Pyone Thant Win
CMSC 426: Computer Vision
Professor Yannis Aloimonos
December 12, 2021
Worked alone on this project

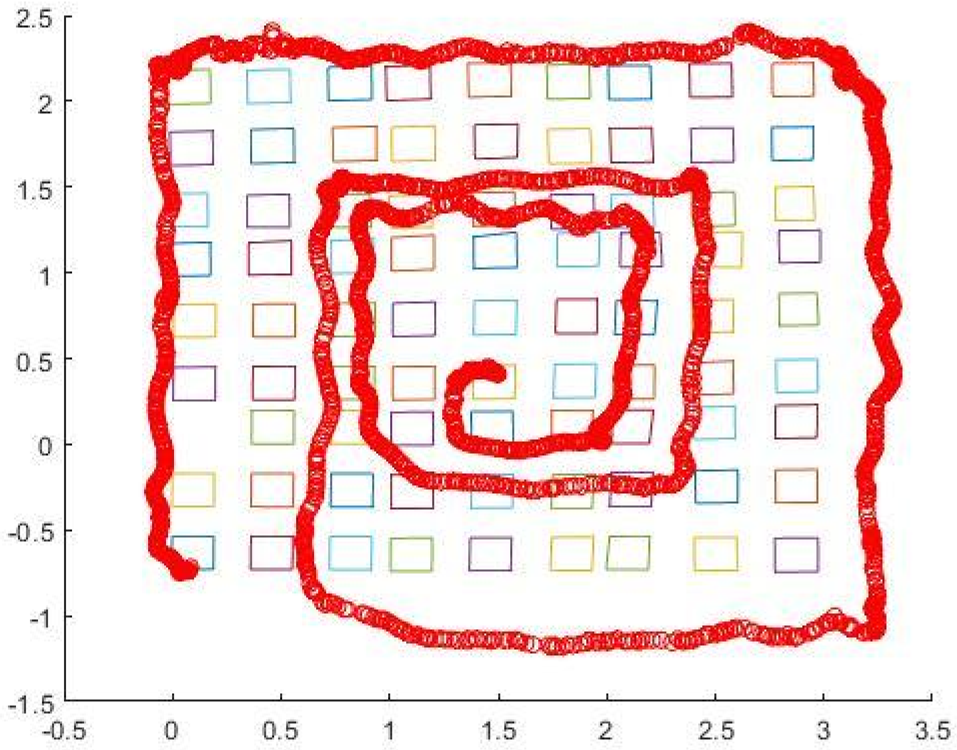
Pre-GTSAM

We are given the image coordinates of the tags at each frame, tag size in real world, and the camera intrinsic calibration matrix (K). I started by setting tag 10 left bottom corner at world coordinate origin. World coordinates were set as x = [0 TagSize TagSize 0] and y = [0 0 TagSize TagSize]. Since the assumption is that all tags lie on an even surface (ground), getting world coordinates of tags and camera poses at each frame can be solved using homography. Homograph H is obtained based on Tag 10's image coordinates and world coordinates. H is then normalized by dividing with the last element of H. Then, I calculated the rotation matrix with K inverse * H = [h1' h2' h3']. Rotation matrix (R = [r1 r2 r3]) is then further derived by performing SVD on [h1' h2' h1'xh2']. The final and best estimation of the rotation matrix is calculated as U * [1 0 0; 0 1 0; 0 0 det(U V')]. U and V were the result of svd earlier. As a next step, translation matrix (T) was calculated by h3' / norm(h1'). Camera pose could be obtained by getting the inverse of (K * [r1 r2 T]). Once I got the camera pose value of the frame, I applied homography to the rest of the tags in the frame based on the camera pose. After this step, I got the world coordinates of the tags in the first frame. Tag 10's world coordinates were set as [0 0 0 0 0 0 0 0] from p1 to p4.

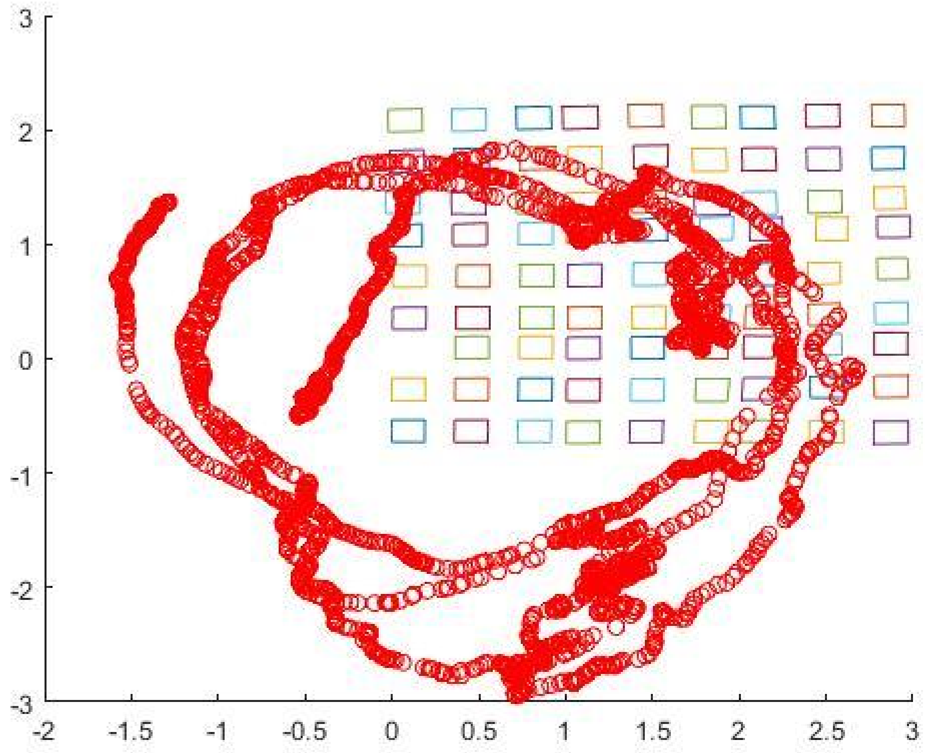
From the second frame until the last frame, all the visible tags were separated into two categories. The first category was the tags that were previously visible and already calculated the world coordinates of. The second category was new tags that were never seen before. I estimated the homography value of the tags using all known tags in a current frame. Then, I did the same procedures of calculations I did for the first frame to obtain the camera pose of the current frame, and the rotation and translation matrices that correspond to the camera pose. Homography was applied to all new tags. The process was repeated until the last frame. During the process, I saved the rotation matrix, translation matrix and the final translation matrix that includes the rotation values as well.

Here are the results after the initial camera poses estimation and the initial tags' world coordinates estimation.

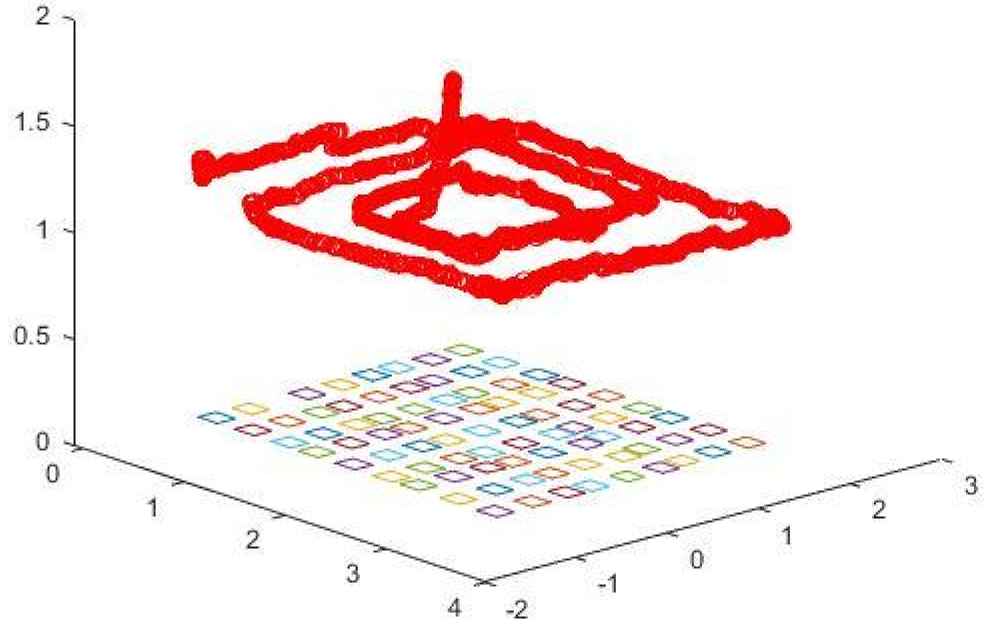
- DataMappingFall2020
  - Top view (T_)



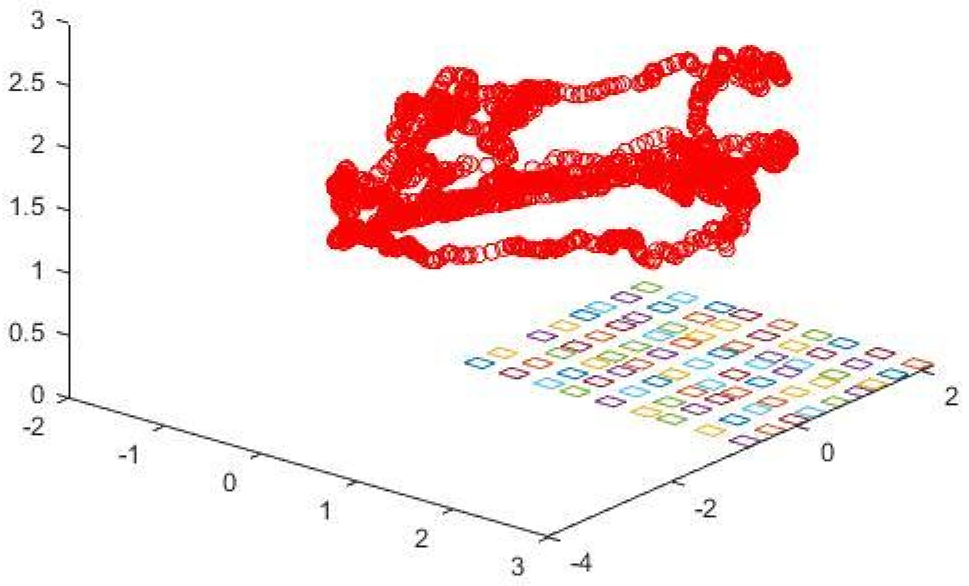
-
- Top view (T)



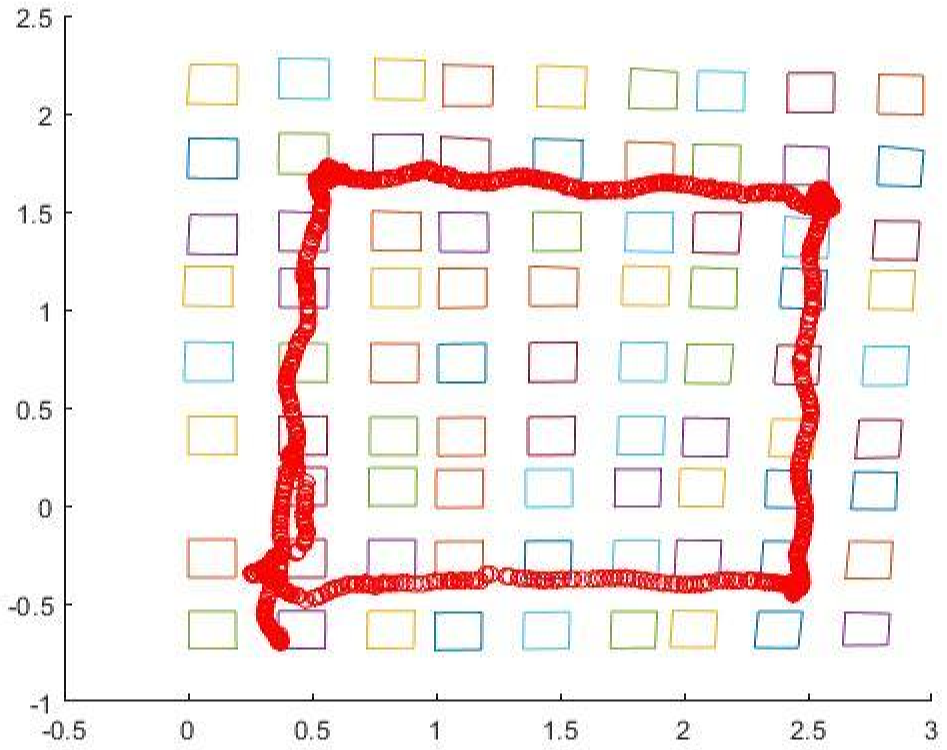
-
- Side view (T_)



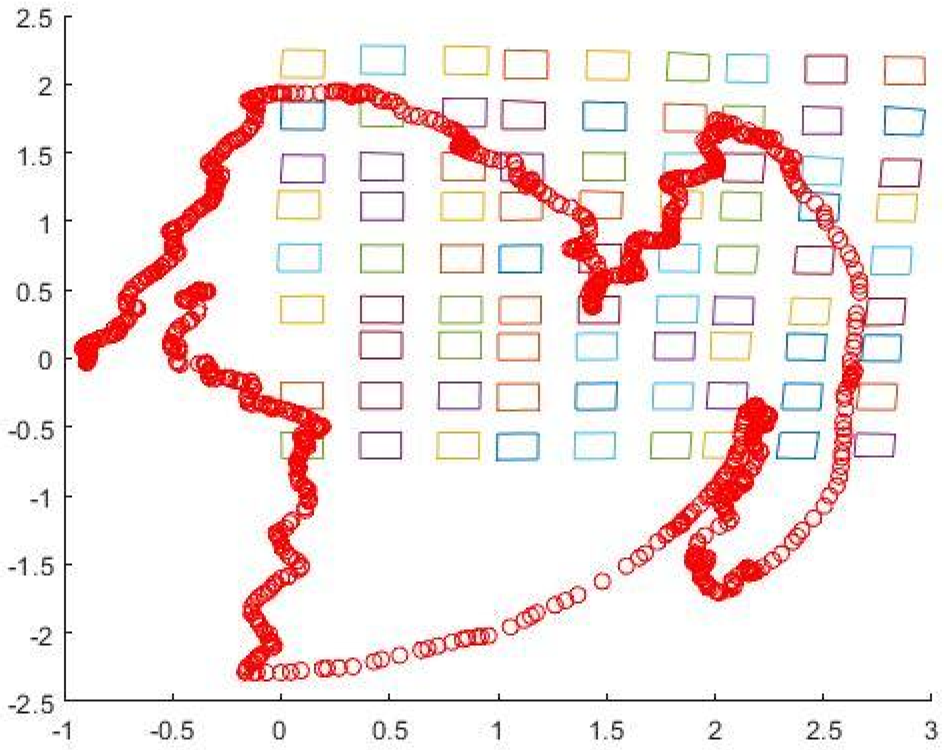
-
- Side view (T)



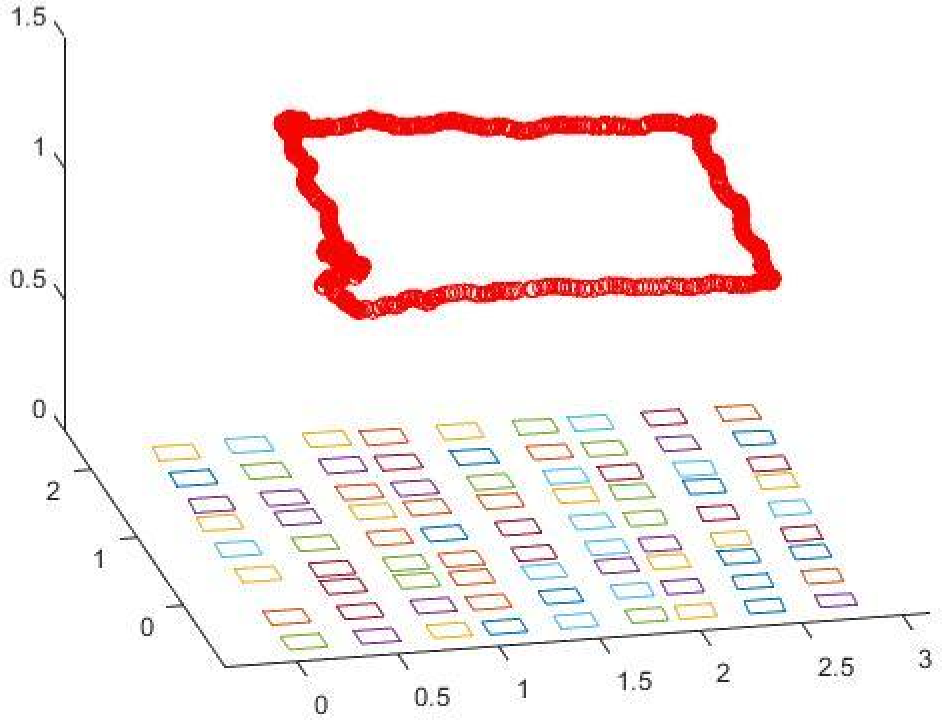
-
- DataSquareFall2020
  - Top view (T_)



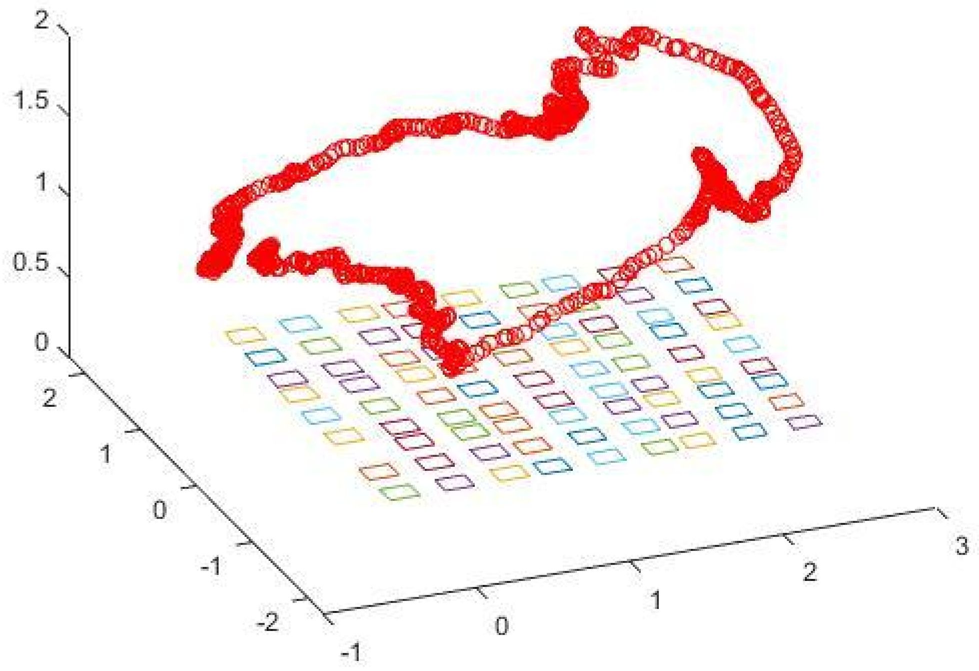
-
- Top view (T)



-
- Side view (T_)



- ○
- ○ Side view (T)



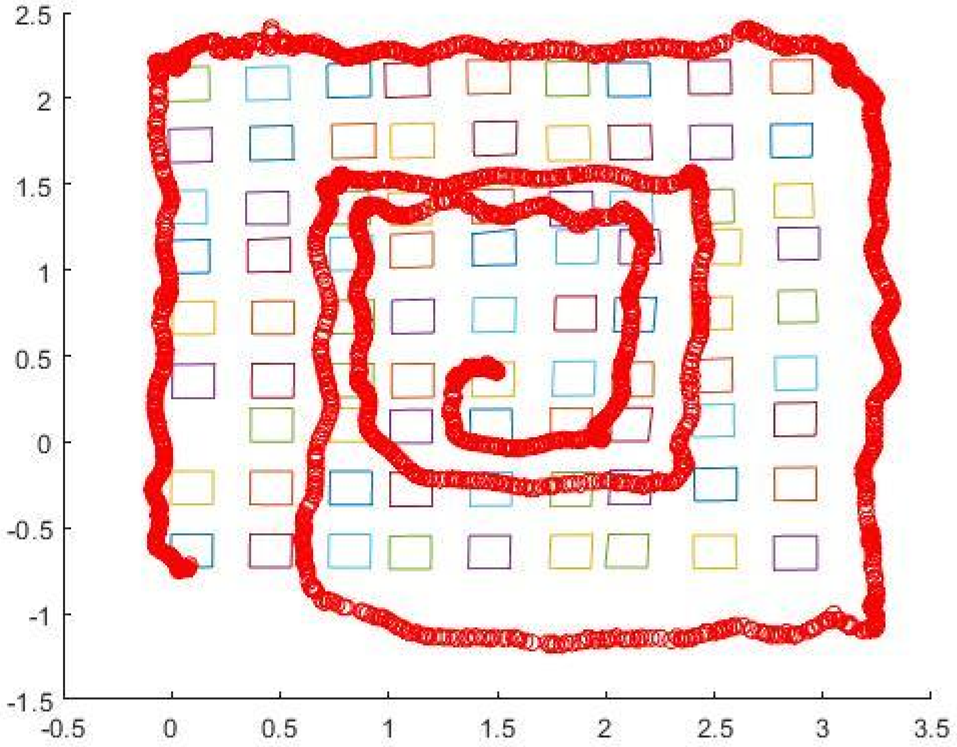
○

GTSAM

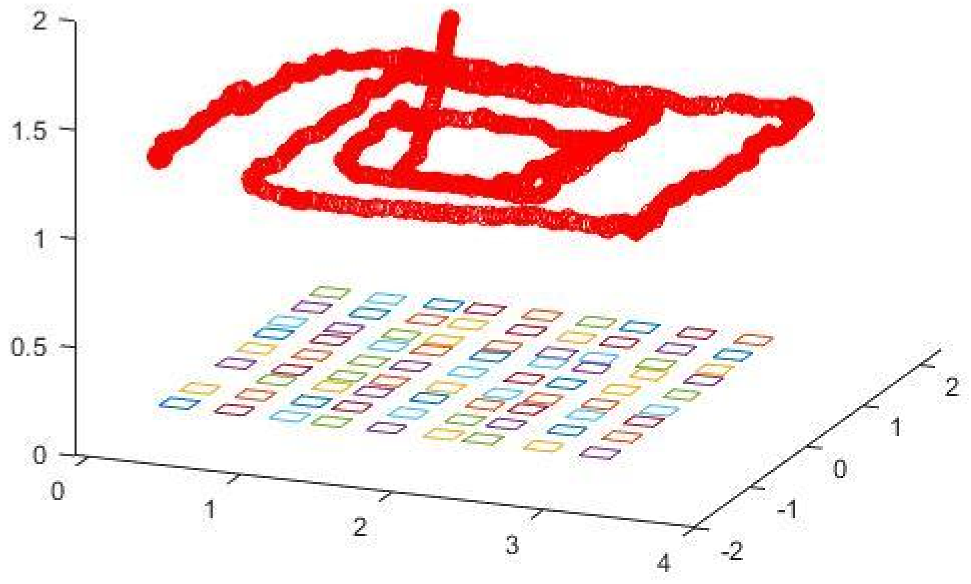
Since we are using the pre-built toolbox for this part, there was no math directly involved in the implementation. GTSAM was used to optimize the initial estimates that I calculated earlier. In the real world, initial estimates are going to be noisy and the camera calibration values cannot always be 100% accurate, so algorithms like GTSAM are necessary to optimize the estimates.

After defining the nonlinear graph object to initialize the GTSAM model, we added some synthetic noise for GTSAM to be able to perform better. I added measurement noise variance of [0.1; 0.1], point noise variance of 0.1, pose noise variance of [0.001; 0.001; 0.001; 0.1; 0.1; 0.1] and odometer noise variance of ([0.1; 0.1; 0.1; 0.1; 0.1; 0.1]. Then, I added the prior factor which was the tag 10 world coordinates, and camera pose and rotation of the first frame. I also added the initial estimate camera poses values of all frames and the world coordinates of the tags. Each node in the graph was connected with a BetweenFactorPose3 object. BetweenFactorPose3 was constructed using the identity matrix as the rotation matrix, 3x1 zero matrix as the translation matrix, and odometer noise. I chose the LevenbergMarquardtOptimizer as DoglegOptimizer was giving me an error that I was not able to debug. One interesting find was that the optimized output and the initial estimated output were extremely similar, which means that we were lucky enough to receive very accurate input values.
The results are:

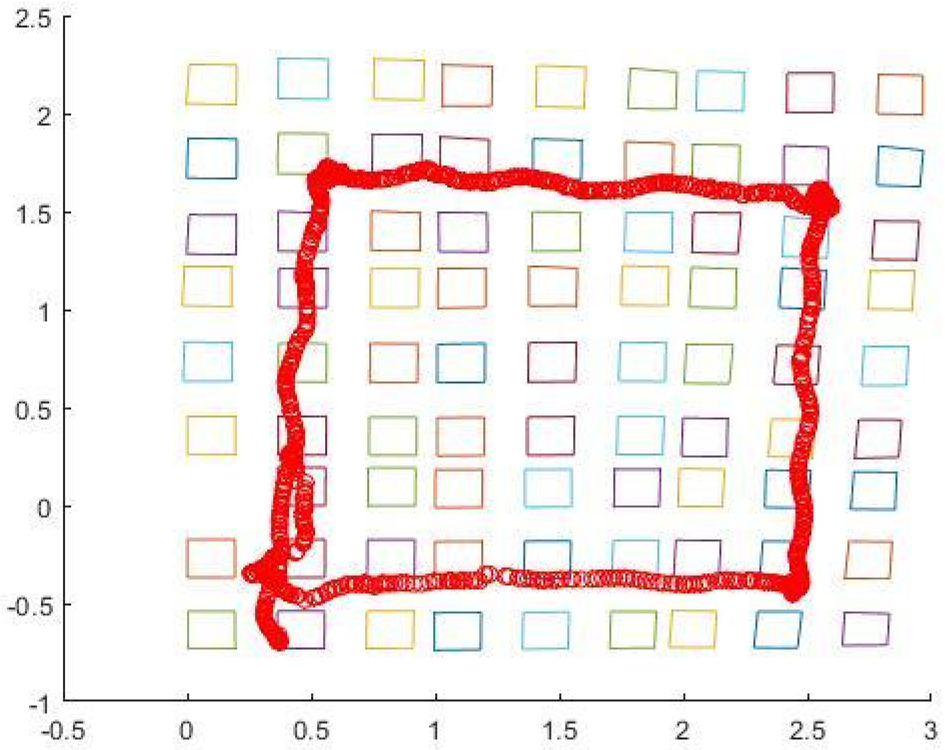
- DataMappingFall2020
  - Top view (T)



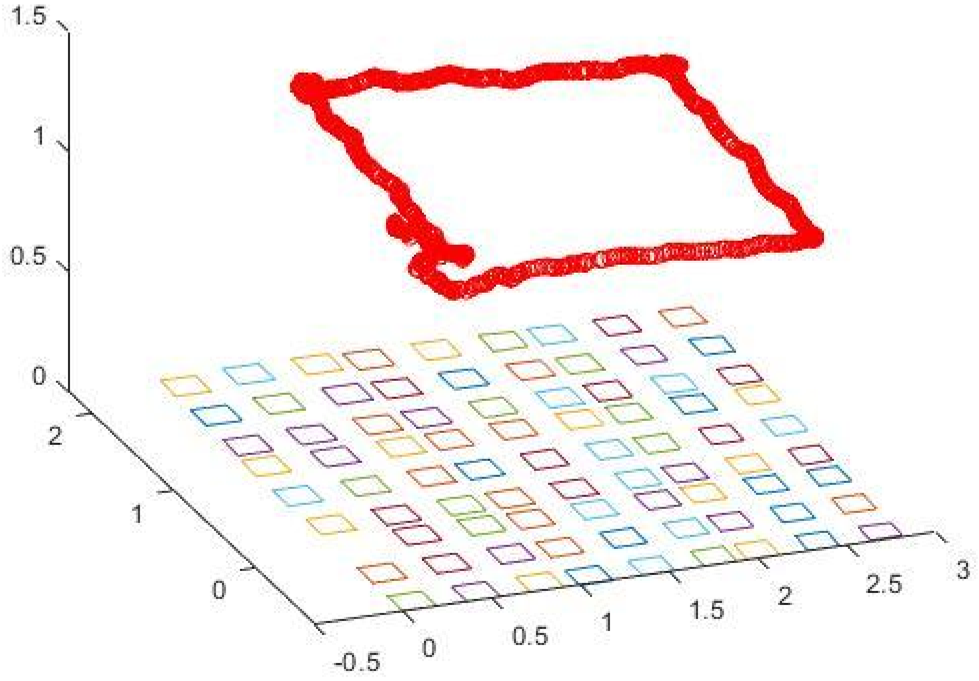
  -
  - Side view (T)



-
  -
- DataSquareFall2020
  - Top view (T)



-
- Side view (T)



○

Challenges

Although there were minor challenges related to the math and matlab functions, I did not encounter any huge challenges. One example was that the matlab's svd() function output U, S, and V. I initially mistaken it as V transpose, but I was able to debug with the help of TA. Additionally, GTSAM documentation was a challenge to understand as there were not many code examples in the documentation although Nitin's code was helpful and TAs were also really knowledgeable and explained well for me to understand and code them properly.

There was one thing that I was not entirely sure of, so I also am including the results of the method I tried below. When I calculated the homography estimation of tag 10 and its corresponding world coordinates, I used x = [0 TagSize TagSize 0] and y = [0 0 TagSize TagSize]. However, I put the tag 10 world coordinates as x = y = [0 0 0 0]. Therefore, in the later frames, this value was used to estimate and apply homography of new tags. However, it made more sense to me that I use x = [0 TagSize TagSize 0] and y = [0 0 TagSize TagSize] as tag 10 world coordinates for the new tag homography estimation as well so I tried and here are the results for the mapping dataset. I expected this result to be better but they were the same, surprisingly.

DataMappingFall2020 (left to right: Pre-GTSAM T, T_, GTSAM T):