Structure from Motion

CMSC 426 (fall'18)

Rachith Prakash (UID 116141468)

Introduction

The general problem of estimating the locations of 3D points from multiple images given only a spare set of correspondences between image features is called as Structure from Motion. This process often involves simultaneously estimating both 3D geometry (structure) and camera pose(motion).

This project aims at retrieving the camera pose and landmark position from a 2-D video sequence. The video sequence contains AprilTags[1], which are specialized markers that helps in landmark point detections. Using landmarks obtained from these markers, the pose of the camera and the position of the landmarks are estimated.

Assumptions

1. Since the landmarks were all placed on the floor, it was assumed that all the tags lie on the plane, z=0.
2. All position and poses mentioned are with respect to a world frame. The world origin is at point1 of tag 10 of the first frame, as mentioned in the project description.
3. At least 4 previously seen landmarks should be seen in next frame in order for this algortihm to work.

Flow

1. We start with an assumption or arbitrary assignment of origin in the world coordinate frame as tag 10's landmark1 (given in the project description). Since we know the distances and the angles of other landmarks of this tag w.r.t landmark1 of this tag, we can get their world coordinates. Using these 4 set of coordinate pairs, we compute the Homography matrix between camera pose1 and the world frame.
2. This Homography matrix is applied to other landmarks as seen from camera pose1.
3. Using the relation between the world coordinates and the image coordinates, we can get the rotation and translation matrices for camera pose 1. This would be w.r.t world frame whose origin is at landmark1 of tag10. The world frame axes are defined as shown in Fig2.
4. We now have the world coordinates of landmarks seen by camera pose1. Several of these will be common in camera pose2. So, these landmarks' (as seen from camera pose2) location is used to calculate the Homography matrix. This matrix is used to calculate the other landmark's world frame coordinates. Using these values rotation and translation of the camera pose2 is calculated.
5. Step 4 is repeated until the last camera pose.

6. Once these values are obtained, we have the landmark locations (world frame), camera poses w.r.t world frame. Thus, we can feed this data into GTSAM to optimize these values based on several constraints as shown in the below Fig1.

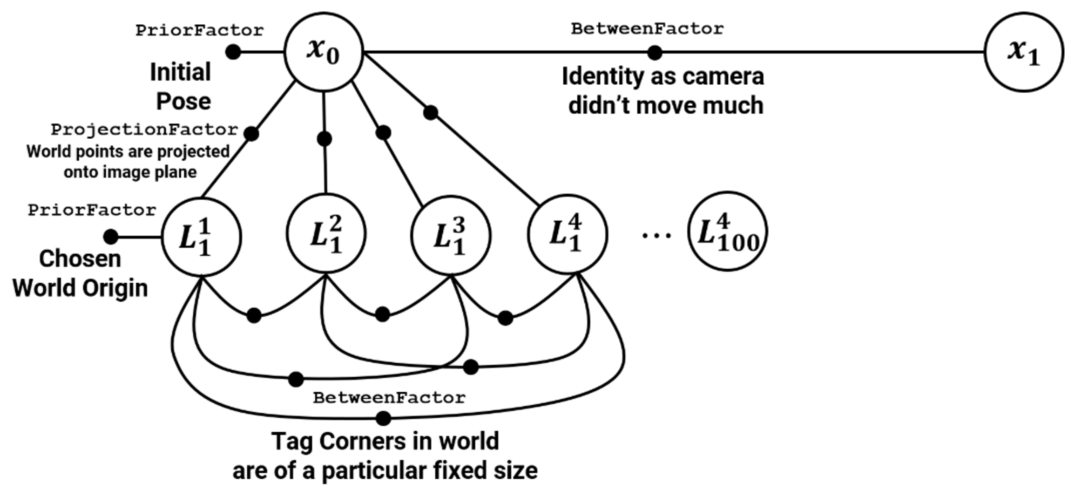

Figure 1

Pipeline

- Initial Camera Pose Estimation

Initial image contains multiple tags, whose corner points in image frame coordinates were obtained using the AprilTag Detection Library. The library also provides the TagID of each detected tag. All the tags have unique ID. The points in each tag were numbered as shown in Fig 2. This was used to identify each tag corner point uniquely.

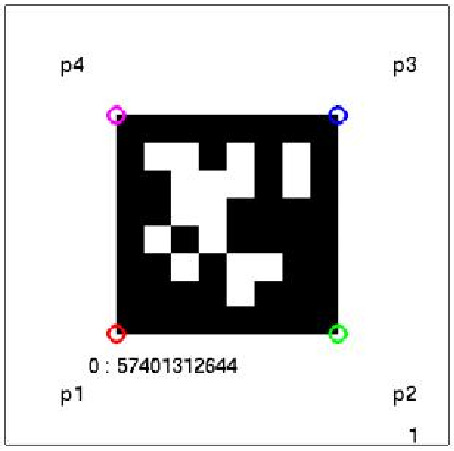

Figure 2

Since we are assuming the world frame to be at tag 10 point 1, we know that the corner points in world coordinates for Tag 10 are (0,0,0), (TagSize,0,0), (TagSize, TagSize,0) and (0, TagSize,0). The projection equation for 3D points on an image is as below,

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = K \begin{bmatrix} r_1 & r_2 & r_3 & T \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix}$$

But in our case, we have assumed that all the landmarks lie on the Z plane, hence the equation reduces to a Homography as shown below,

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = K \begin{bmatrix} r_1 & r_2 & T \end{bmatrix} \begin{bmatrix} X \\ Y \\ W \end{bmatrix}$$

In this equation, the u and v are given by the corner detection algorithm of the AprilTags. The calibration matrix was given in the problem statement. And X, Y for all points of tag 10 are known. Therefore, we can calculate the initial R, T.

- ## Initial Frame Landmark Position Estimation

Now that we have calculated the camera pose for the frame, we try to estimate the position of all other tags seen in this frame (other than tag 10). We do this using the same 2-D projection equation (Homography) mentioned as above. We pre-multiply both sides with the inverse of the K*RT and plug in the values of u and v (image coordinates are given), giving us the 3-D positions of the tags as the output (Z is always assumed to be 0). This is repeated for all the tags in the frame.

- ## Consecutive Camera Pose Estimation

Since we will see at least one tag that was seen before, whose world frame coordinates are known. Using all these positions and the corresponding image frame coordinates, we can calculate the world frame coordinates of the camera pose with respect to new frame. This is achieved using the same algorithm mentioned in the section, Initial camera pose estimation. In order to simplify the code development, this was done using the *estimateWorldCameraPose* function from MATLAB.

- ## GTSAM

All the calculations we made above aren't perfect. For example, pixels have a resolution limit, and hence, the projection can only be as good as the resolution projected onto the world coordinates. Also, the error in each frame calculations propagate since the future calculations are dependent on previously calculated values. As a result, we get different landmark and camera pose values of for the same entity.

GTSAM is an optimization toolbox designed specifically to perform simultaneous localization and mapping. In order to use the toolbox, we first must build a factor graph. A factor graph is simply a method of representing an optimization problem. Symbols (shown as white circles) in the factor graph represents the variables that must be optimized. Factors (shown as black dots) represent constraints on the different symbols. GTSAM provides functions which help in creating and running this factor graph.

Fig1 shows how to form the factor graph for this project. Each 'L' symbol represents one corner of a tag i.e. landmark, while each 'x' symbols represents the camera pose at that frame. The prior factors are the initial values for the camera pose (initial camera rotation and translation) and the tag 10's landmark 1 location is (0,0,0). The black dots between the landmarks(L) and the camera poses(x), represent how both these symbols are constrained to each other by the projection equation. GTSAM provides a function, *GenericProjectionFactorCal3_S2* which helps us to easily create this constraint. The black dots between the camera poses(x), represent the transformation between the camera frames, which is assumed to be very small and hence they were set as R=eye(3) and T=[0,0,0]. This constraint was added using the *BetweenFactorPose3* function from GTSAM. The black dots between the landmarks represent the knowledge that they all are part of one tag and hence we know their relative positions (size of tags is fixed). This constraint was added using the *BetweenFactorPoint3* function from GTSAM.

All the constraints we add are not exact, and hence we must add a noise value model to each of them. This was done using the *noiseModel.Isotropic.Sigma* and *noiseModel.diagonal.Sigma* function from GTSAM. Since we know that the relative positions between the points of the tag, those factors were set to have a low sigma(1e-5) noise model, while the other factors were set to have a higher value, around 0.1 and some to 0.5.

Although this model of the factor graph was suggested in the project, we felt the factor graph was over constrained from the second dataset(spiral) and hence the factors between the tag points were removed for the second dataset.

Results

Dataset 1 – Square trajectory of camera w.r.t landmarks.

For this dataset, Dogleg optimizer worked perfectly! As can be seen from the figures below, after applying GTSAM the results were better.

Before GTSAM

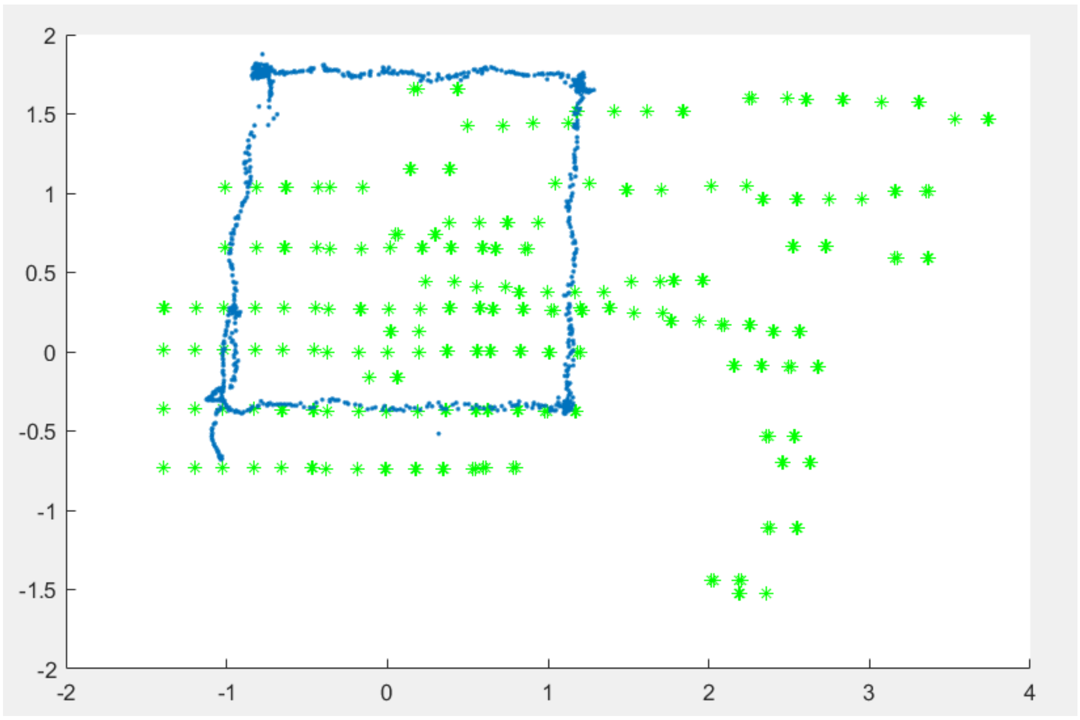

Figure 3: Without GTSAM top view

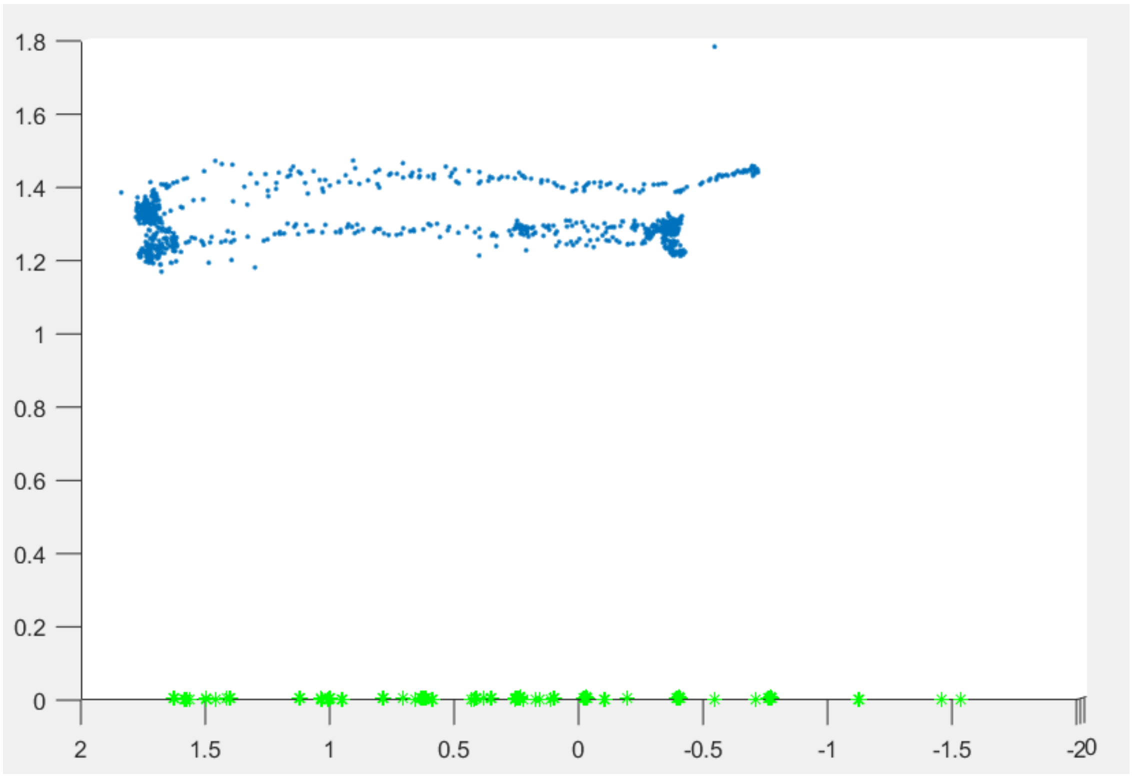

Figure 4: Without GTSAM side view

Fig 4.

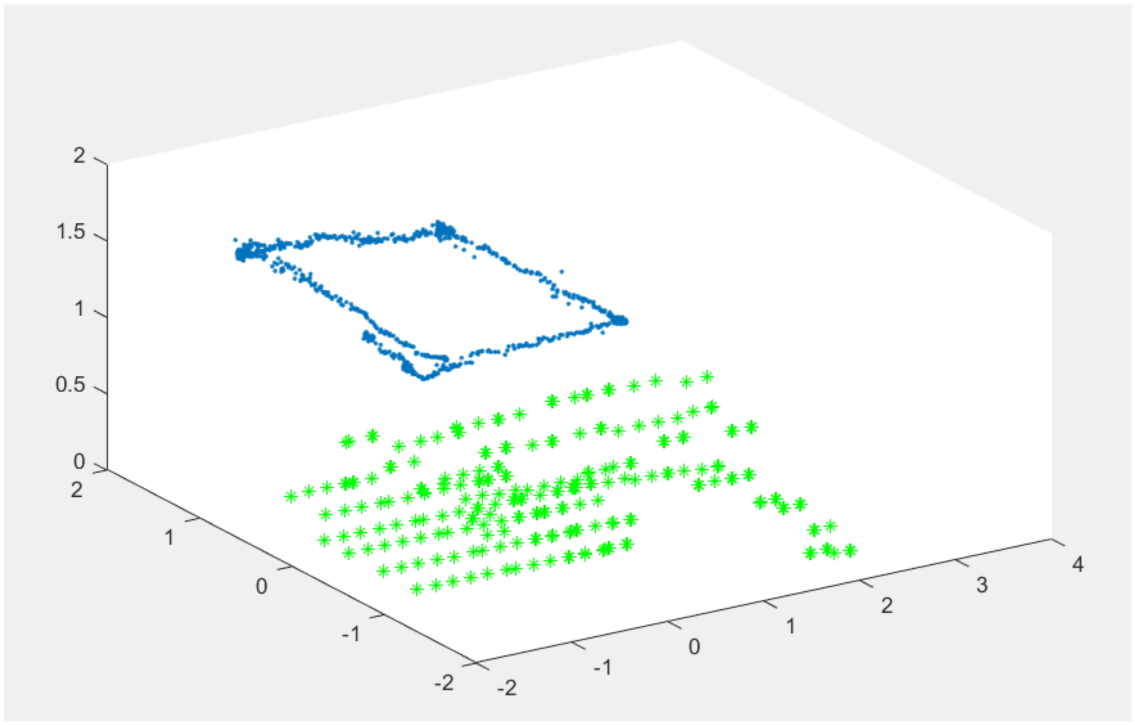

Figure 5: Without GTSAM isometric view

After GTSAM

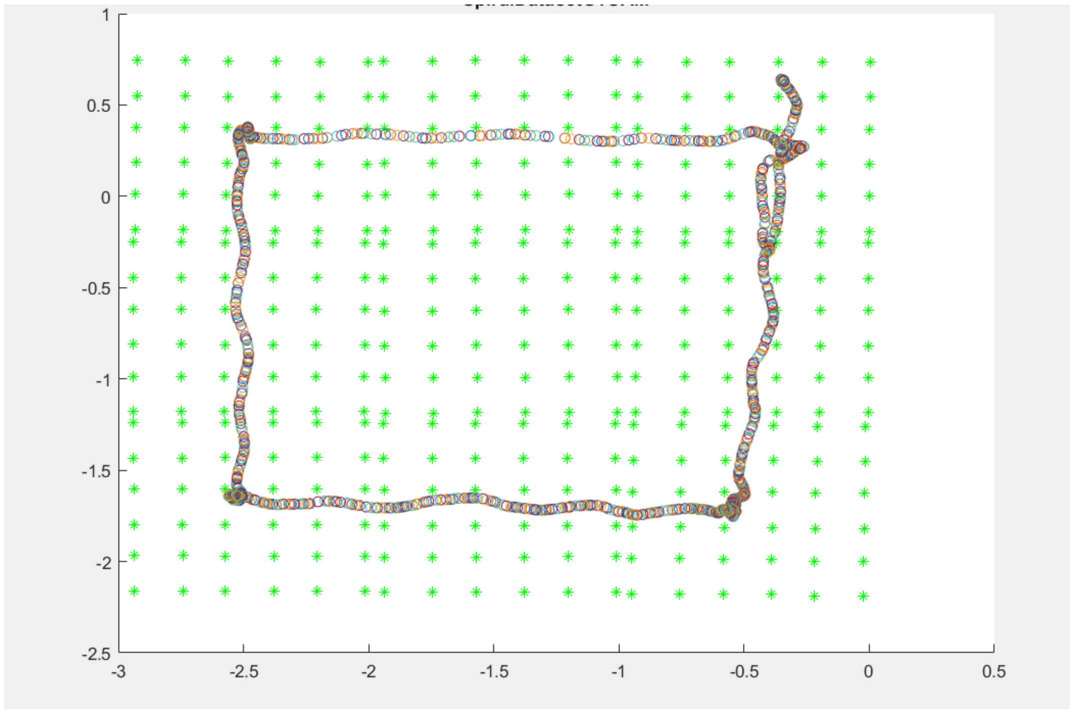

Figure 6: After GTSAM top view

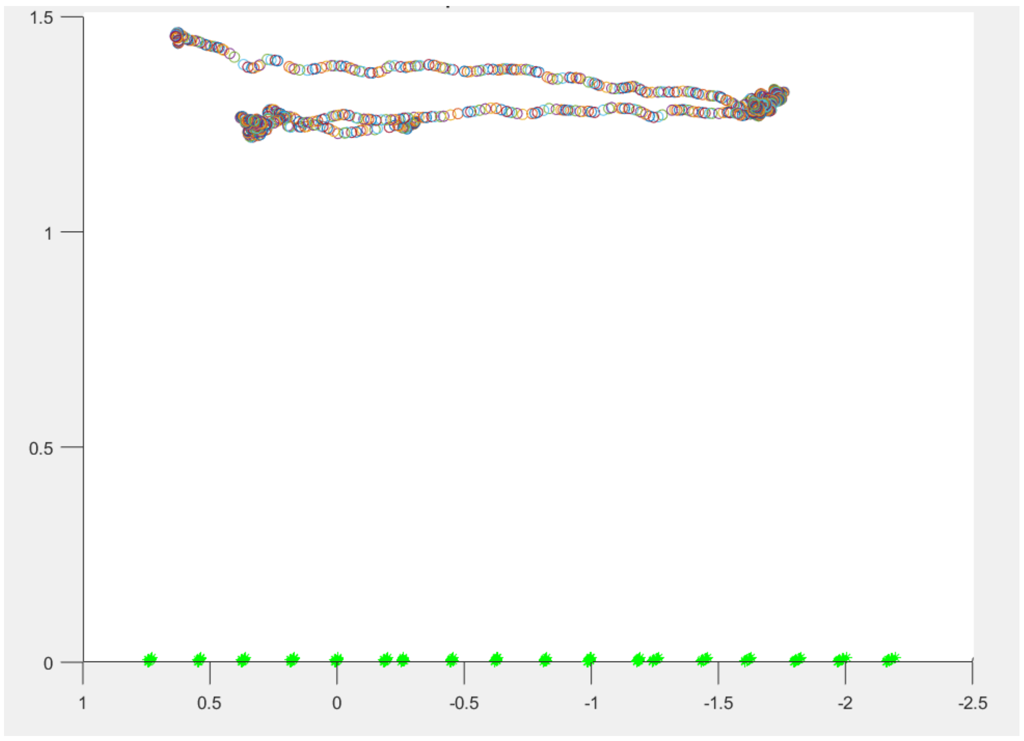

Figure 7: After GTSAM side view

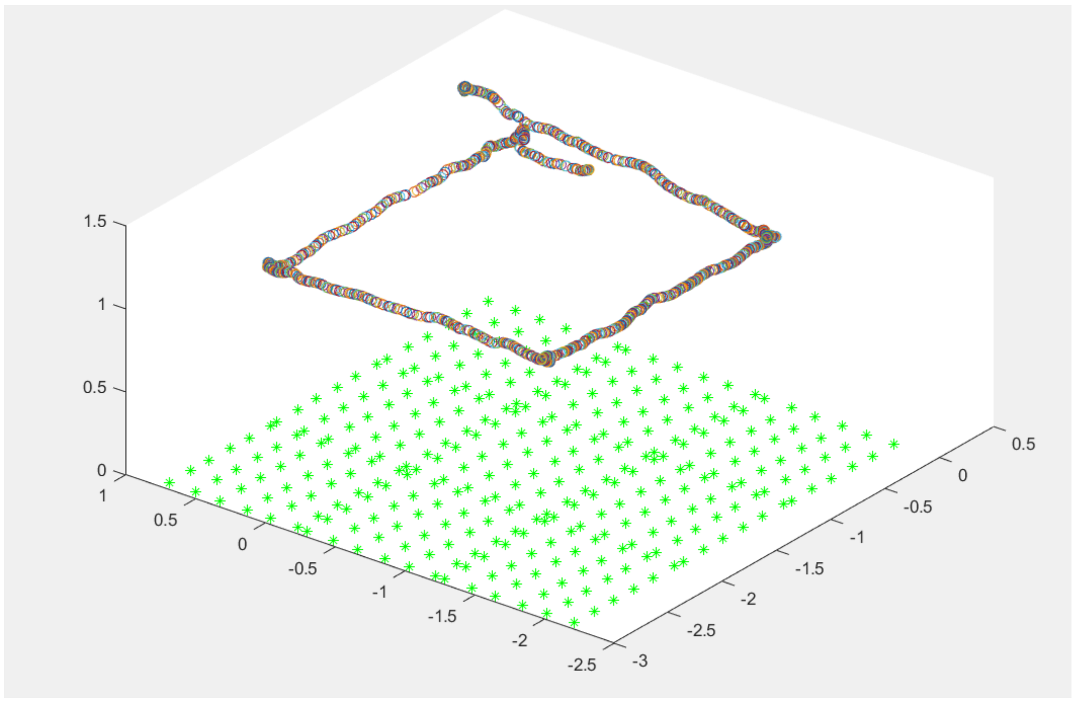

Figure 8: After GTSAM isometric view

Dataset 2 – Spiral movement of camera w.r.t landmarks

For this dataset, Dogleg optimizer was not working. Could not debug this. Optimizer was not throwing any error. It did not assign any pointer value to the result where result is obtained from Dogleg optimizer.

Hence, used *LevenbergMarquardt* optimizer. This worked, but the results, however, were not good. The results obtained before GTSAM were a bit better than after GTSAM. I need to work on how to optimize the graph. This also means I need to understand working of GTSAM more thoroughly.

Parameters used for *LevenbergMarquardt* were that of Lambda, a lambda value of 0.1 was giving almost the same result as 0.5. Used 0.1 for the figures shown below.

Before GTSAM

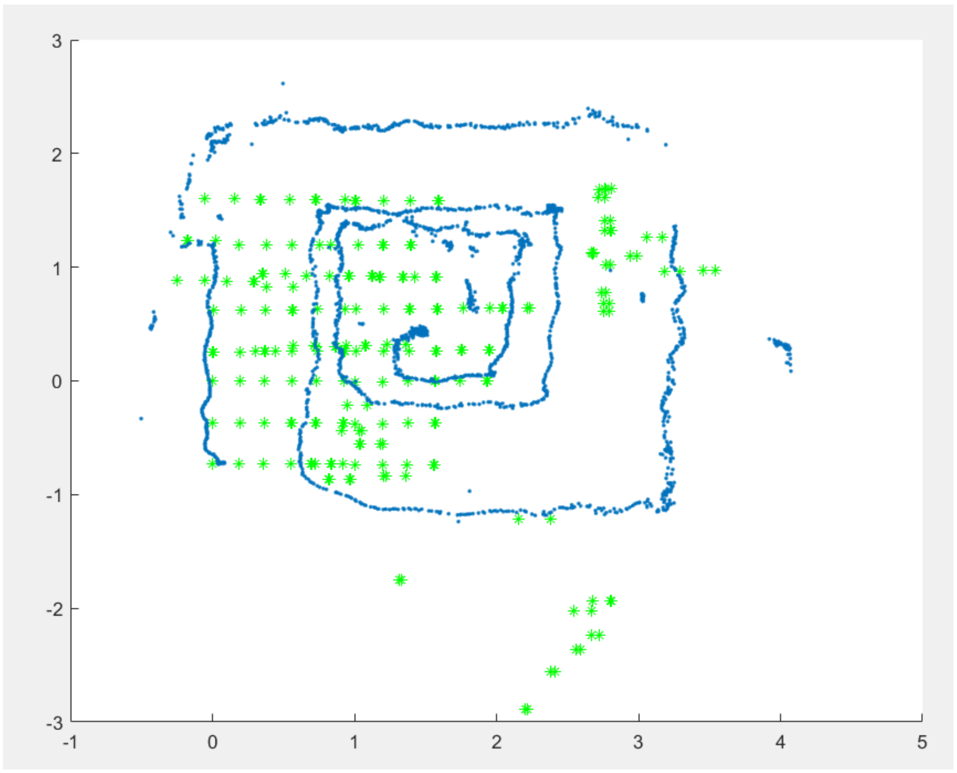

Figure 9: Before GTSAM top view

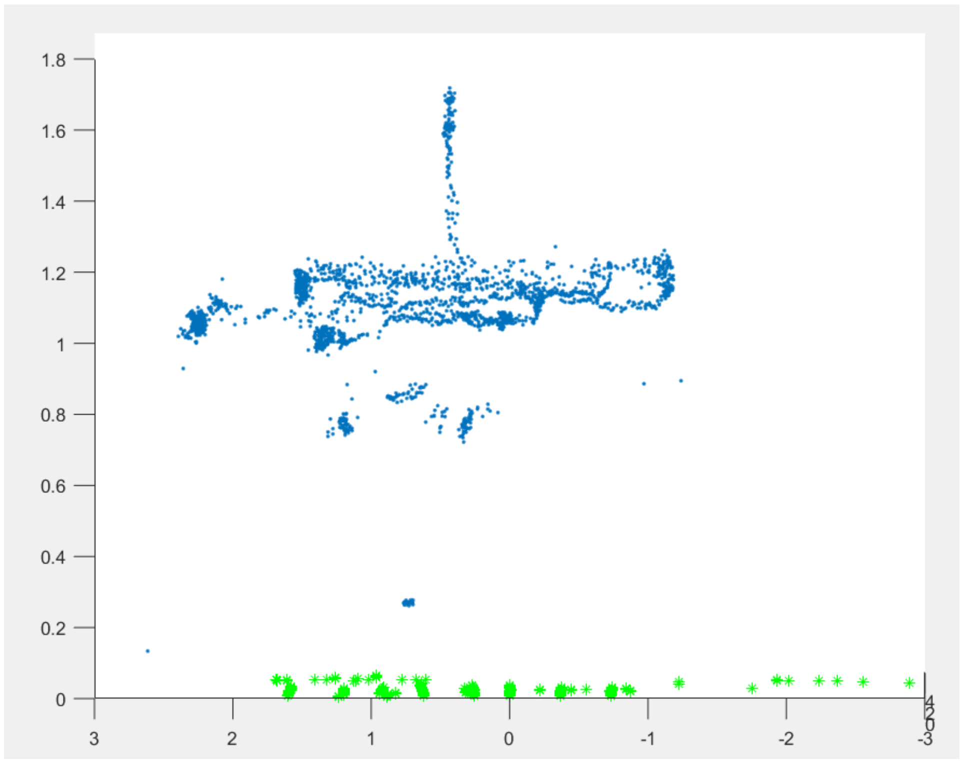

Figure 10: Before GTSAM side view

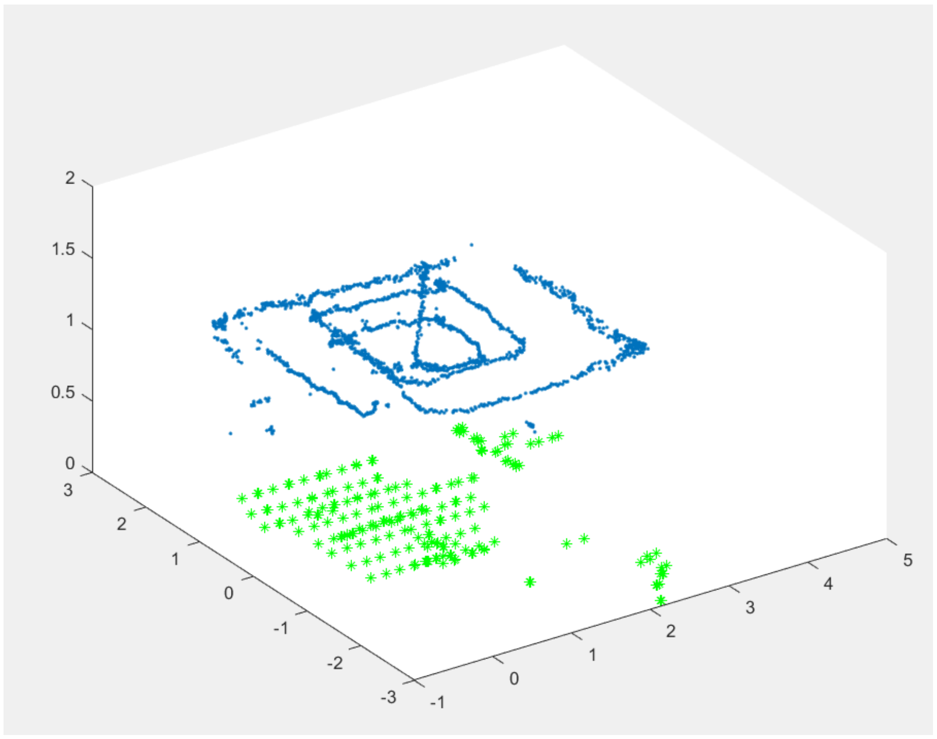

Figure 11: Before GTSAM isometric view

After GTSAM

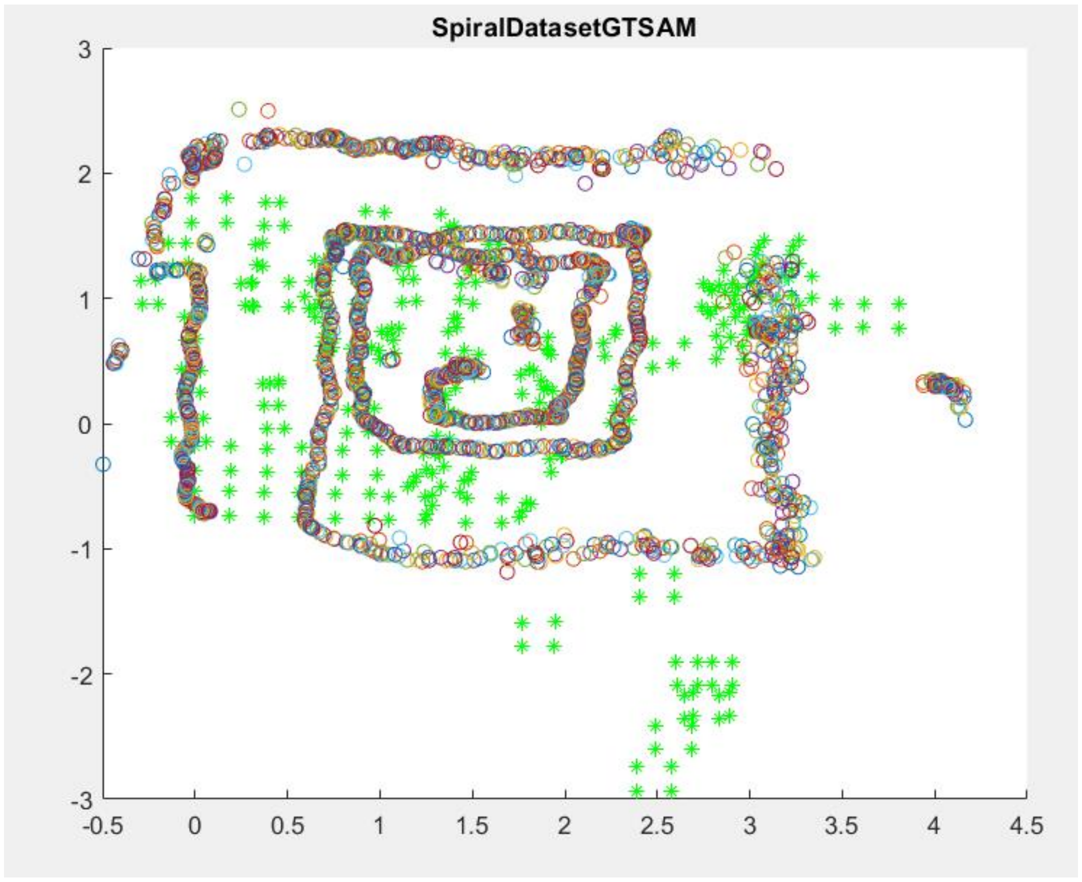

Figure 12: After GTSAM top view

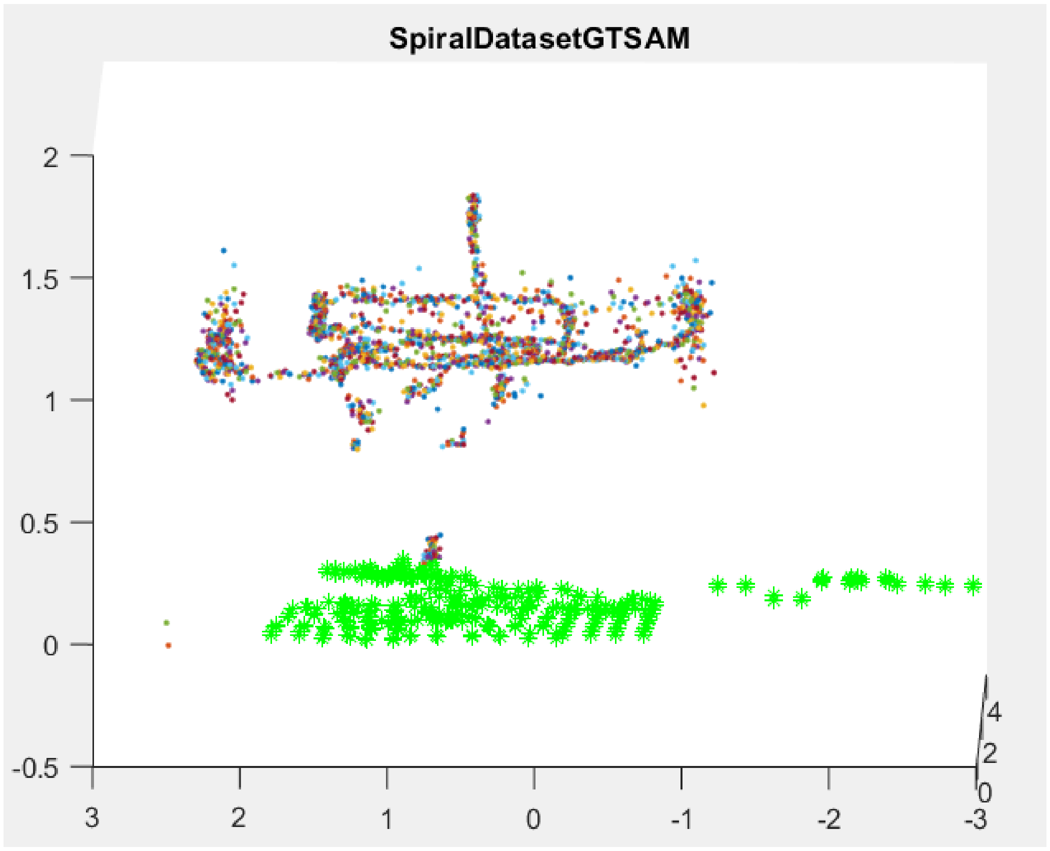

Figure 13: side view

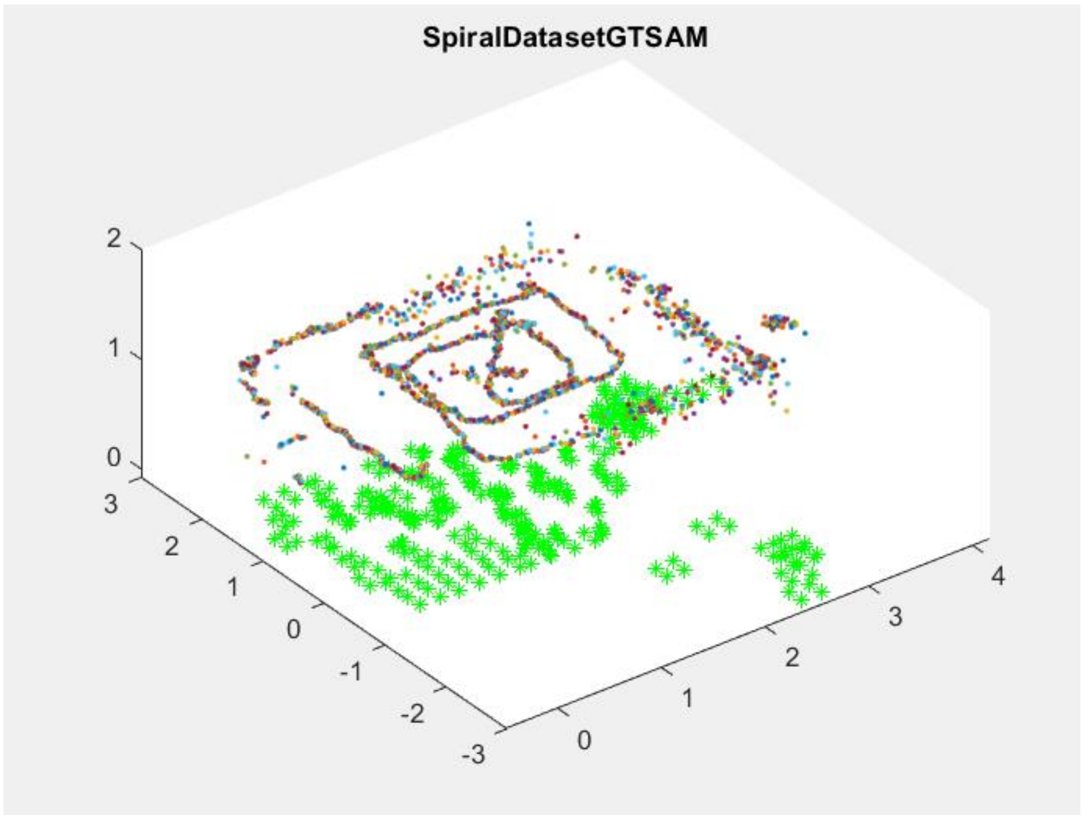

Figure 14: isometric view