Multiview Geometry

Lab 4-Report

Stereo Visual Odometry

Delivered by:

Muhammad Faran Akram (u1999088)
Solomon Chibuzo Nwafor (u1999124)

Supervisor:

Professor Nuno Gracias

Date of Submission:

26/12/2024

1. Introduction:

In this lab, we implemented stereo-visual odometry (SVO) from an input stereo image sequence of the UTIAS Long-Term Localisation and Mapping Dataset provided by the University of Toronto Institute for Aerospace Studies.

2. Implementation:

2.1 Input Stereo dataset: Since we have the data downloaded, we set the *getFileFromFTP* to false, extracted the images, and stored them in a temporary folder.

Figure 1: First stereo image pair of the sequence

2.2. Camera Parameters Settings:

In this section, we set two cameras mounted on a rigid body with a displaced baseline. In our case, the intrinsic parameters of the two cameras are the same, and the extrinsic parameters of the second camera (right camera) are defined by the baseline and no rotation in the coordinate system, as provided in the live script. We used the provided matlab structure—*cameraParameters()* and *stereoParameters()*—to set these camera parameters.

Code:

```
%
http://asrl.utias.utoronto.ca/datasets/2020-vtr-dataset/text_files/transform_c
amera_vehicle.txt
initialPoseRotation = [7.963270829459690e-04, -0.999999682931538,
1.034228258589565e-12; -0.330471987613750, -2.631638783086032e-04,
-0.943815763879472; 0.943815464625260, 7.515860537466446e-04,
-0.330472092396028];
initialPoseTranslation = [0.159, 0.119, 1.528];
initialPose = rigid3d(initialPoseRotation,initialPoseTranslation); % create a
rigid3d object with the initial camera pose

% Create a stereoParameters object to store the stereo camera parameters.
```

```
% The intrinsics for the dataset can be found at the following page:
%
http://asrl.utias.utoronto.ca/datasets/2020-vtr-dataset/text_files/camera_para
meters.txt
focalLength     = [387.777 387.777];     % specified in pixels
principalPoint  = [257.446 197.718];     % specified in pixels [x, y]
baseline        = 0.239965;              % specified in meters
intrinsicMatrix = [focalLength(1), 0, 0; 0, focalLength(2), 0;
principalPoint(1), principalPoint(2), 1]; % Note that matlab uses the K matrix
as the transpose of what the noormal convention...
imageSize       = size(currILeft,[1:2]); % in pixels [mrows, ncols]
cameraParam     = cameraParameters('IntrinsicMatrix', intrinsicMatrix,
'ImageSize', imageSize);
intrinsics      = cameraParam.Intrinsics;
stereoParams    = stereoParameters(cameraParam, cameraParam, eye(3),
[-baseline, 0 0]);
```

2.3. Incoming Frame Processing:

The images are first undistorted to compensate for lens distortions and then rectified to align the image from the left camera with the right camera using these Matlab functions *undistortImage()* and *rectifyStereoImages()*. The result is shown below.

Figure 2: The incoming frame processing

2.4 Feature extraction:

We use the feature extraction to identify points in the image that will match their correspondences. The two plots in Figure 3a and Figure 3b compare the results of standard feature extraction and bucketing. Figure 3a shows that features are concentrated in texture-rich regions, leaving other areas under-represented, which can be problematic for visual odometry. And in Figure 3b, the plot is a more uniform distribution achieved by dividing the image into grids and extracting features from each grid, ensuring better spatial coverage and improving the robustness of the VO system in static scenes.

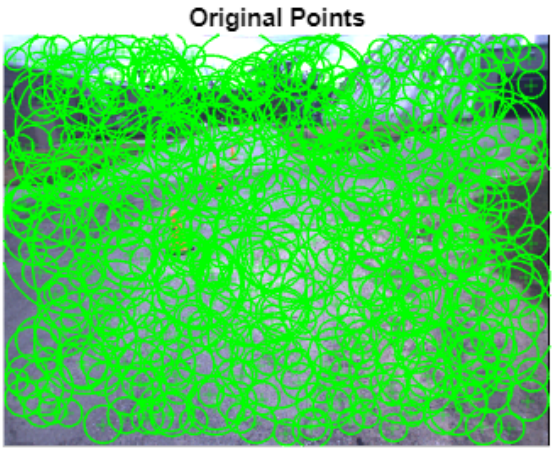

Figure 3a: Original Point Extraction

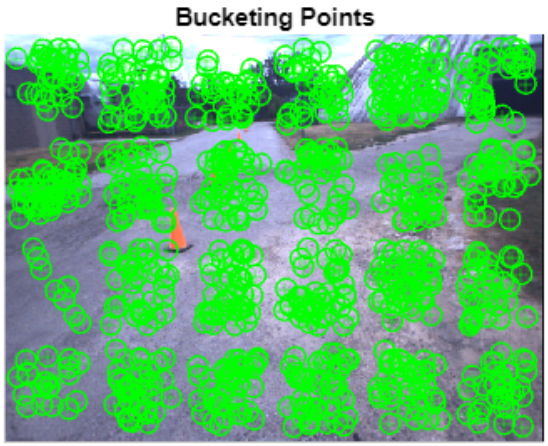

Figure 3b: Bucketing Point extraction

2.5 Circular matching

In the circular matching, we use *CircularMatching()* to establish correspondences between points across four images, as shown in Figure 4. A match is accepted only if the keypoint in the left image at *t–1(P1)* can be consistently tracked through the cycle of images P1 to P5 (left *t–1*→ right *t–1* → right *t* → left *t* → back to left *t–1*).

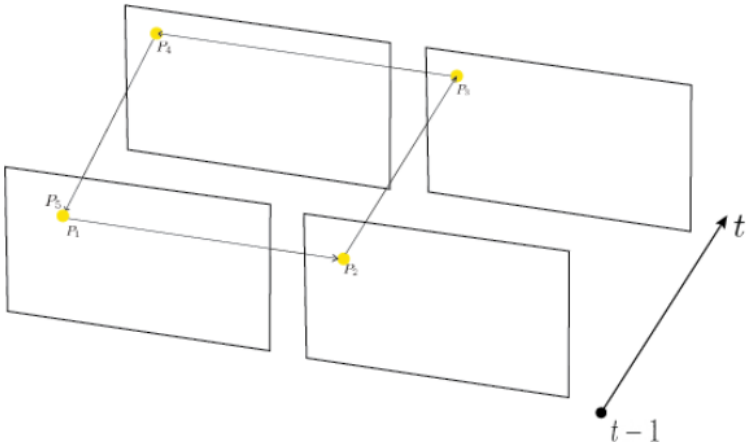

Figure 4: Circular matching of points in two paired images

Figure 5 shows the match points between the leftimage at t-1 and t.

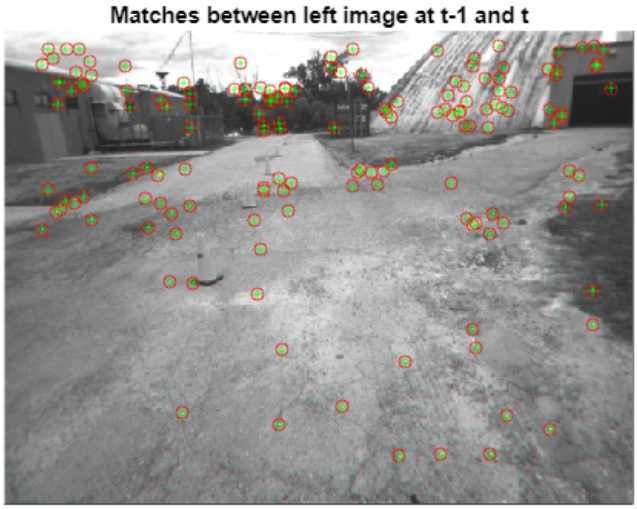

Figure 5: Matches between left image at t-1 and t

Circular Matching Questions:

1. Does the circular matching use geometric constraints?

Answer: Yes, circular matching in multiview geometry is constrained by geometry. The constraints arise from the epipolar geometry, which defines the geometric relationship between multiple views of the same scene.

2. Does it guarantee correct matches?

Answer: It does not guarantee correct matches but it improves the reliability of correspondences. It reduces the likelihood of mismatches by leveraging constraints like circular consistency and epipolar geometry.

If not, then;

a) describe a situation where it would fail.

Answer: If we have symmetry in the views, like a grid-like pattern, then we may find the wrong correspondences due to features extracted from different but similar view.which will lead to a mismatch.

b) provide a detailed explanations on what geometric constraints could be imposed and how.

Answer:

Epipolar Constraint:

Matches between points in two views must satisfy the epipolar geometry constraint:

$$x_2^T F x_1 = 0$$

This restricts potential matches to lie along the epipolar line in the second view, reducing the search space.

Triangulation:

The matched 3D positions reconstructed through triangulation must align with the camera projections.The reprojected points in all views are checked for consistency with the observed 2D points.

Rigidity Constraint:
After initial matches and triangulation, distances between pairs of points are checked for consistency. Matches violating this rigidity assumption are flagged as outliers.

2.6 Two methods to estimate stereo motion

In this section, we compare two methods—3D-to-3D and 2D-to-3D—for estimating stereo motion, which require circular matches. The matched points obtained with the *getInliersIdx()* function in the left and right images at t-1 and t are shown in FIG. These matched features are triangulated to obtain 3D points, and the rigid transformation between the two 3D point clouds is estimated.

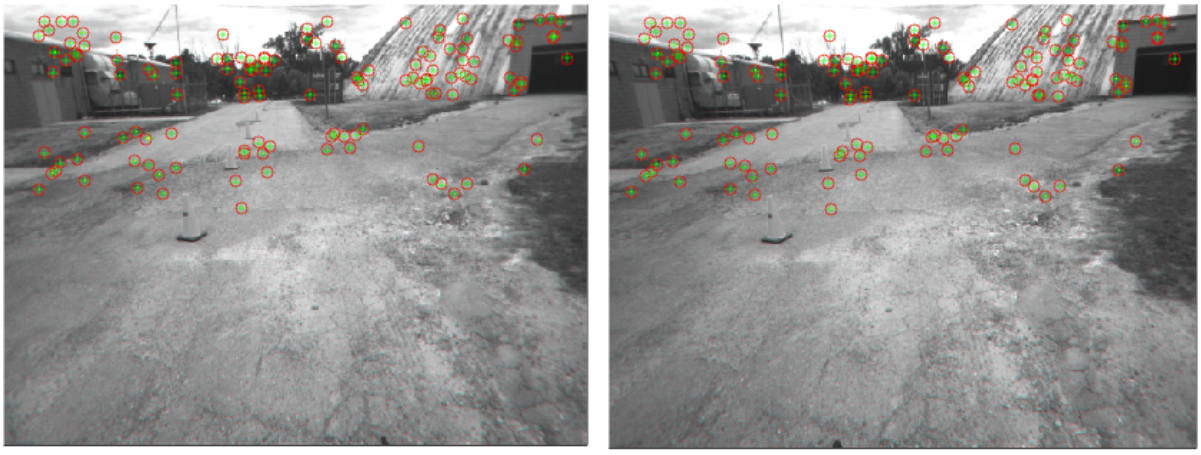

Figure 6: Matched points of the left and right images

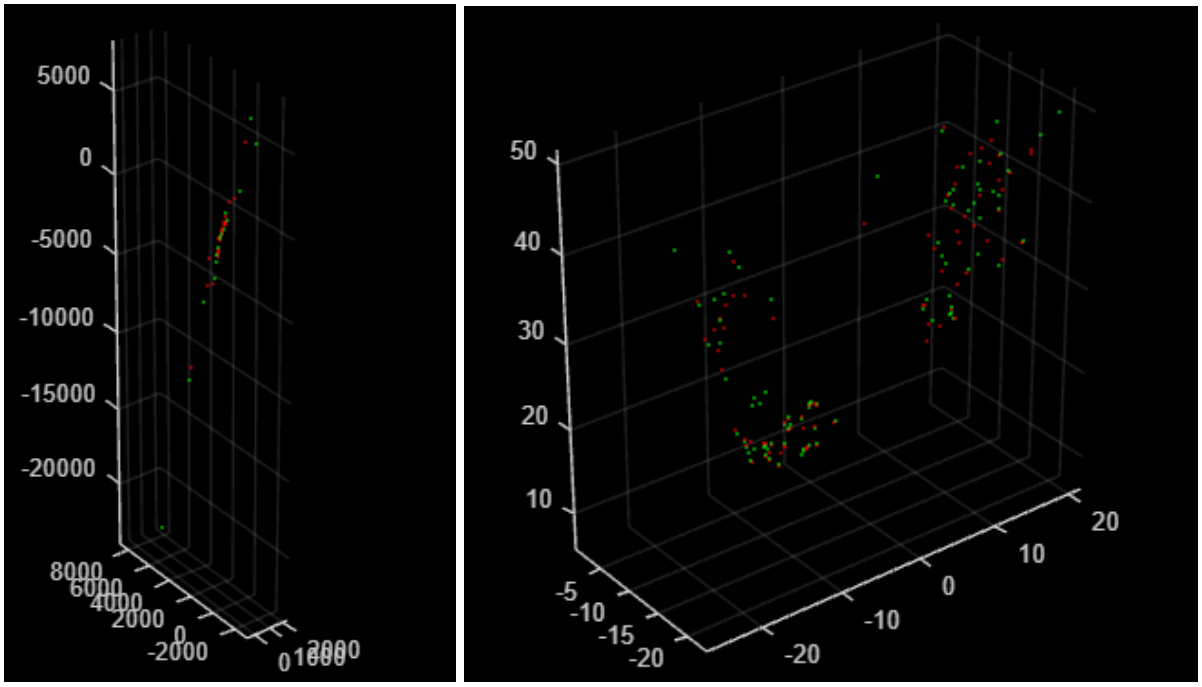

Figure 7: a) point clouds, before alignment b) point clouds, before alignment but after removing invalid areas

In the **3D-to-3D method**, 3D point clouds are triangulated from both stereo pairs and aligned using a rigid transformation model to estimate rotation and translation. And the translation vector is shown below

```
The result of 3D to 3D (translation) is -0.050976  0.00025272   -0.066513
```

While in contrast, the **2D-to-3D method** generates a single 3D point cloud from the first stereo pair, then uses 2D points from the second stereo pair to solve the Perspective-n-Point (P3P) problem, estimating the pose of the second camera relative to the first.

```
The result of 2D to 3D (translation) is 0.044324   -0.066302    0.057141
```

2.7 Performing sensitivity analysis using Monte Carlo:

In this section, we performed Monte Carlo simulations to analyze the sensitivity and robustness of the 3D-to-3D and 2D-to-3D alignment techniques under noise. Gaussian noise with a standard deviation of 0.1 pixels was added to the stereo correspondences, and each method was evaluated over 1000 iterations.

```
% Do Monte-Carlo sensitivity test for method 3D to 3D
noiseSTD = 0.1; % st dev of noise in pixels
numRuns = 1000; % number of runs
```

2.7.1 select_based_on_z_distance:

This function filters 3D points based on their Z-coordinate values, ensuring that they lie within a specified range zmin-zmax. A logical mask, validMask, identifies points that satisfy the condition, and only these valid points are returned. This helps eliminate outliers or points outside the region of interest.

This filtering ensures only the valid 3D points are used for rigid transformation estimation, improving accuracy under noise.

```
% Do Monte-Carlo sensitivity test for method 3D to 3D
function [points3D_t1_valid, points3D_t2_valid] =
select_based_on_z_distance(points3D_t1, points3D_t2, zmin, zmax)
% Check points that are in a section of the 3D space
%% your code here %%
validMask = (points3D_t1(:,3) >= zmin) & (points3D_t1(:,3) <= zmax) &
(points3D_t2(:,3) >= zmin) & (points3D_t2(:,3) <= zmax);
% Filter the points using the valid mask
points3D_t1_valid = points3D_t1(validMask, :);
points3D_t2_valid = points3D_t2(validMask, :);
end
```

2.7.2 mvg_montecarlo_3d_to_3d_reg:

This function performs Monte Carlo simulations to estimate 3d-to-3d rigid transformations under noisy conditions. It adds Gaussian noise to stereo point correspondences, triangulates 3D points, filters them based on z-coordinate and estimates rigid transformation. The following steps are carried out in each iteration:

1. ***Triangulation**:* Stereo correspondences are triangulated to generate 3D points at two time steps.

```
%% your code here %%
% Triangulation of 3D points from pair 1
points3D_t1_noisy=triangulate(pts_matches_l1,pts_matches_r1,stereoParams);
% Triangulation of 3D points from pair 2
points3D_t2_noisy=triangulate(pts_matches_l2,pts_matches_r2,stereoParams);
```

2. ***Filtering**:* The function **select_based_on_z_distance** is applied to remove invalid 3D points based on the Z-coordinate range.

```
zmin = 0.5; % all points must be farther than this, in meters
zmax = 50;  % all points must be closer than this, in meters
[points3D_t1_noisy_valid,points3D_t2_noisy_valid] =
select_based_on_z_distance(points3D_t1_noisy,points3D_t2_noisy,zmin,zmax);
```

3. ***Rigid Transformation****:* A 3D-to-3D rigid transformation is estimated between the two filtered 3D point clouds. We stored the translation vectors we obtained in **translationStack(iterIdx,:)**.

```
[estimatedTform_noisy]=estimateGeometricTransform3D(points3D_t2_noisy_valid,
points3D_t1_noisy_valid,'rigid','MaxDistance',10);
translationStack(iterIdx,:) = estimatedTform_noisy.Translation;
```

2.7.3 mvg_montecarlo_2d_to_3d_reg:

This function performs Monte Carlo simulations for 2d-to-3d registration under noise. It adds Gaussian noise to stereo correspondences, triangulates 3D points from the first pair, and estimates the 2D-to-3D pose for the second pair. The translation vectors translationStack. The following steps are carried out similar to the 2D-to-3D:

1. ***Triangulation****:* 3D points are triangulated from the stereo pair.

```
% Triangulation of 3D points from pair 1
points3D_t1_noisy=triangulate(pts_matches_l1_noisy,pts_matches_r1_noisy,stereoPar
ams);
```

2. ***Pose Estimation****:* 2D-to-3D pose estimation is performed for the stereo pair at t using the triangulated 3D points from t−1 and 2D correspondences from t.

```
intrinsics = stereoParams.CameraParameters1.Intrinsics;
[~,t]=estimateWorldCameraPose(pts_matches_l2_noisy,points3D_t1_noisy,intrinsics,'
MaxReprojectionError', 10);
translationStack(iterIdx,:) = t;
```

The translation distributions are shown in FIG and FIG using pcshow. Two separate point clouds are plotted to compare the translations from the 3D-to-3D and 2D-to-3D methods respectively.

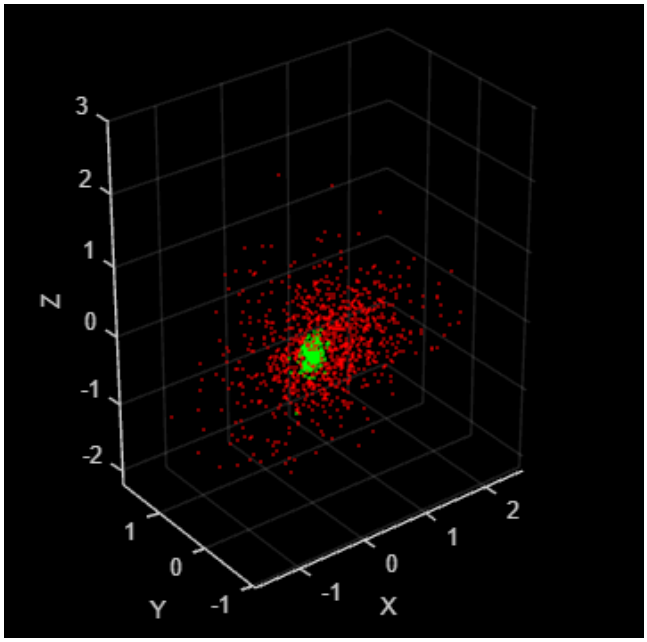

Figure 8: Monte Carlo sensitivity analysis for 3D-to-3D and 2D-to-3D 1

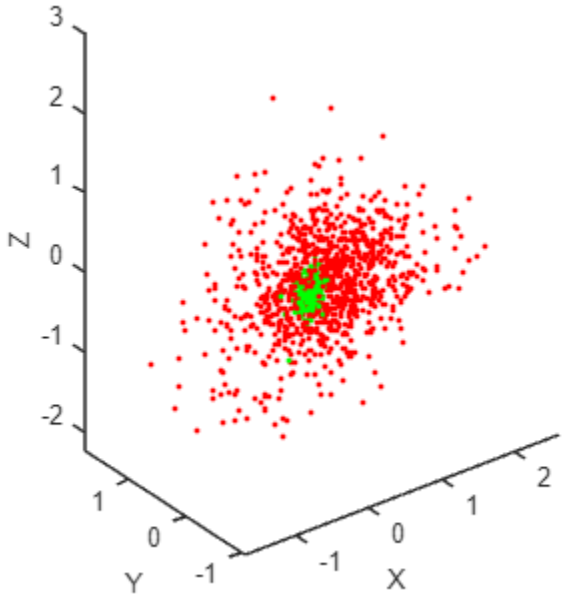

Figure 9: Monte Carlo sensitivity analysis for 3D-to-3D and 2D-to-3D 2

Monte Carlo analysis Part Questions: The 2D-to-3D method uses 3 images whereas the 3D-to-3D method uses 4. However, the 3D-to-3D method performs considerably worse. Please provide a justification on what may be the cause of this.

Answer: The reason why the 3D-to-3D method performs worse than the 2D-to-3D method is because of triangulation errors accumulated across 3D-paired points, which may also have limited overlap due to occlusions before transformation estimation. These errors are caused by

noise in feature points or as a result of calibration, which affects the accuracy of the 3D-to-3D. And if there are occlusions, the ambiguity in aligning the points will increase. Also during triangulation, the 3D-to-3D method may have more outliers since we are comparing 3D paired points. The outliers would make the rigid transformation we estimate deviate more.

2.8 Processing the Complete Sequence:

Here we process the entire sequence of stereo image pairs to estimate the trajectory of the camera in the reference frame. For each stereo pair, features are extracted, matched using circular matching, and triangulated to obtain 3D points. Using valid 3D points from consecutive frames, the relative motion (rotation and translation) is estimated through camera pose estimation. The trajectory is then updated iteratively, and the history of positions is stored. The final plot shown in Figure 10 is the 3D space of the camera trajectory based on the stereo-visual odometry process.

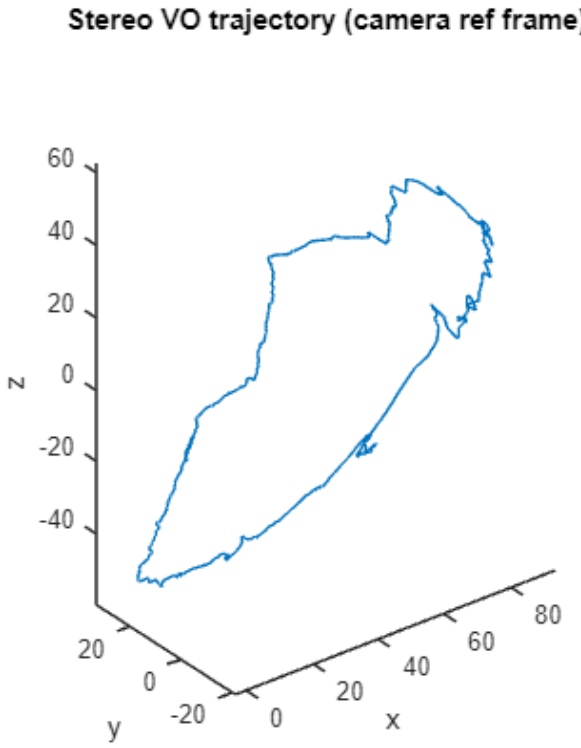

Figure 10: Stereo VO trajectory projected in 3D space

The GPS trajectory data in the dataset is provided as the ground truth, and we use this *helperTransformGPSLocations* function to align them with the visual odometry trajectory. We do this by calculating the relative yaw between the initial segments of both trajectories. It adjusts the GPS coordinates to match the VO reference frame through a transformation involving rotation and translation. The result is shown in Figure 11.

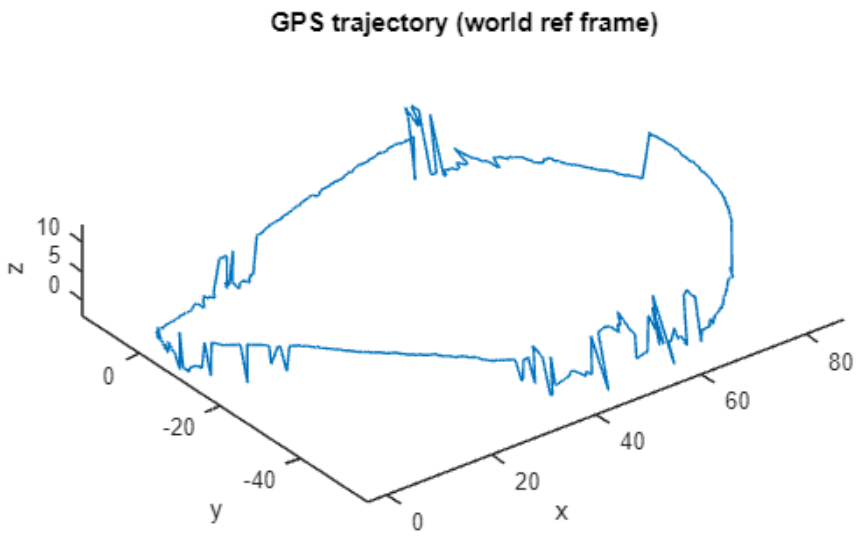

Figure 11: GPS trajectory projected in 3D space

Questions

As you can see, the simple methods seen in this example have modest performance. How could it be improved? Define and describe additions to these methods, or alternative strategies that you consider that would **improve the performance in terms of accuracy**. Focus you answer on the 3D reconstruction and geometric aspects and not on the navigation and filtering aspects (such as EKF or SLAM). Use between half a page to one page for your answer.

Answer: The accuracy of these methods can be improved by the following means:

1) Using multiple-views:

   Relying on a single stereo pair for triangulation can introduce inaccuracies due to noise or sub-optimal viewing angles. By incorporating multiple stereo pairs, the 3D point estimation becomes more robust:

   a) **Redundancy**: Triangulating the same 3D points from multiple views can reduce the impact of noise or outliers in any single frame.
   b) **Improved Baseline**: Points seen across a sequence of stereo frames have a larger effective baseline, leading to more accurate depth estimation, especially for distant points.

2) Using SIFT instead of SURF:

   Since features are matched across stereo image pairs and across frames, transformations such as scale, rotation, and translation are involved due to the motion of the cameras and environmental factors, and our focus is improving reconstruction

accuracy. The use of SIFT, even though it may be slower than SURF, is more robust than SURF in handling these transformations, reducing the likelihood of mismatched features, especially in challenging scenarios.

3) Using depth data from sensors:

   If views have similar structure or patterns but have differences of depths. Using depth sensors like LiDAR or depth cameras can constrain and refine 3D points in ambiguous regions.

4) Refining Triangulation:

   Instead of basic triangulation, iterative non-linear optimization can minimize reprojection errors across all views, leading to more precise point reconstruction.

2.9 Align the two trajectories and then plot the two trajectories aligned:

"Align the two trajectories and then plot the two trajectories aligned. To do this, compute the transformation that best aligns the first 25% of the VO trajectory w.r.t. the GPS trajectory. Then apply that transformation to the complete VO trajectory. Note that the z coordinate of the GPS data may have outliers, and that we are only interested in visualizing the two trajectory on the ground plane."

The normalized Z axis of the GPS trajectory is shown in Figure 12.

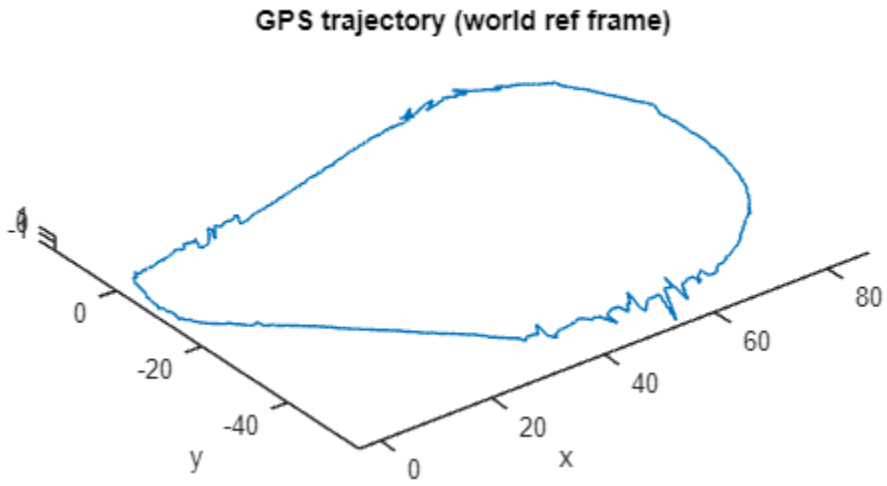

Figure 12: Normalized z axis of the ground truth (GPS)

2.9.1 Setting Z of GPS to Zero

Here, the Z-coordinate of the GPS trajectory is set to zero, constraining the alignment to the ground plane. The first 25% of the VO and modified GPS trajectories are used to estimate the transformation. This adjustment eliminates z-coordinate outliers, resulting in better alignment in the XY-plane, which is particularly useful for visualizing and comparing ground-level trajectories.

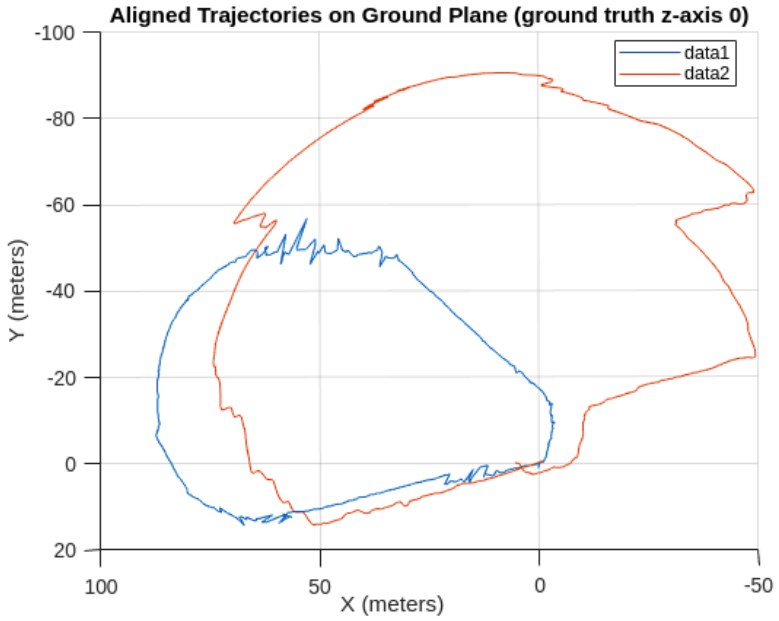

Figure 13: Aligned Trajectories on Ground Plane

2.9.2 Setting Z of GPS to the Mean Value

In this case, the Z-coordinate of the GPS trajectory is replaced with its mean value. This approach smoothens the trajectory in the Z-dimension while preserving some vertical information. The first 25% of the VO and modified GPS trajectories are used for alignment, providing a compromise between ground-plane visualization and maintaining Z-axis consistency.

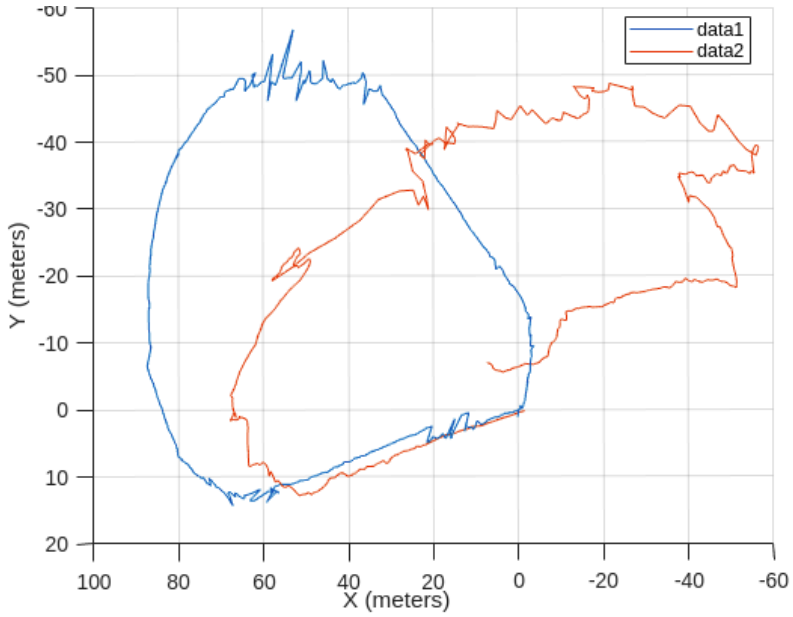

Figure 14: Aligned Trajectories on z-mean

2.9.3 Using Optimization for Alignment

This method optimizes the transformation parameters using *fminunc*, which is a Matlab function for nonlinear optimization to minimize alignment errors between the first 25% of the VO and GPS trajectories. A custom error function computes the geometric differences, and the optimized transformation is applied to align the entire VO trajectory. This approach ensures a globally refined alignment and handles noisy data more robustly by minimizing overall alignment errors.

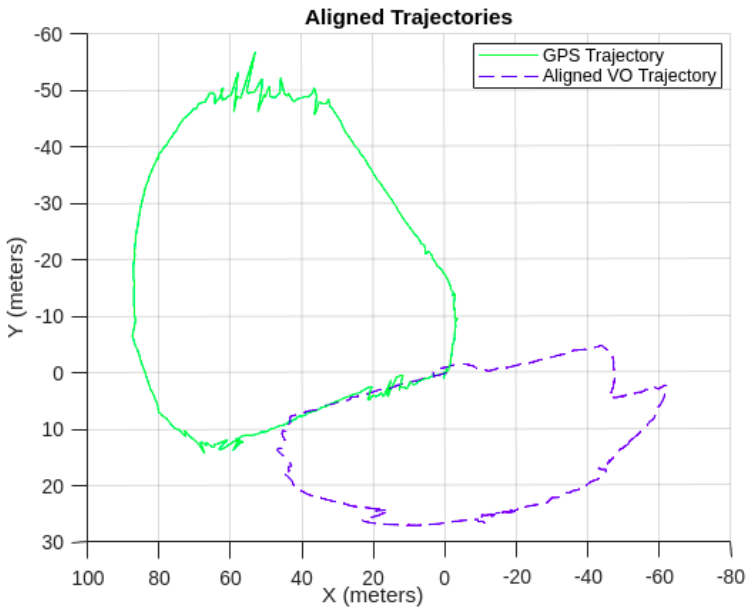

Figure 15: Aligned Trajectories