CMSC426 Project 4

Final Project

Group Members: Marc Haller, Anthony Mahshigian, Matthew Wong

Contributions

Marc Haller, Anthony Mahshigian, Matthew Wong

We decided for this project we would be most effective if we met and did the project together the whole time, instead of trying to assign things separately. This way, there were much less gaps in knowledge and we all spent equal time working on the project. We developed together with group input for the most part. We were able to take advantage of Matlab Drive, something we were unaware of for the other projects. This was incredible in making sure we could all see each others' edits in almost real time. Zoom screen sharing was also a big help to talk about live edits. More specifically, Marc led the way with the initial frame and started the pipeline, Anthony worked on matching tags iteratively, and Marc and Matt were able to finish poses and get images. Marc and Matt were able to implement the GTSAM toolbox, and Anthony was responsible for keeping track of progress so we could summarize things in the report in an effective way. Overall, we had a very collaborative process and were able to save plenty of time this way. We were also able to make sure we all knew what was happening at each step, and we could catch each others' errors, which we did often.

Process

Introduction

The purpose of this project is to find the 3-Dimensional location and the pose of the drone camera in the world frame based on the detection of AprilTags from the provided frames. We are also trying to map the world around us. The key with simultaneous localization and mapping (SLAM) is that we must deal with these two variables, and can only build estimates. Luckily, we have constraints that we can force with a certain degree of accuracy to guide and correct our estimates.

Algorithm

- Solve Homography
    - First, we chose AprilTag 10's bottom left corner to be the origin of the world. We then extracted the corner points and computed the initial pose of the camera with

    the homography. Because we know that the drone is moving relatively slow, we also know that there should be common corner points (landmarks) between two poses. This means that we can estimate the homography matrix for the remaining camera poses by matching tags computed in the previous poses.

- Build Factor Graph
    - For building the factor graph, we added in a bunch of constraints based on measurements we had. This included: the prior for the first pose, the prior for the world origin, the distance between poses, the measurement of each real world point onto the projected image, and the distance between the corners of a tag
- Initial Estimate
    - For the initial estimate, we added in the values we calculated based on the first part (homographies). This included: the poses we estimated, the real world positions of the landmarks we estimated
- Optimization
    - Now that we have our finished factor graph and initial estimates, we can put those values into a function called LevenbergMarquardtOptimizer from the GTSAM package to optimize those results. We can also show these results in a 3-D representation.

Problems Encountered

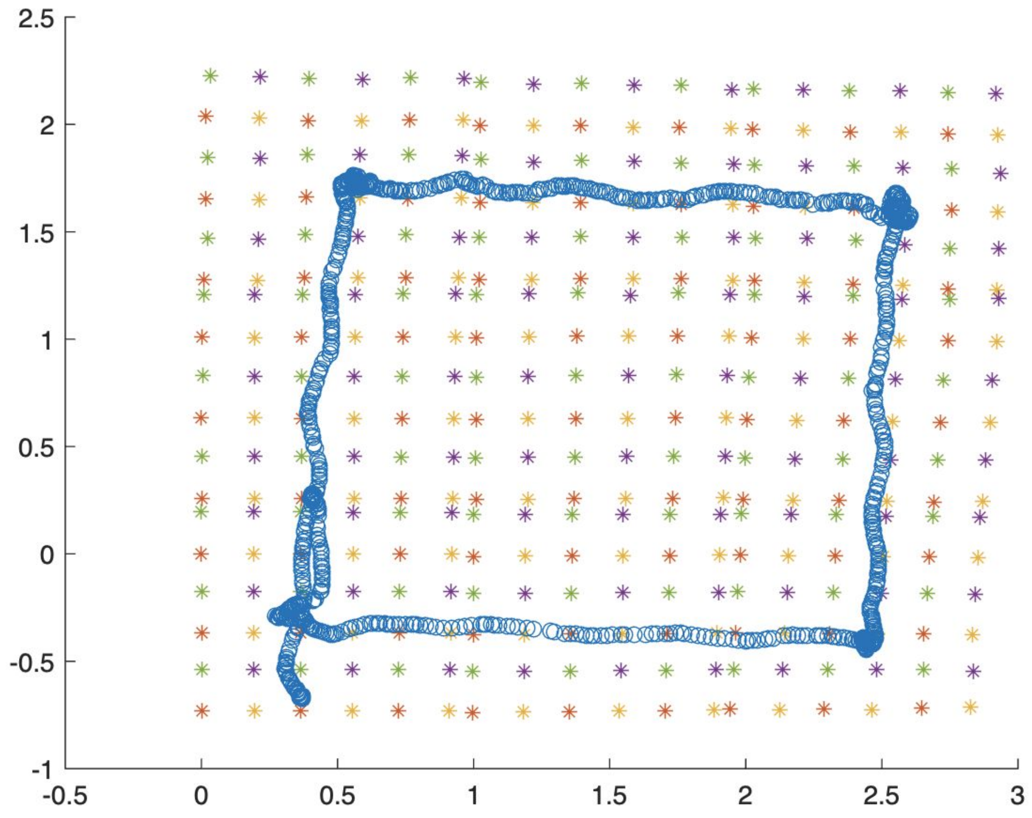
- At first, we did not use the pointsToWorld function. This presented big challenges for us, since we did not have accurate real world data points (they were all overlapping) and we did not know where the issue was along the pipeline
- We moved to pointsToWorld and were able to debug much easier how we were using the homography incorrectly
- We only had one windows computer which presented a problem since we all wanted to be able to test and run code, but we were able to use Matlab Drive to share documents, keep everyone's version up to date, collaborate, and use the GTSAM toolbox. It can be used when using Matlab Drive
- Another problem we encountered was not setting up the initial corners of the first AprilTag correctly for the estimated homography. This gave us problems with debugging because we were generating the wrong homography and since we were matching different corners to each other, the estimated real world points for the AprilTags were very wrong and looked like they overlapped.
- The camera poses and landmarks could be varied because our calculated values for each frame can also be varied from minor errors such as detection. Poses can also be varied because they are correlated with the landmarks.
- Understanding the parameters needed for using GTSAM was a challenge for us because we did not read the documentation for GTSAM that thoroughly and the given documentation of GTSAM is not very straightforward. There also weren't many good examples for implementing the GenericProjectionFactor.
- We can definitely notice drift error as the program advances. This is an issue with dealing with only estimates, in that errors will only build upon themselves. We are able to enforce constraints, but it is still possible for errors to propagate.

- We had issues getting GTSAM to run: DoglegOptimizer never worked, so we used Levenguard Marquardt Optimizer
- High sigmas on the poses would mess up the data extensively, especially when not using the identity transformation as our betweenpose factor.

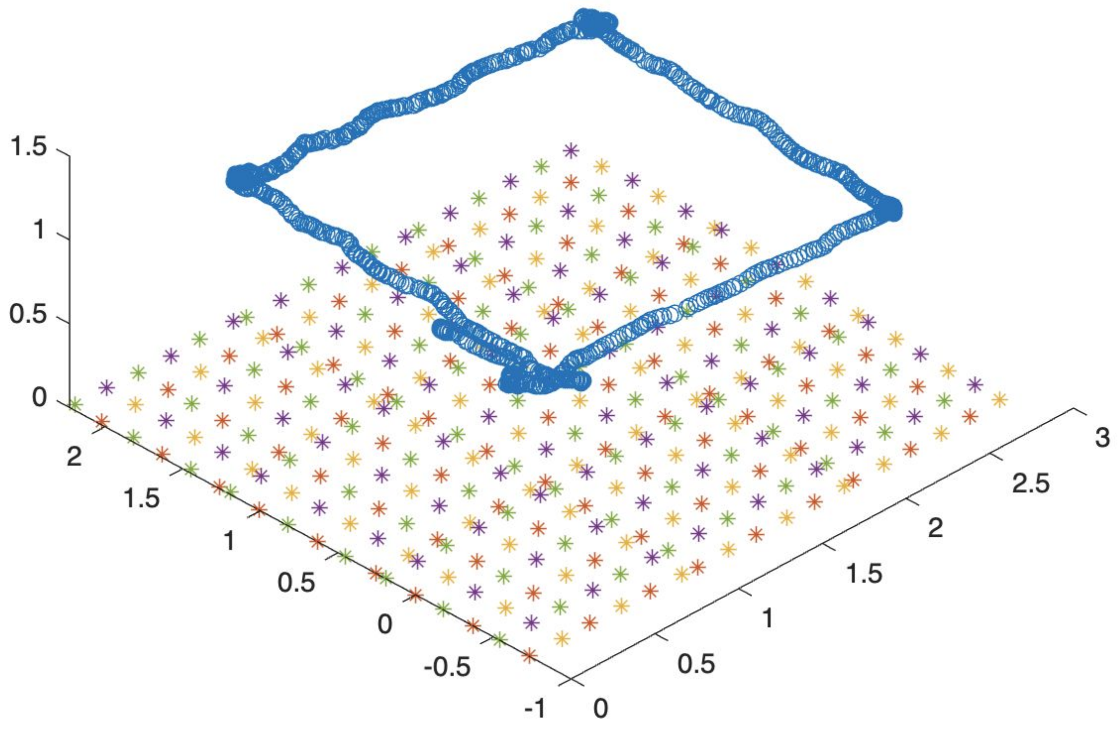
Results
(Without GTSAM)

We were able to get great results without GTSAM. Obviously, these are not perfect and there is some inaccuracy, but we're able to paint a very clear picture for the shape of the path. The exact values need more work, but the shape is there and it looks good. We can also see a fair amount of drift in that the errors propagate as the drone advances through the scene. At the starting point near the origin, the landmarks look to be in a uniform pattern, as the pose moves, it seems to lean and the landmarks fall out of pattern more and more.

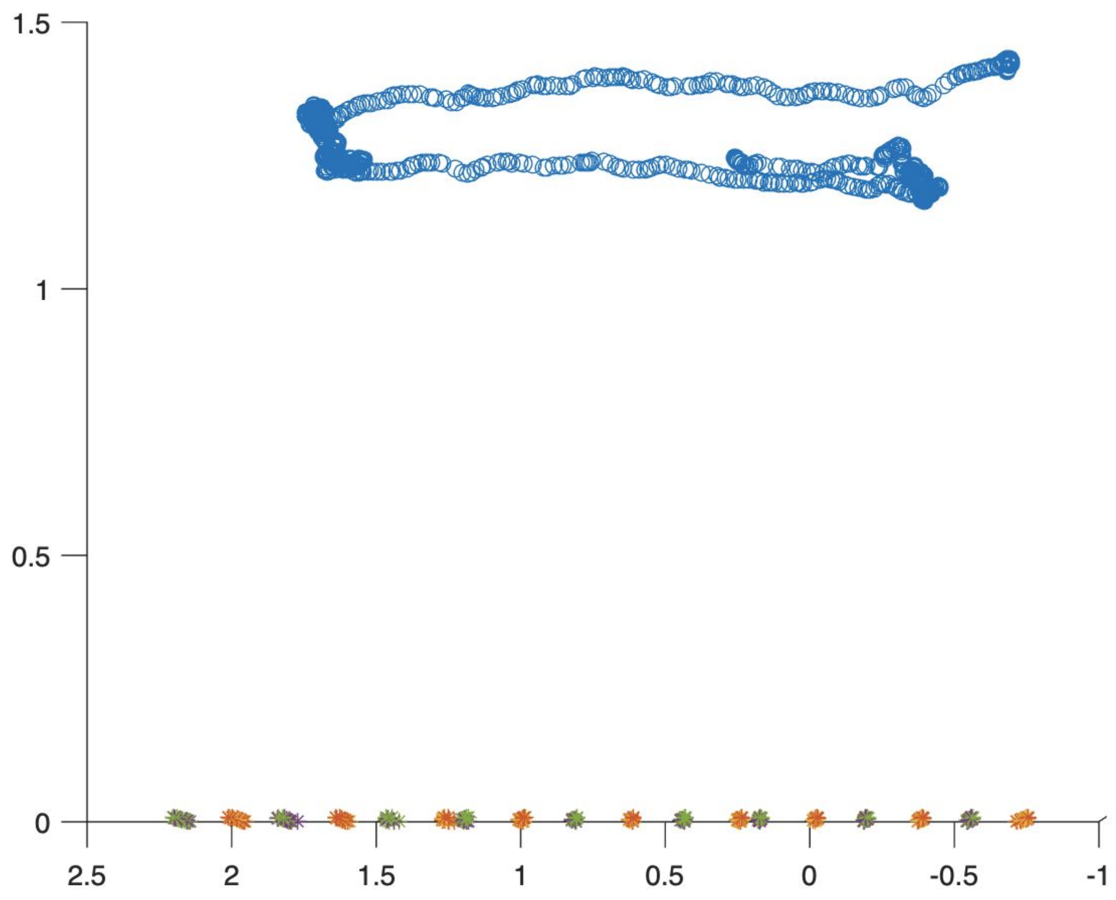
<u>DataSquare.mat</u>
(Top View)



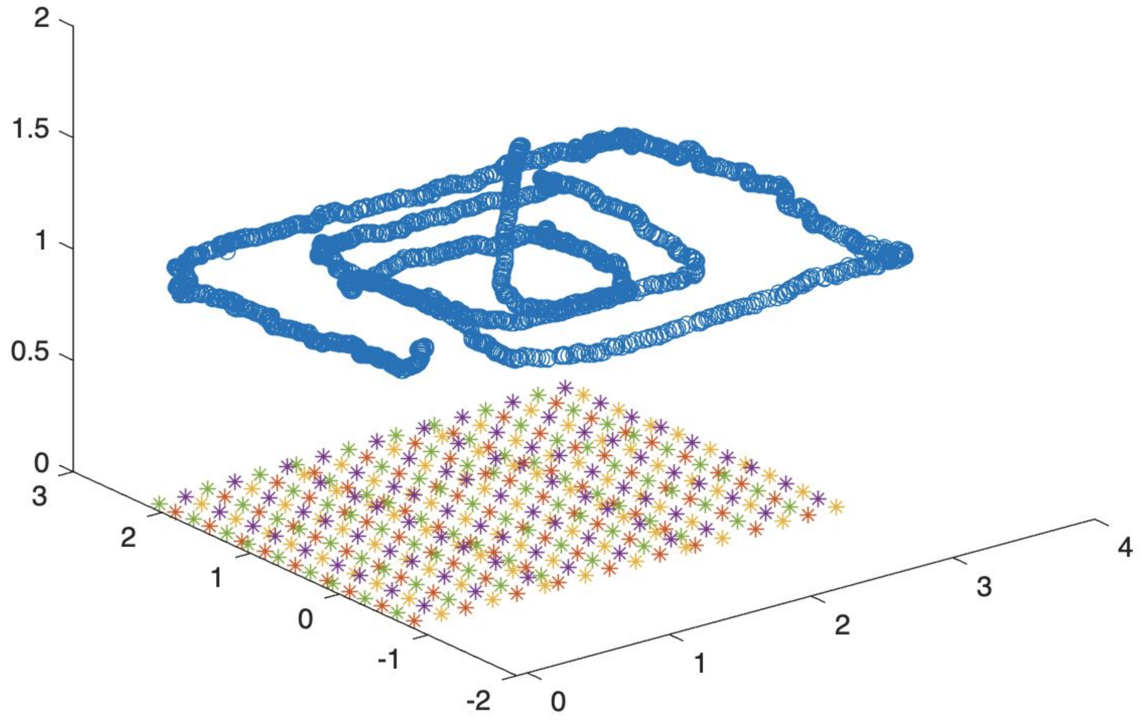
(Isometric View)



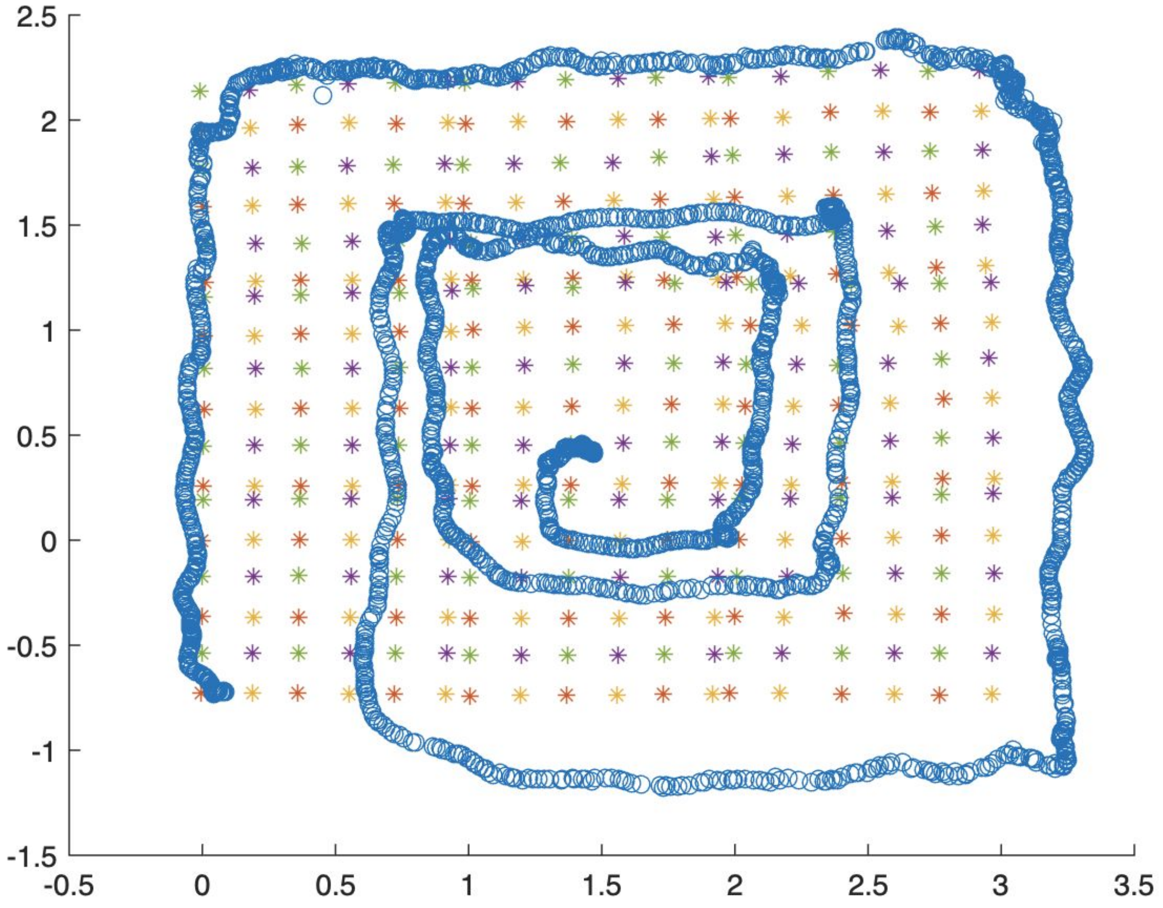
(Side View)



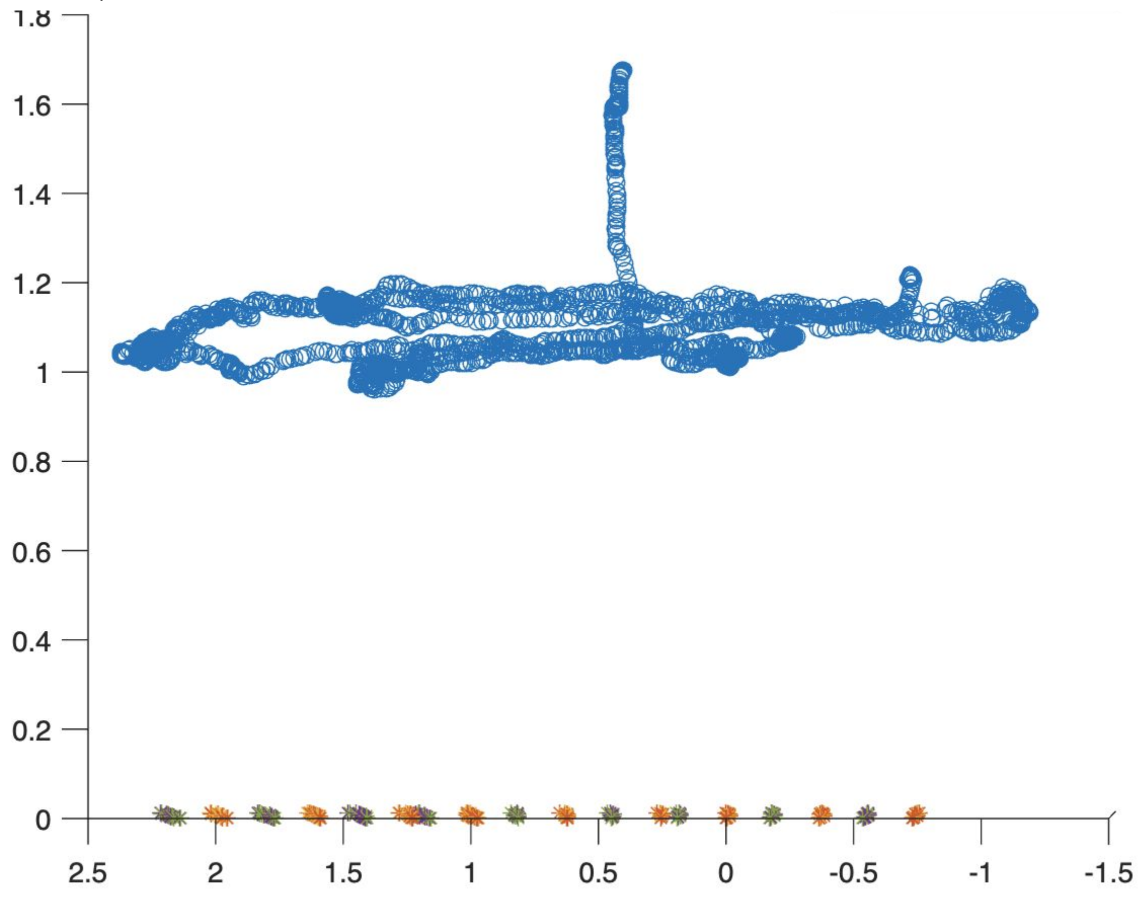
DataMapping.mat
(Top View)



(Isometric View)



(Side View)



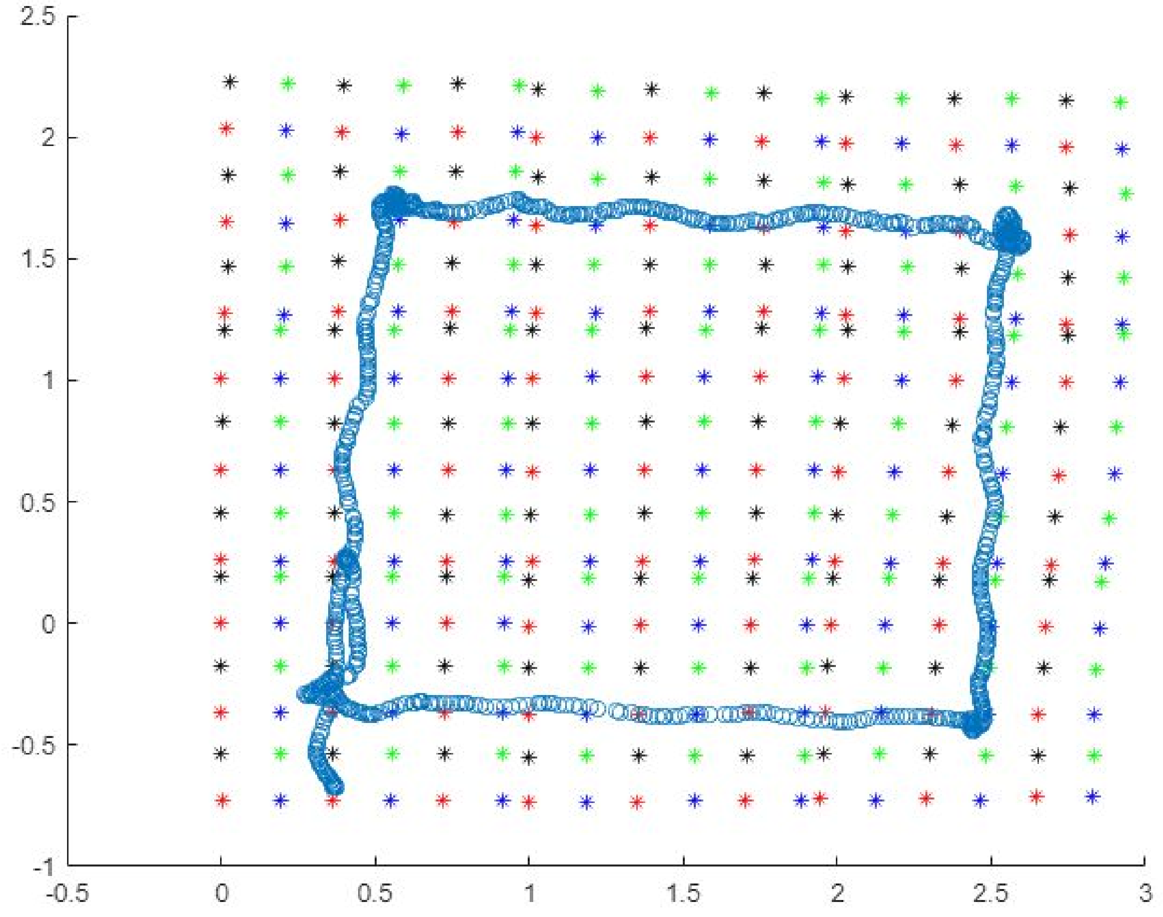
Results
(With GTSAM)

GTSAM was able to improve our results. Instead of just estimations, we now have constraints that we can enforce to try and correct errors (to a certain extent) before they propagate. SLAM is not perfect, but this is a much more intelligent way for it to be done. The improved results are shown below. We can see drift errors are less and we have a greater amount of localization and mapping consistency.

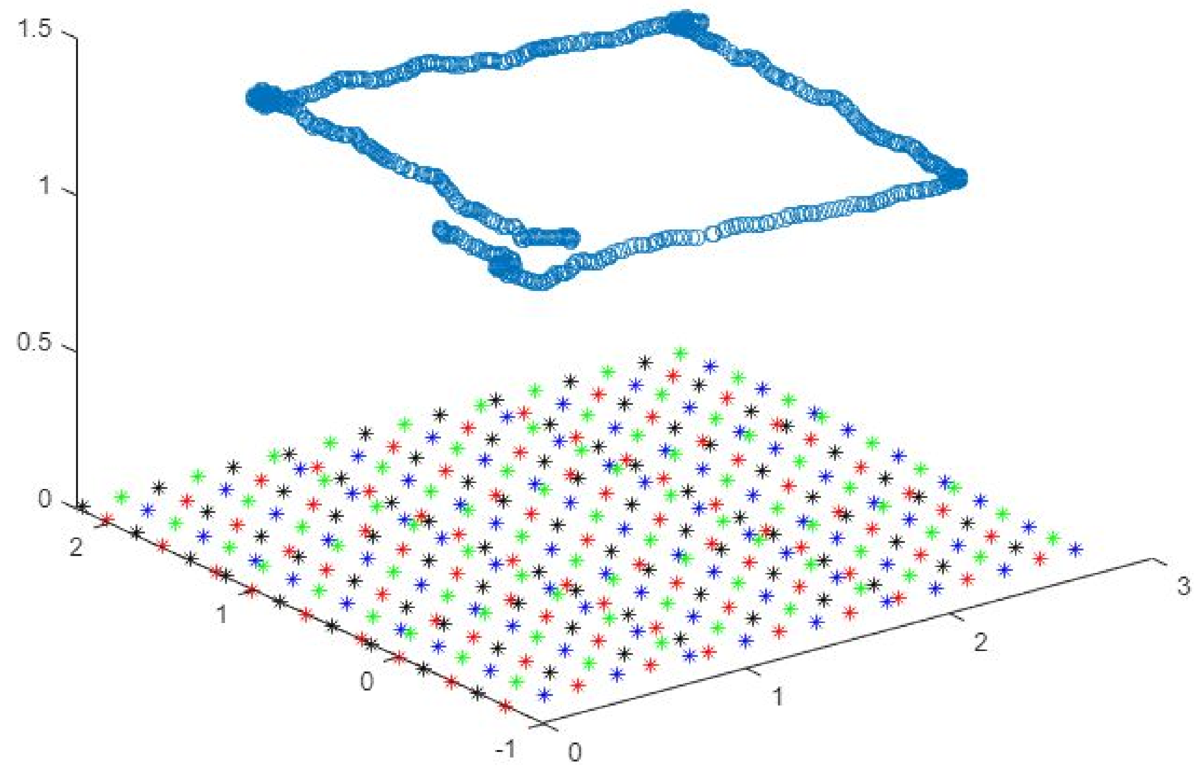
However, our GTSAM results were actually very similar to our initial data. If you look at our data, especially at the square, you'll notice a pretty smoothe graph with not many gaps/errors. While we don't really understand the lambda parameter for the optimizer, the output showed that the old error was consistently less than the new error for the square data set in some cases, resulting in minimal/no change.

We noticed the largest changes to the Z axis for the poses, as well as slight improvements to the landmarks not spreading out at the end.

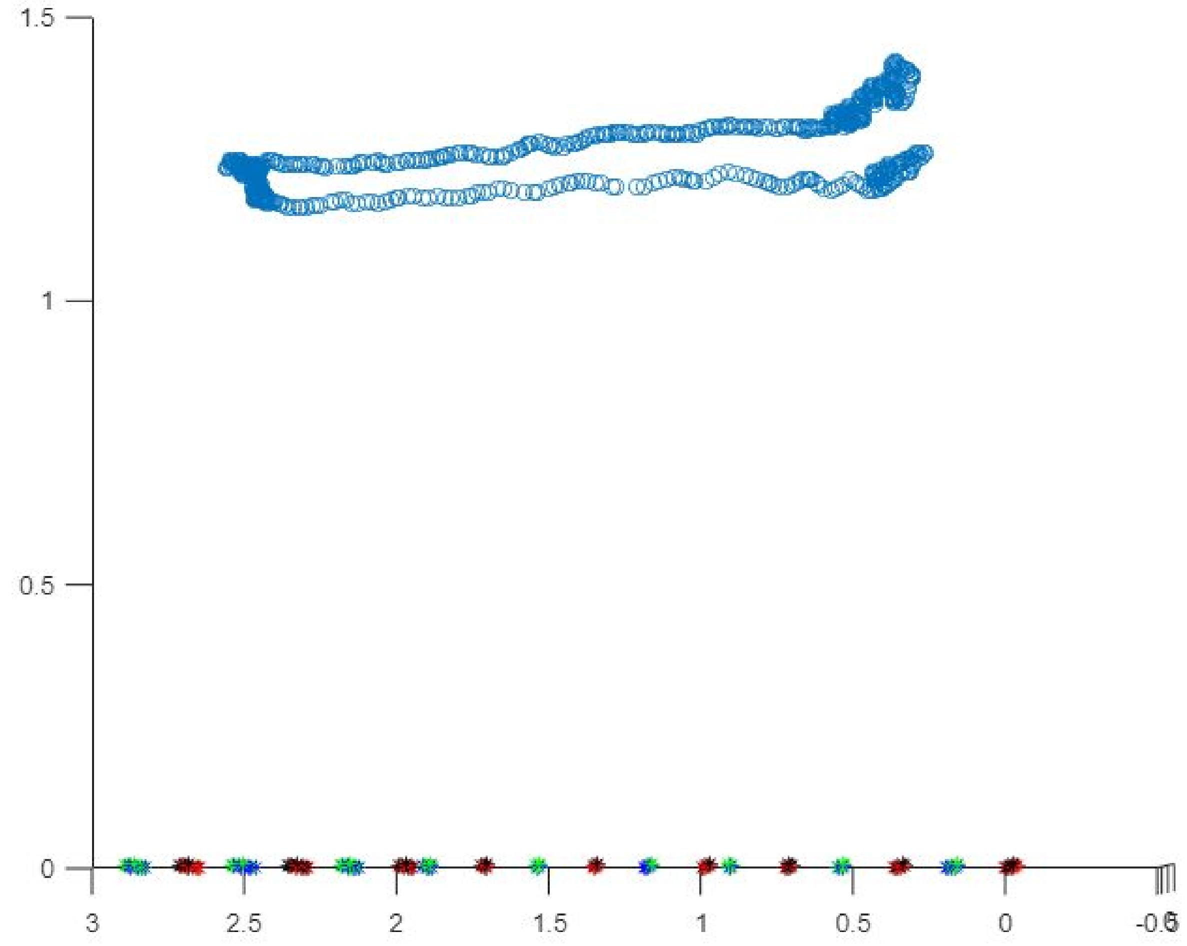
DataSquare.mat
(Top View)



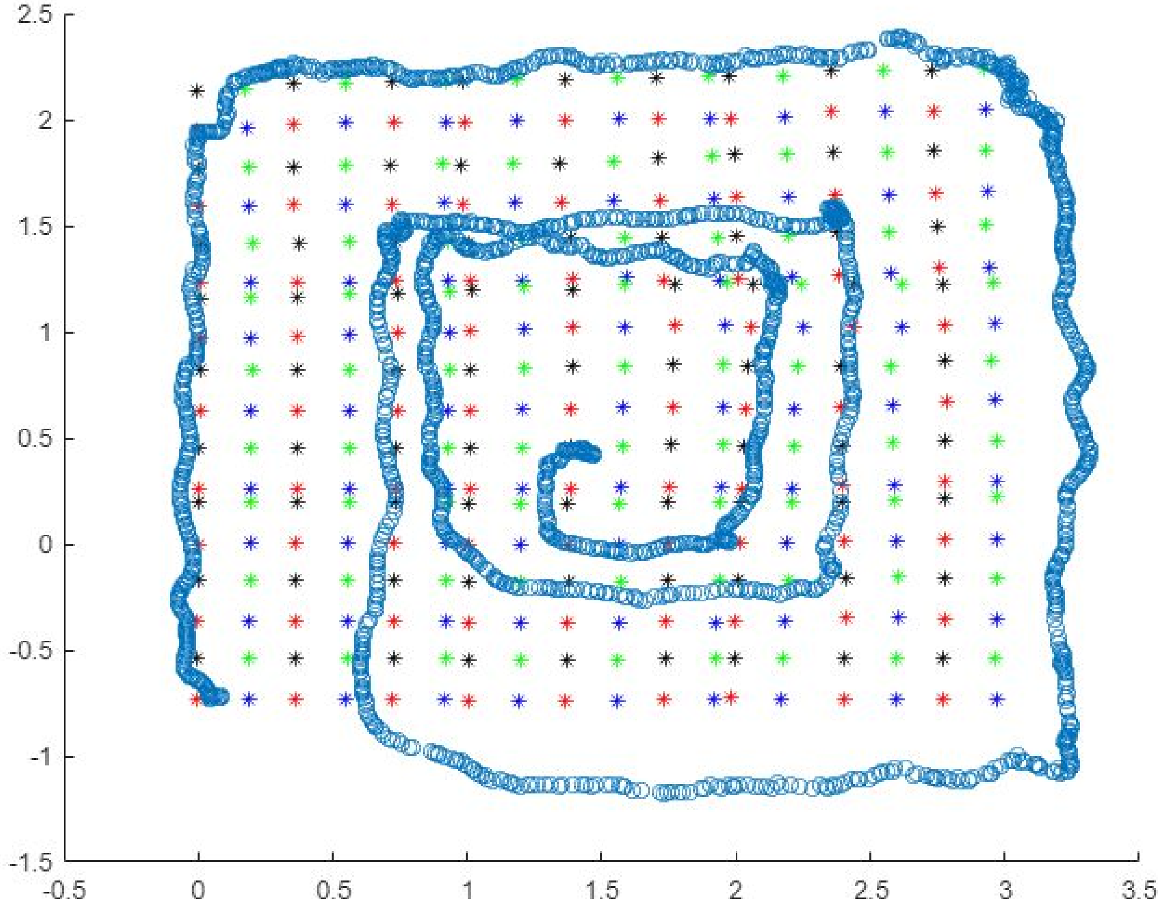
(Isometric View)



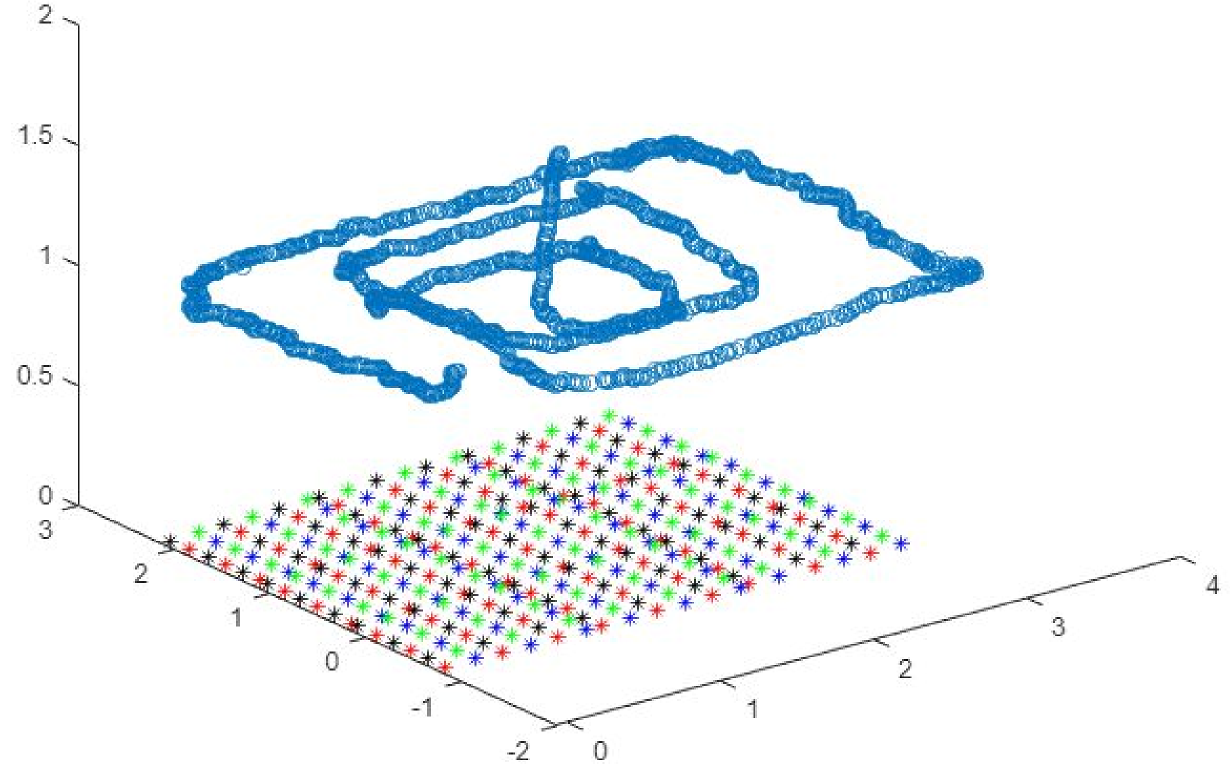
(Side View)



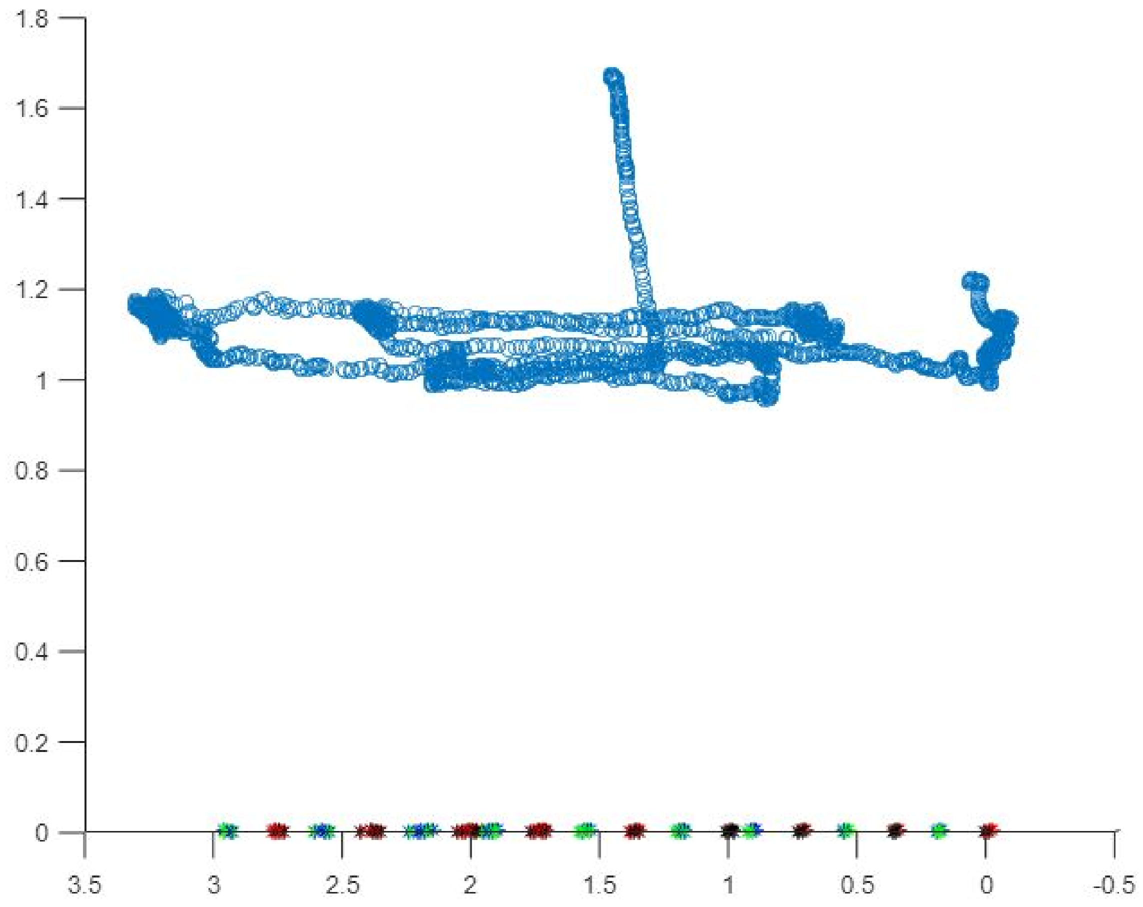
DataMapping.mat
(Top View)



(Isometric View)



(Side View)



1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
3.5
3
2.5
2
1.5
1
0.5
0
-0.5