Connected layer GPS encoder to encode the GPS feature.

The IMU encoder is a BI-LSTM + a FC layer. The BI-LSTM is a dynamic mapping module to encode the IMU data. I think it can help to obtain the pose mode precisely. Then I use a weighted sum to sum up the GPS encoder and IMU encoder features. The last layer is a FC 3 by 3 layer to output the 3-dim x,y,z pose.

4. Train the model

The **train.py** is the implementation for section 2, 3 and 4.

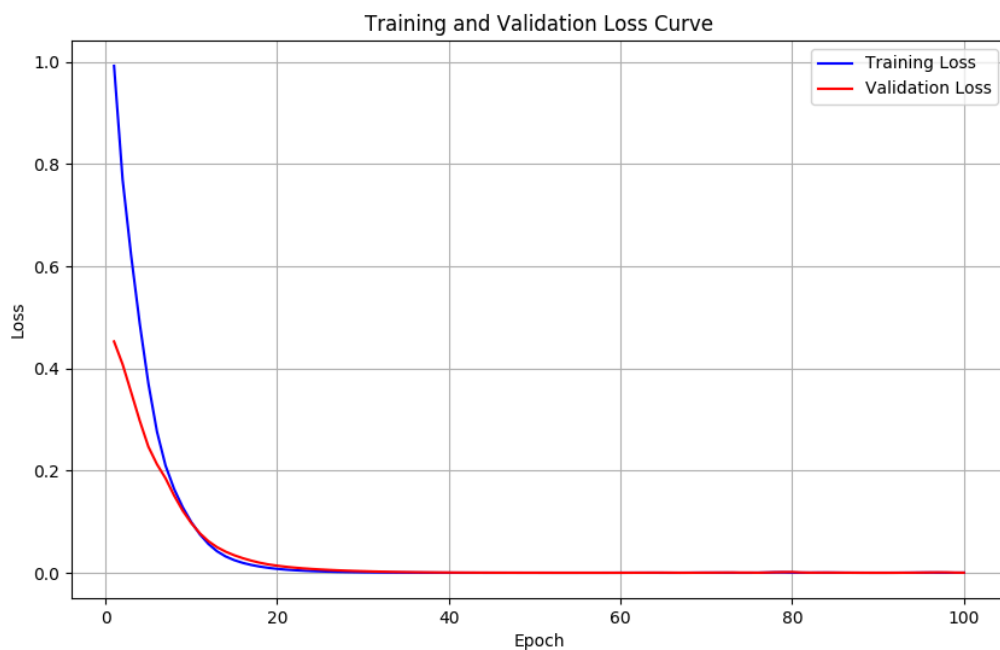
Here are the training environment and hyperparameters:

- GPU: RTX-3090 (24GB)
- Batch_size: 64 (due to VRAM restriction)
- Lr: 1e-3

- Weighted sum: GPS: 1.0, IMU: 0.1
- Loss function: MSE
- Training time: 1.2 hrs
- Epoch: 100

The dataset sequence 00 ~ 20 is for training and the sequence 21 ~ 50 is for validation.

Here is the training curve.



5. Inference the result

The inference code is in **`inference_by_seq.py`**
It outputs 3-dim pose by specific `sequence_index`.

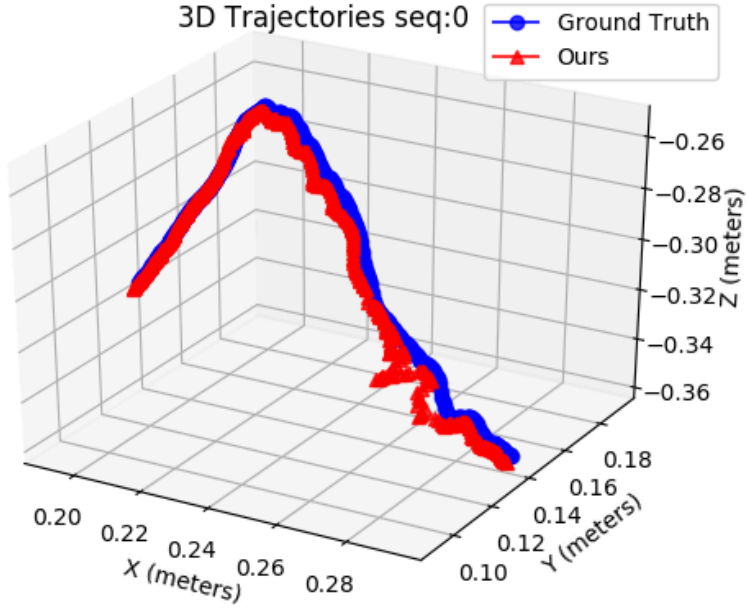
6. Evaluate the result

Then we can evaluate the testing set results.

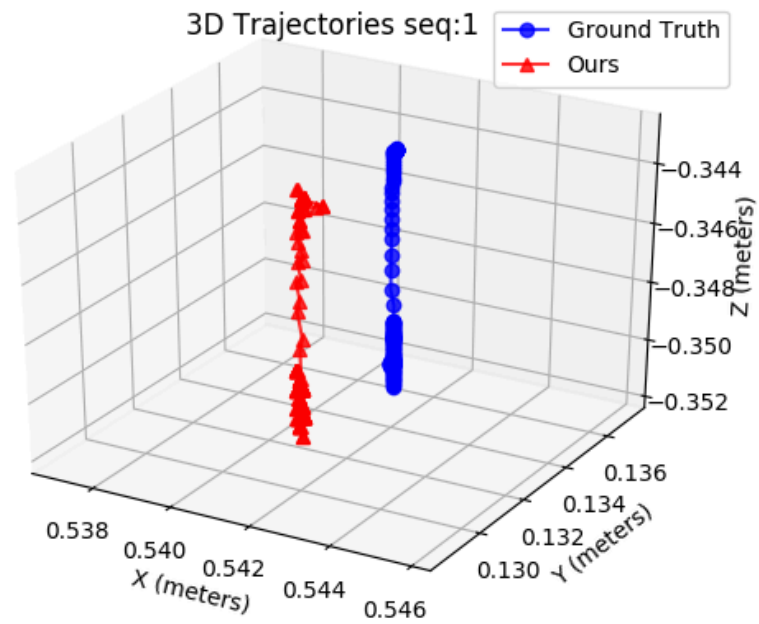
Following is the result for each sequence(00 ~ 28).

The trajectory in **blue** is the ground_truth and the trajectory in **brown** is my result.

The errors information is below the trajectory in each sequence

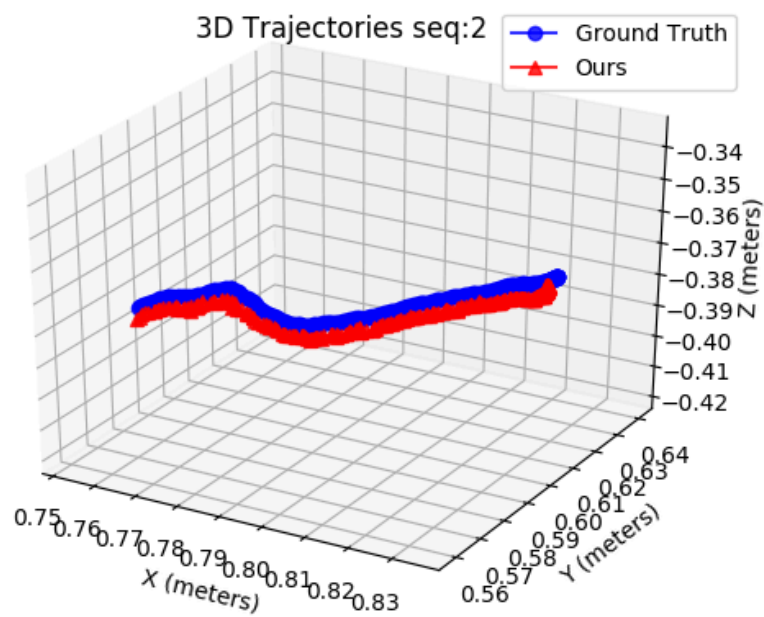
Seq	Trajectory comparison, ATE(m) and Translation Error(%)
00	<div><p>3D Trajectories seq:0</p><p>Absolute Translation Error (m): 0.0034 Translation Error (%): 0.83%</p></div>

01



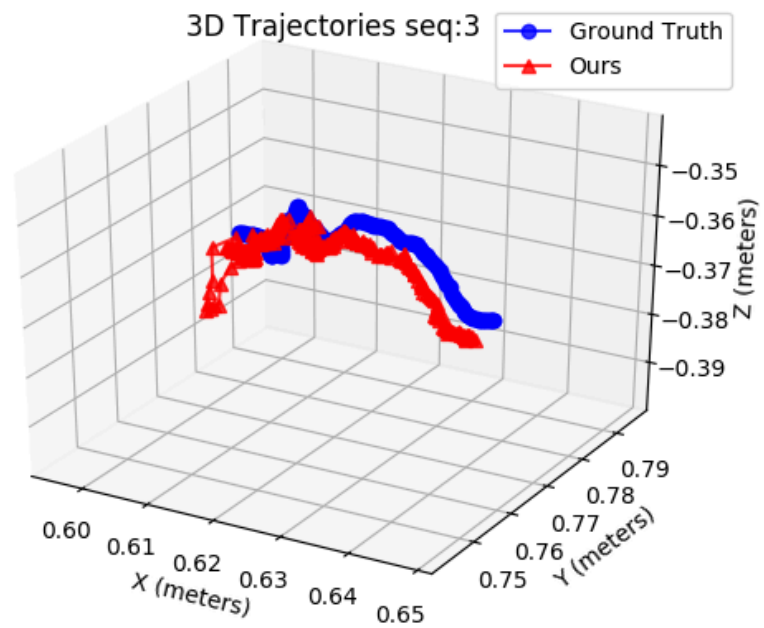
Absolute Translation Error (m): 0.0026
Translation Error (%): 0.39%

02



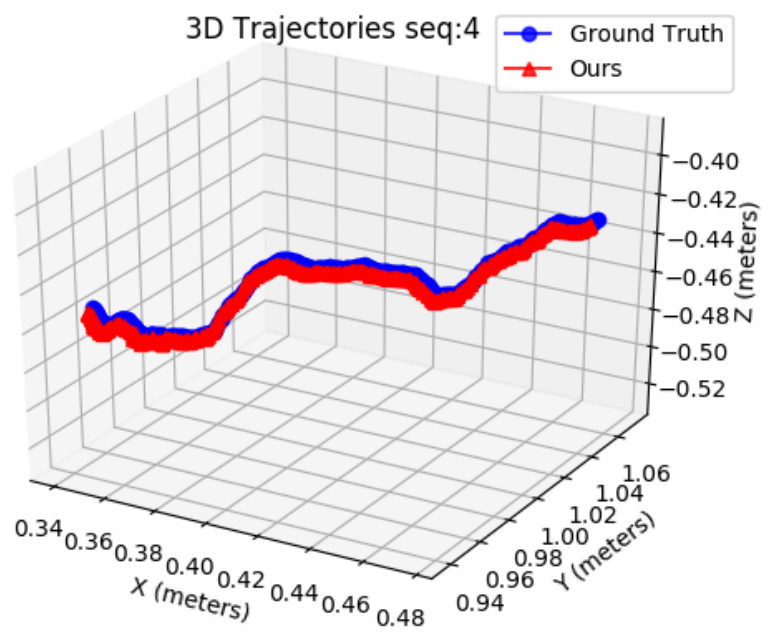
Absolute Translation Error (m): 0.0048
Translation Error (%): 0.45%

03



Absolute Translation Error (m): 0.0049
Translation Error (%): 0.47%

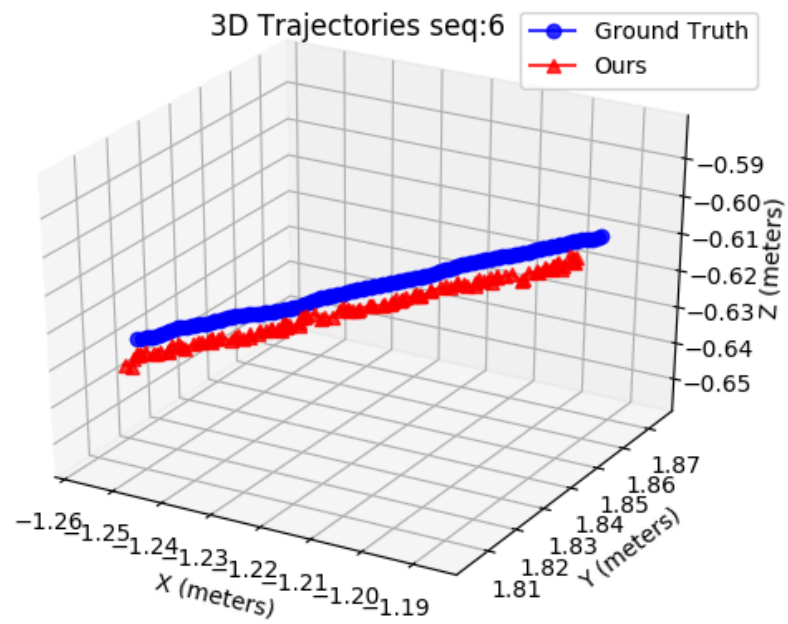
04



Absolute Translation Error (m): 0.0055

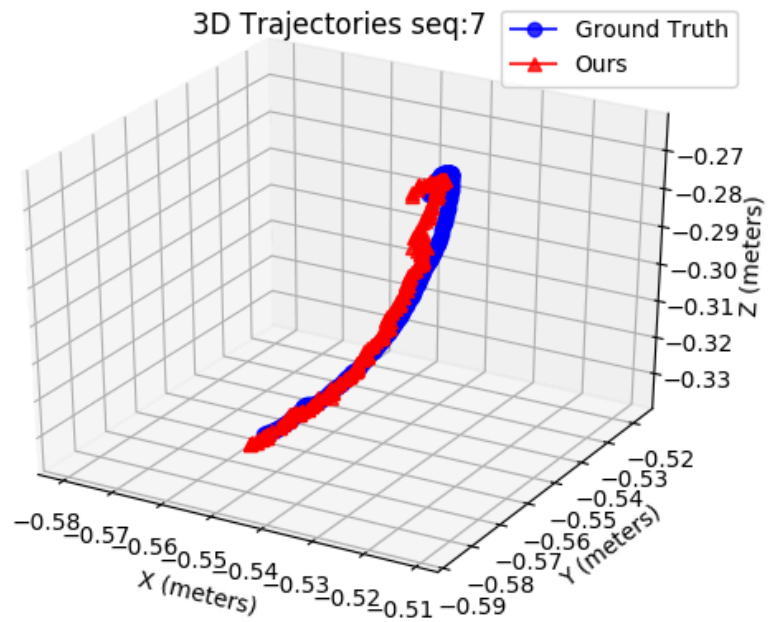
	Translation Error (%): 0.46%
05	<div data-bbox="542 414 1300 1030" data-label="Figure"> <p>3D Trajectories seq:5</p> <p>The figure is a 3D plot titled '3D Trajectories seq:5'. It displays two trajectories: 'Ground Truth' (blue line with circular markers) and 'Ours' (red line with triangular markers). The plot is set in a 3D coordinate system with axes labeled 'X (meters)', 'Y (meters)', and 'Z (meters)'. The X-axis ranges from -0.9 to -0.4, the Y-axis from 1.8 to 2.3, and the Z-axis from -0.3 to -0.8. Both trajectories follow a similar path, starting at approximately (-0.8, 2.2, -0.6), moving towards the center, and then slightly upwards and to the right. The 'Ours' trajectory closely follows the 'Ground Truth' trajectory, demonstrating high accuracy.</p> </div> <p>Absolute Translation Error (m): 0.0151 Translation Error (%): 0.67%</p>

06



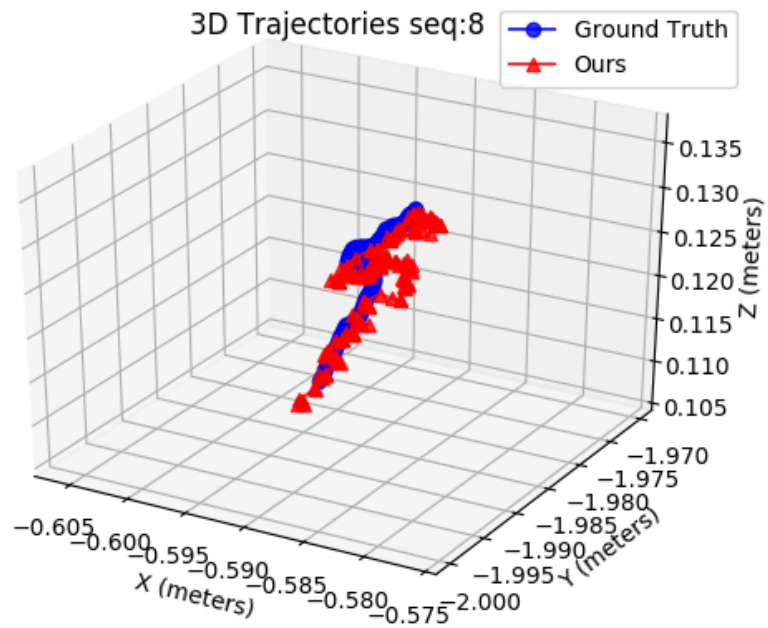
Absolute Translation Error (m): 0.0134
Translation Error (%): 0.59%

07



Absolute Translation Error (m): 0.0039
Translation Error (%): 0.47%

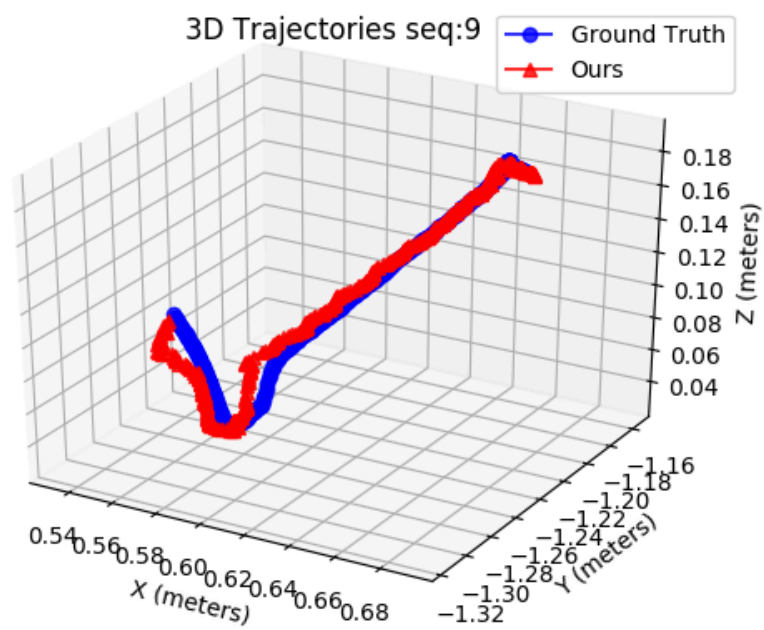
08



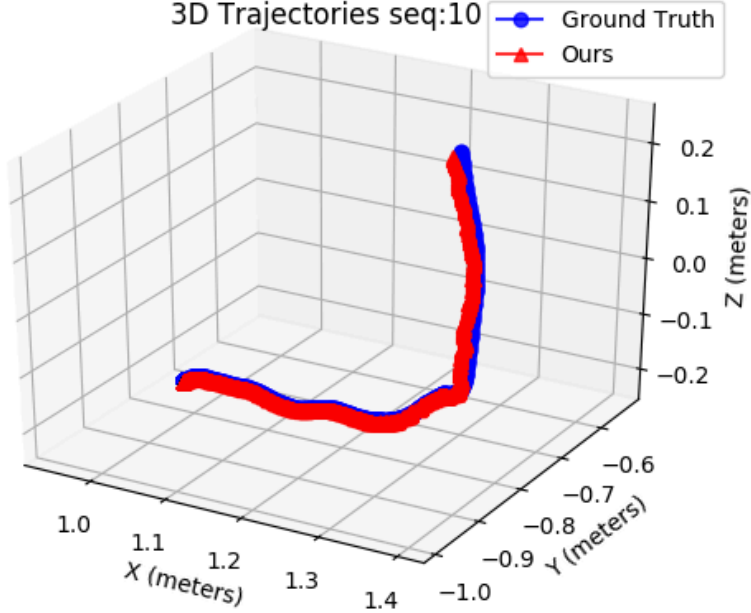
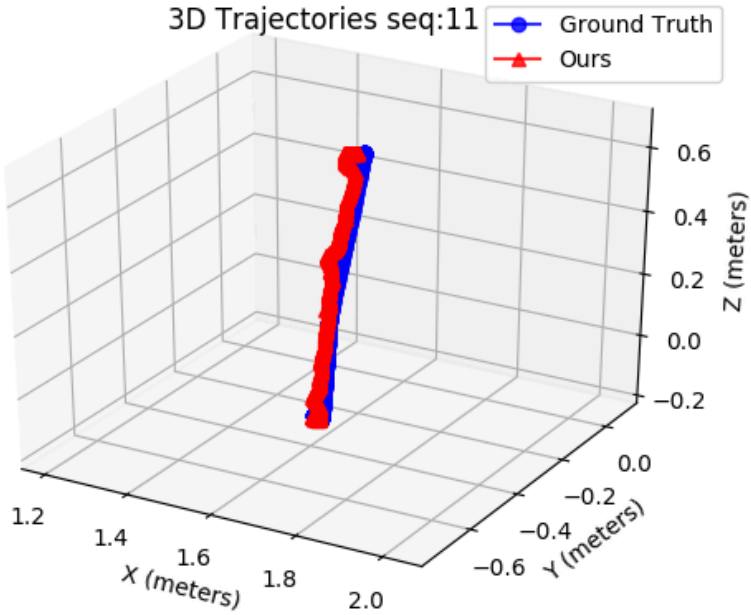
Absolute Translation Error (m): 0.0063

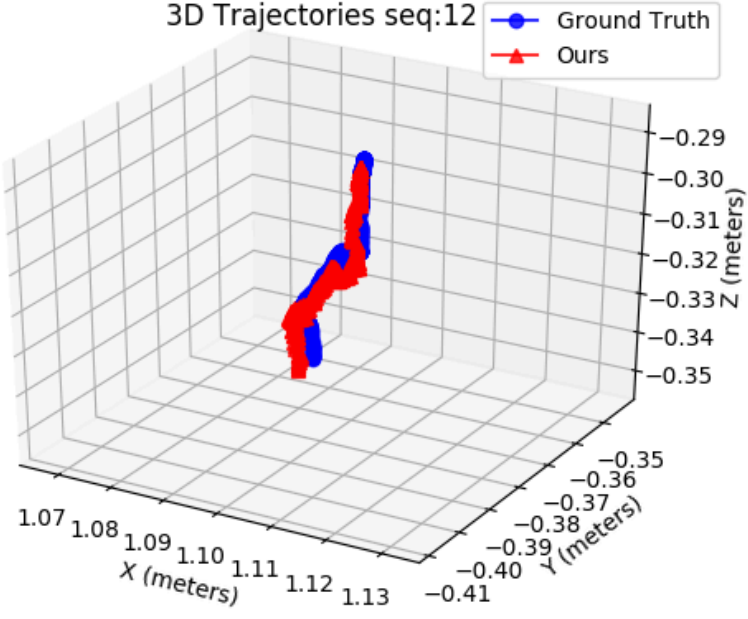
Translation Error (%): 0.30%

09

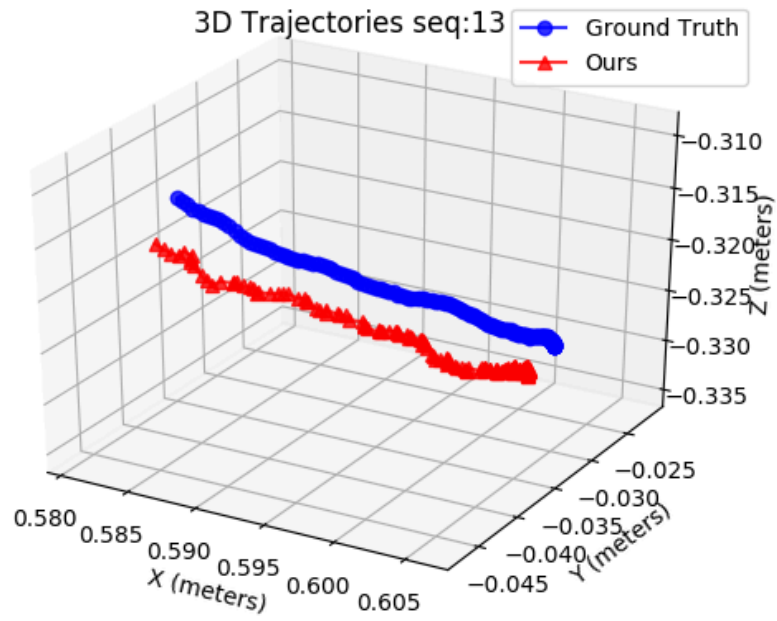


Absolute Translation Error (m): 0.0077

	Translation Error (%): 0.56%
10	<p>3D Trajectories seq:10</p>  <p>Absolute Translation Error (m): 0.0085 Translation Error (%): 0.59%</p>
11	<p>3D Trajectories seq:11</p> 

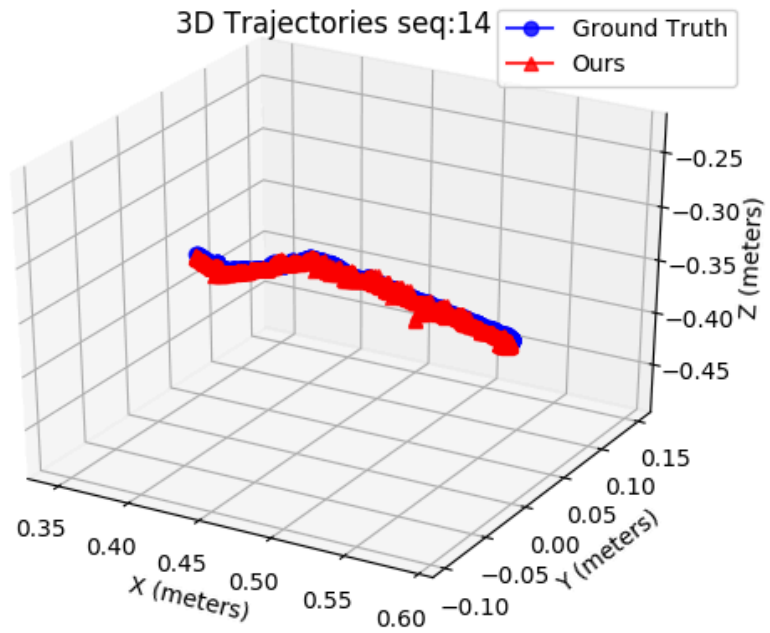
	<p>Absolute Translation Error (m): 0.0196 Translation Error (%): 1.15%</p>
12	<p>3D Trajectories seq:12</p>  <p>Absolute Translation Error (m): 0.0046 Translation Error (%): 0.38%</p>

13



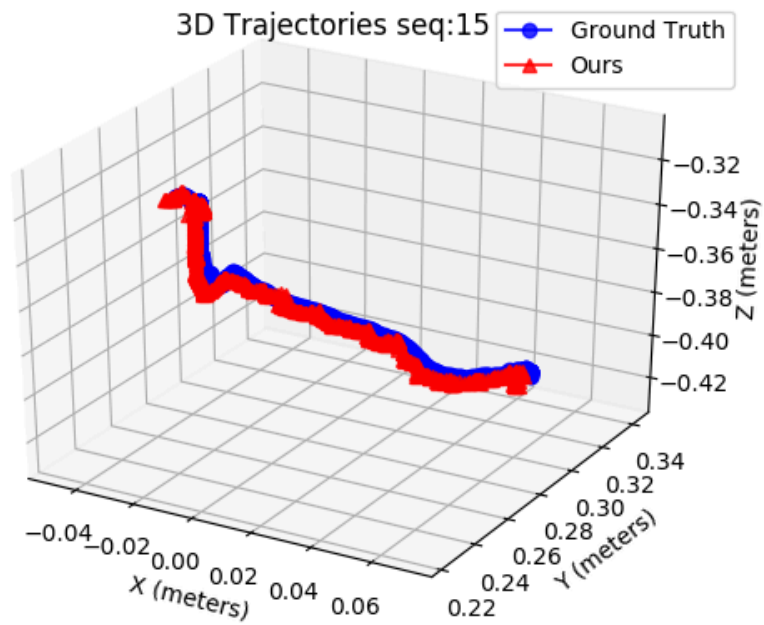
Absolute Translation Error (m): 0.0036
Translation Error (%): 0.53%

14



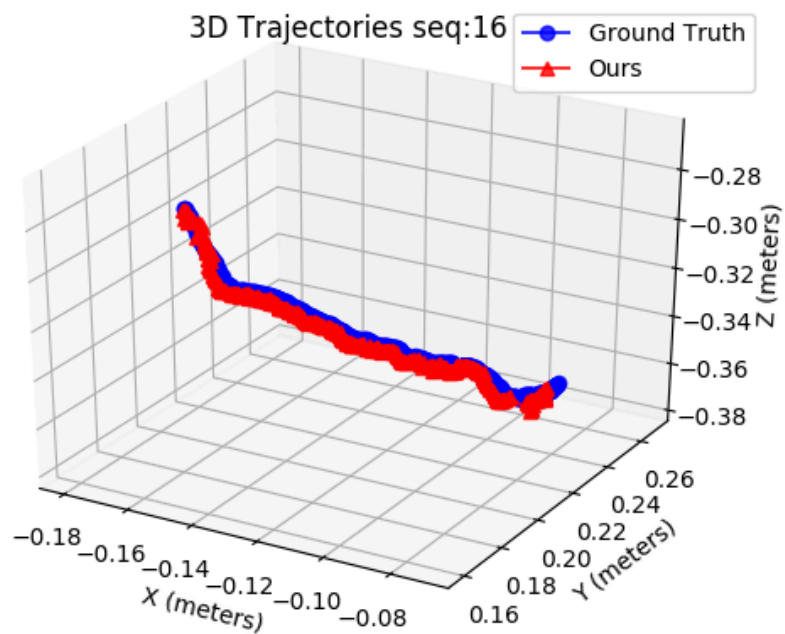
Absolute Translation Error (m): 0.0032
Translation Error (%): 0.55%

15



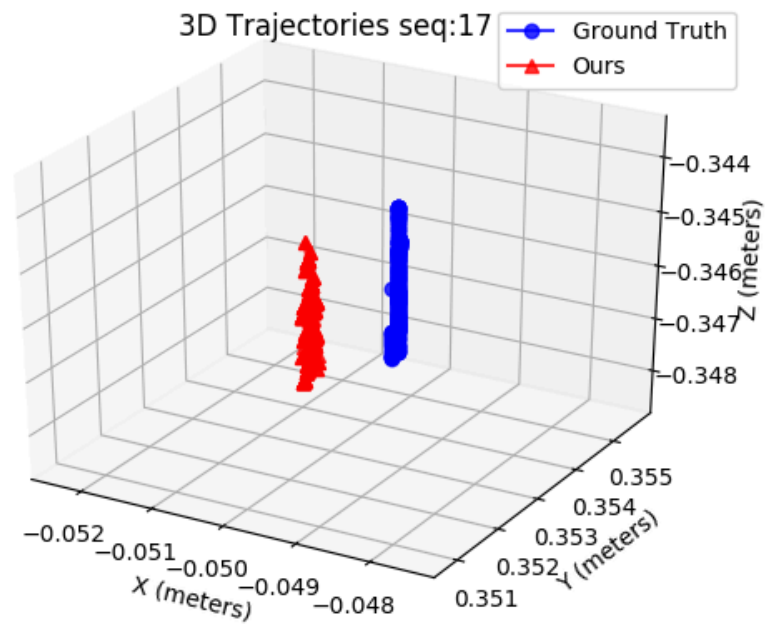
Absolute Translation Error (m): 0.0032
Translation Error (%): 0.67%

16



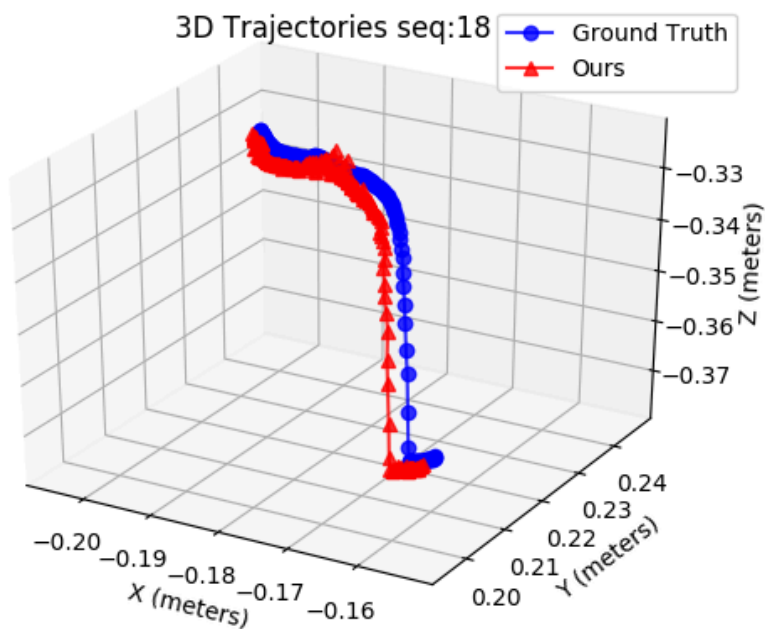
Absolute Translation Error (m): 0.0035
Translation Error (%): 0.85%

17



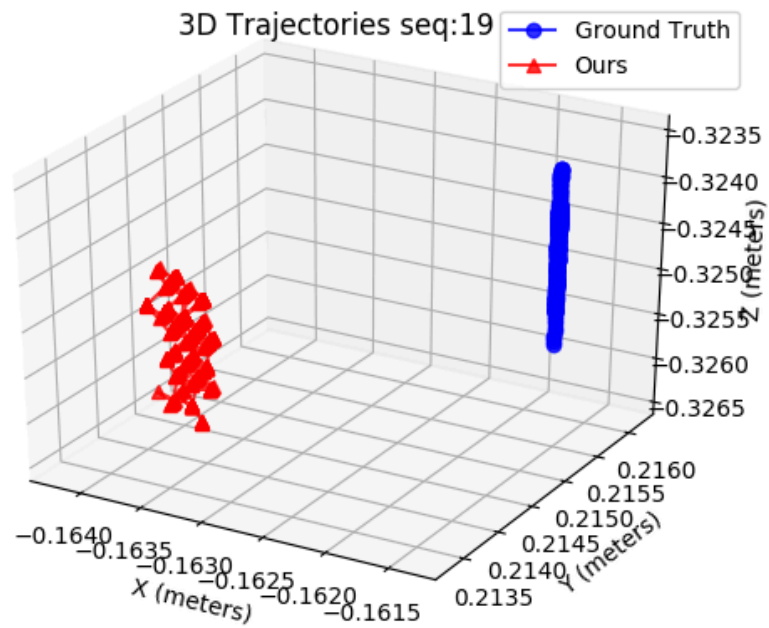
Absolute Translation Error (m): 0.0047
Translation Error (%): 0.95%

18



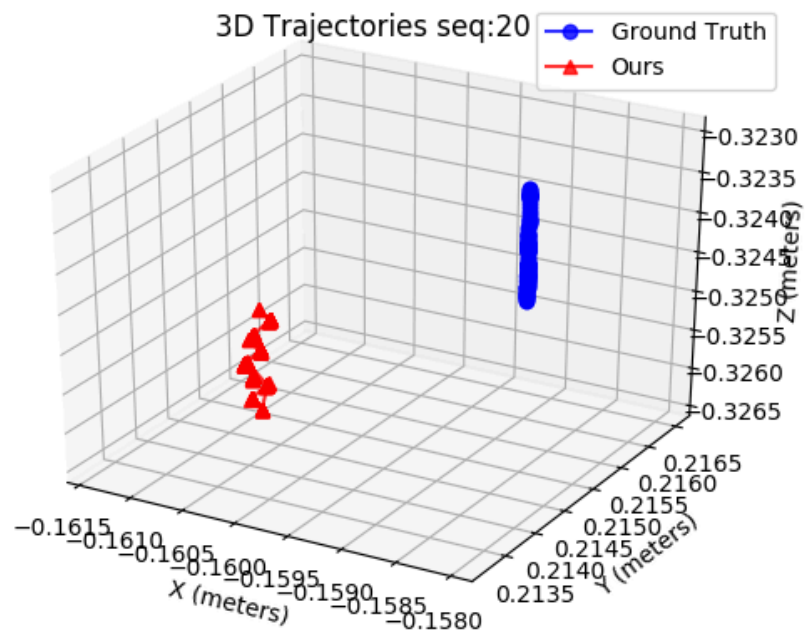
Absolute Translation Error (m): 0.0038
Translation Error (%): 0.88%

19



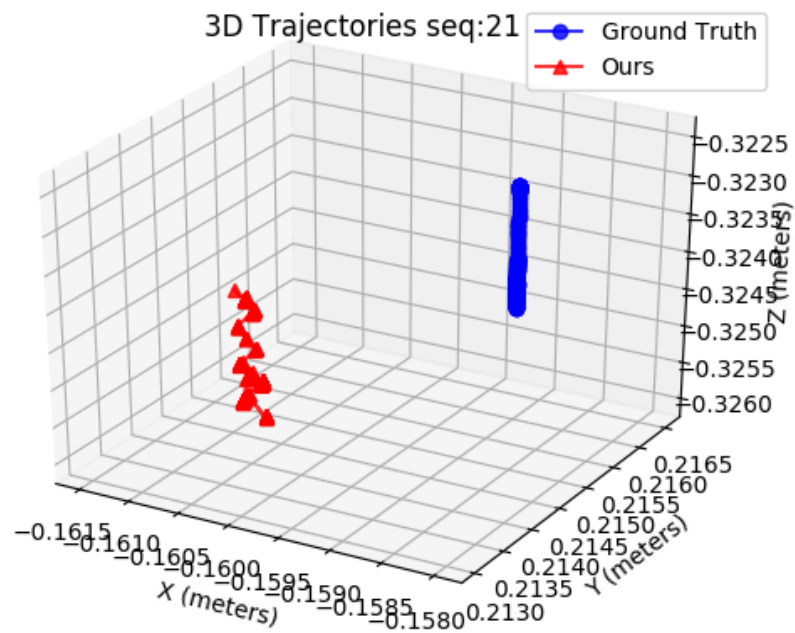
Absolute Translation Error (m): 0.0031
Translation Error (%): 0.74%

20



Absolute Translation Error (m): 0.0035
Translation Error (%): 0.83%

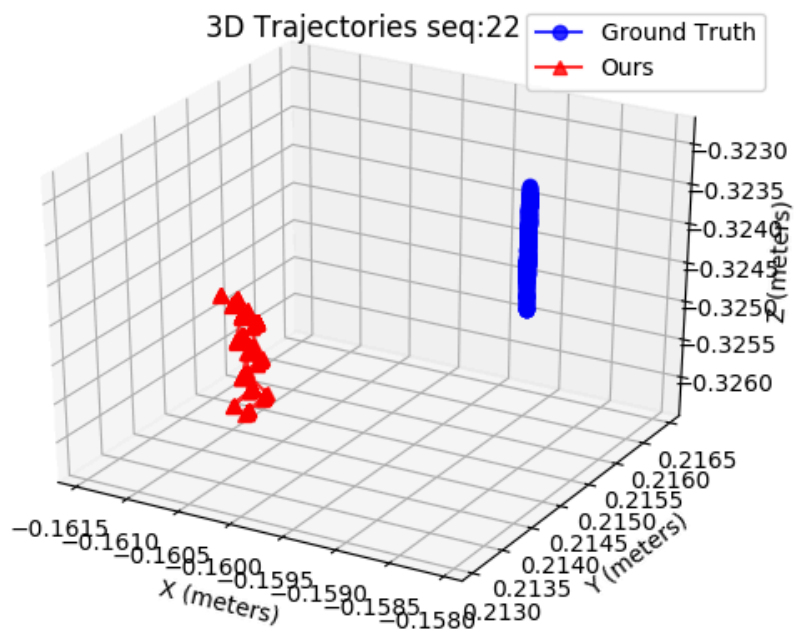
21



Absolute Translation Error (m): 0.0035

Translation Error (%): 0.83%

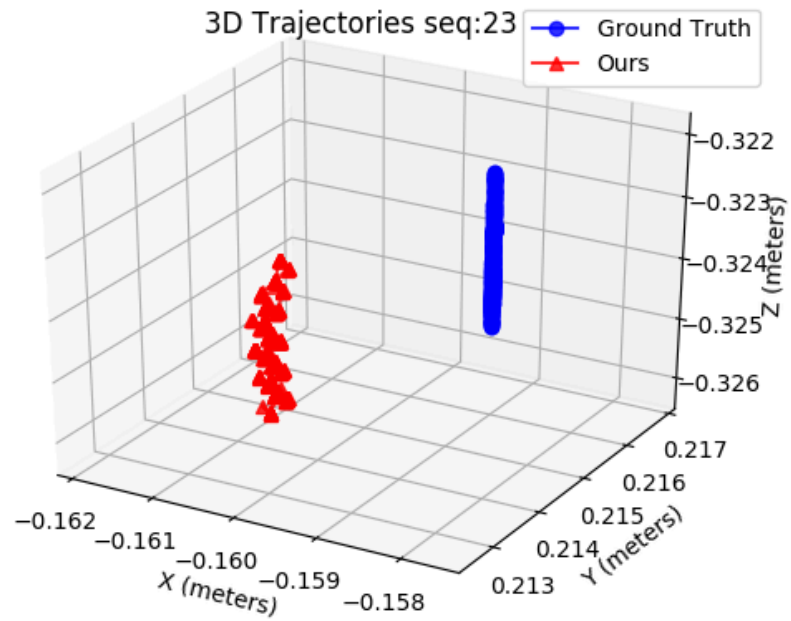
22



Absolute Translation Error (m): 0.0035

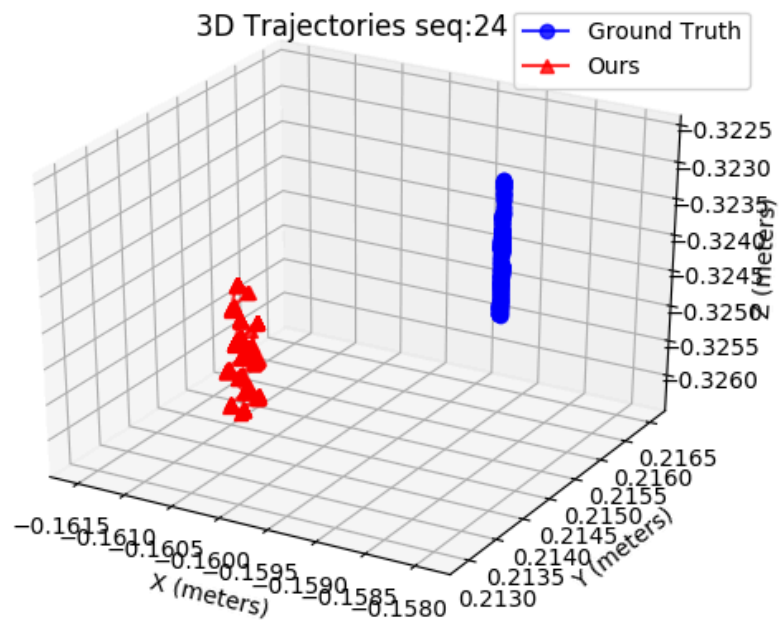
Translation Error (%): 0.83%

23



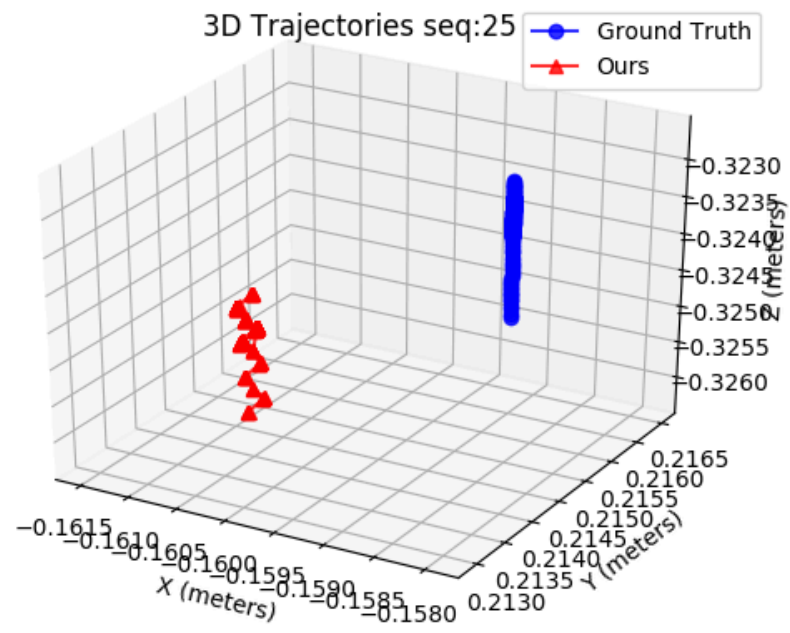
Absolute Translation Error (m): 0.0035
Translation Error (%): 0.82%

24



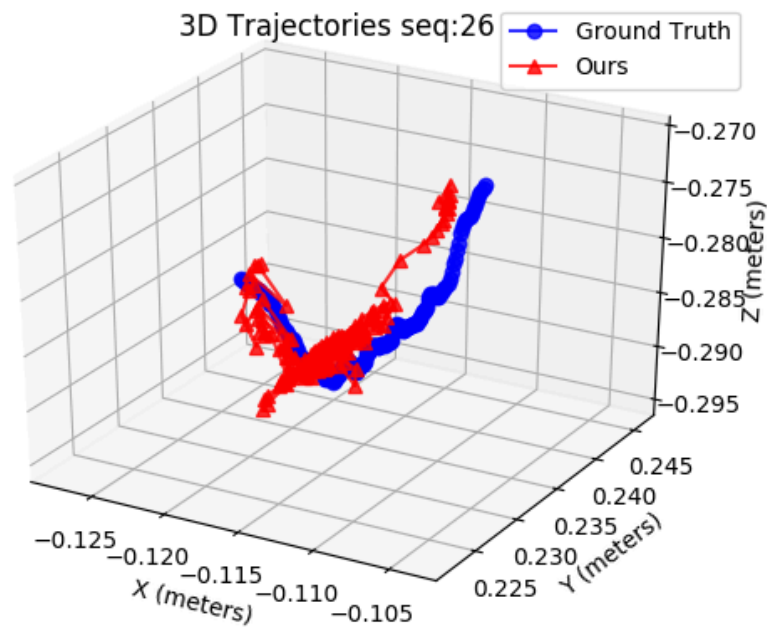
Absolute Translation Error (m): 0.0035
Translation Error (%): 0.83%

25



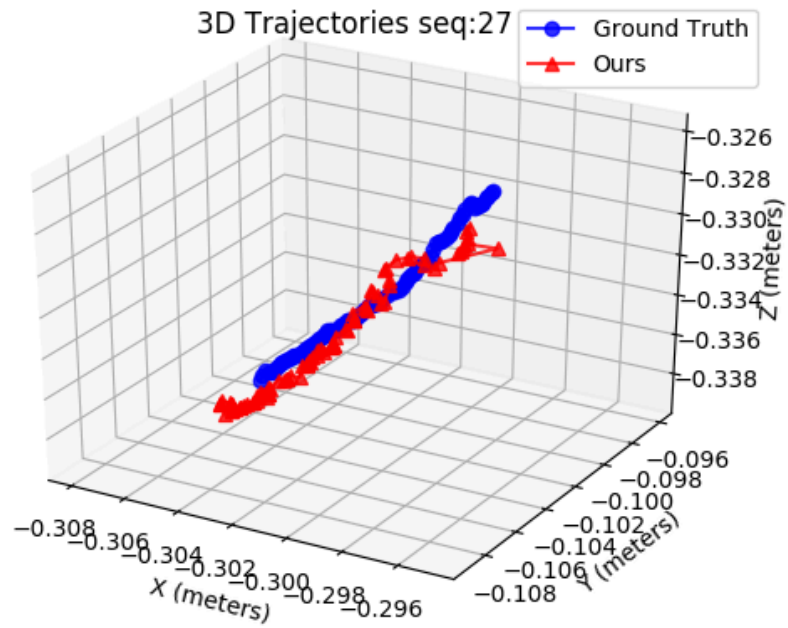
Absolute Translation Error (m): 0.0035
Translation Error (%): 0.83%

26



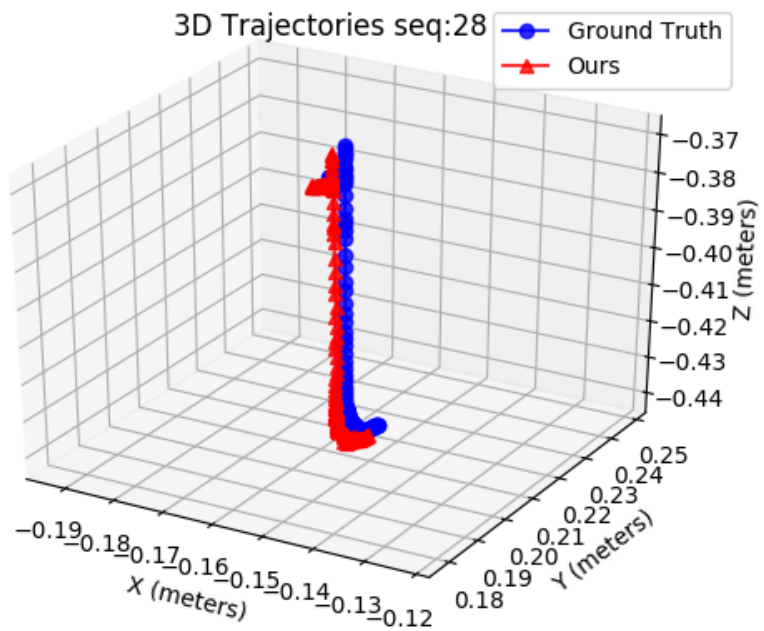
Absolute Translation Error (m): 0.0048
Translation Error (%): 1.22%

27



Absolute Translation Error (m): 0.0028
Translation Error (%): 0.60%

28



Absolute Translation Error (m): 0.0034
Translation Error (%): 0.68%

--	--

8. Conclusions

I think the result is not bad. Maybe it's because this is not a very difficult problem.