Visual odometry

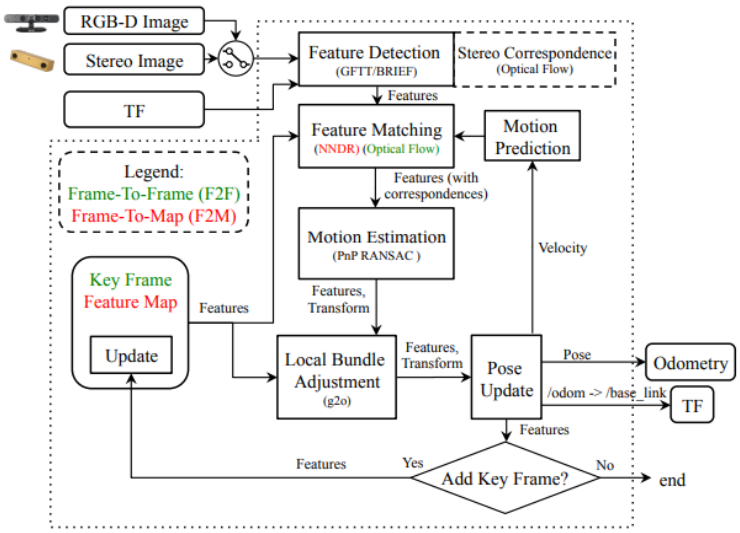
In visual odometry we use RTAB-Map based visual odometry. It mainly has 6 key steps:

1. *Feature Detection* : When a frame is captured, we need to detect the maximum no of features (which can be varied).
2. *Feature Matching* : From the above detected features we match the features and find corresponding image points in two images.
3. *Motion Estimation* : When correspondences are computed, the RANSAC is used to compute the transformation of the current frame.
4. *Local Bundle Adjustment* : The resulting transformation is refined using local bundle adjustment on features of all required key frames.
5. *Pose Update:* With the estimated transformation, the output odometry is then updated
6. *Key Frame and Feature Map update :* If the number of inliers computed during Motion Estimation is below the fixed threshold, the Key Frame or Feature Map is updated.

RTAB-Map implements two standard odometry approaches called *Frame-To-Map (F2M)* and *Frame-To-Frame (F2F)*. The main difference between these approaches is that F2F registers the new frame against the last keyframe, and F2M registers the new frame against a local map of features created from past keyframes.

The implementation we do is *Frame-To-Map* implementation, because in *F2F* the key frame is simply updated using the current frame and if there is error in the current computation then it will affect later steps too. But in *F2M* the Feature Map is updated by adding the unmatched features of the new frame and updating the position of matched features that were refined by the Local Bundle Adjustment module.

Feature Map has a fixed maximum of features kept temporary (consequently a maximum of Key-Frames). When the size of the Feature Map is over the fixed threshold, the oldest features not matched with the current frame are removed. If a key frame does not have features in Feature Map anymore, it is discarded. The update of the key frames



- For feature detection the paper suggests GFFT, but we can use ORB or other detectors because they are new and developed and perform better than GFFT (ref). For reference : ORB
- For feature mapping as mentioned in the paper we shall go with Nearest neighbour, can vary the k parameter and experiment the results.

As discussed in the meetings, the subtasks are

- *Map initialization* using the first two images we need to make a map. This involves feature detecting, matching, and pose estimation.
- *Tracking* - When a new image is given we should be able to estimate the pose of the image using the map generated above. This involves feature detection, matching, and pose estimation.
- *Map update* - Now from the new image data we need to find which points are not seen before add to the map, update the descriptors of the existing points. Also bundle adjustment to be done to add the further change.
- *Full scale tracking* - From the image and map built, we need to get the pose.
- *Results* - We shall compare output with the odometry data (ground truth ad decide the performance)

The implementation was done using F2F, now needs to be updated to F2M. As a dataset, RGBD images will be provided along with Odometry reading for turtle bot. The implementation was done by taking the points on the ground for reference because we know the *height* at which the camera is placed. It needs to be modified by adding the pixels of vertical obstacles too.

Challenges:

- Implementation of Bundle adjustment
- Detecting the new features and updating them in the map.

References and links:

- **2015-ULaval.pptx (usherbrooke.ca)** --
- **Labbe18JFR_preprint.pdf (usherbrooke.ca)** -- RTab Map paper
- **uoip/monoVO-python: A simple monocular visual odometry project in Python (github.com)**
- **[1502.00956] ORB-SLAM: a Versatile and Accurate Monocular SLAM System (arxiv.org)**
- **gaoxiang12/g2o_ba_example: An easy example of doing bundle adjustment within two images using g2o. (github.com)**
- **SLAM Implementation: Bundle Adjustment with g2o (fzheng.me)**

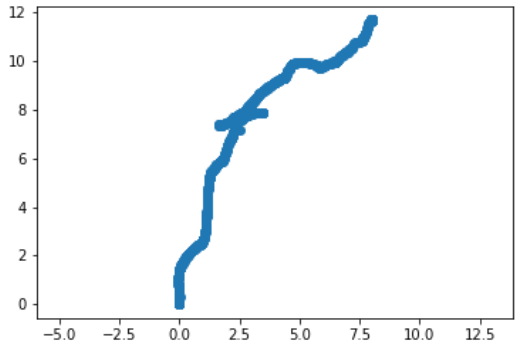
---

mask produced is (224,640)
Image is (1218,3544)

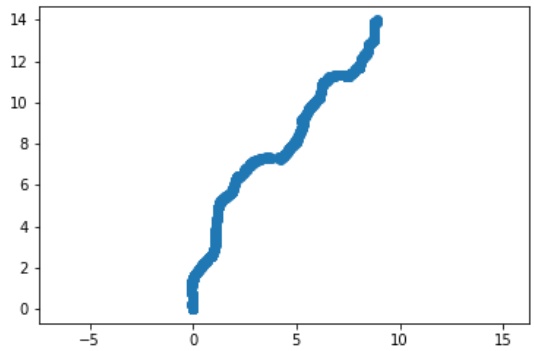
While running *orb.detectAndCompute()* its detecting features randomly i.e. its not detecting features all time when I run it. -- Import mask correctly

What values to take for focal length, cx, cy ?

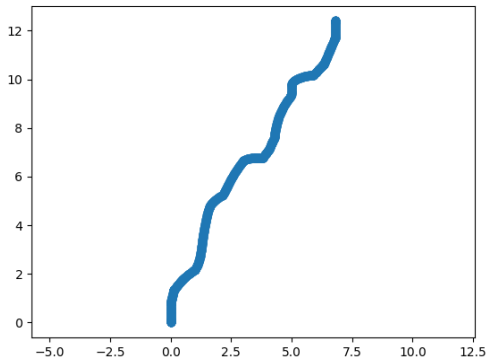
```
cy = 670
height = 85.5
imWidth = 3544
imHeight  = 1218
cx = imWidth / 12
f = cx * 3**0.5

orb = cv2.ORB_create(nfeatures = 200000)
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
```



With Calche



Without Calche



Ground truth

f2f is tested with and without calhe.

CLAHE is a variant of *Adaptive histogram equalization (AHE)* which takes care of over-amplification of the contrast. CLAHE operates on small regions in the image, called

tiles, rather than the entire image. The neighboring tiles are then combined using bilinear interpolation to remove the artificial boundaries

The outputs are as shown above. The deviation in the above images should be resolved when done with f2m, because the error in one frame won't get carried.

Implementation for f2m has started.

- Few Individual blocks for tracking are implemented and the basic structure is ready.
- BA needs to be implemented.
- Should I consider 2D transformation or 3D and same for bundle adjustment

---

Initial transformation is calculated using vertical obstacles. Only the points whose depth is under 3.5 m are used in finding the transformation.

The initial transformation is calculated using the matching 3d points whose Y coordinates are close enough to each other, because there is no translation in y.

The initial estimate is better than the one calculated using floor map alone.

- BA is took from the python scipy inbuilt functions
- Errors are not able to reduce much
- Values of errors are close to initial values

Map Initialisation:
Initial Map is generated by 2 sets of images. The results shown were the output on the images 100 and 110th images in the indoor dataset given.

Image 100

Image 110

- A mask is applied to the image to only take the points under the depth range 3.5m.

- ORB feature detector is applied to find the key features in the image. Feature matching was done to find the correspondences.

- Now a set of 3D points are generated for the key points using the available depth map. As there is no movement along Y-axis the generated 3D points should have close y-coordinates but however due to noise in depth reading 3D points are not generated as expected. There is significant deviation in the Y direction for a few points.

- So to get a better initial estimate a 2D transformation between the set of points is applied considering the points whose y deviation is less than 10 cms (can be varied).

- Now the generated initial transformation, set of 2D and 3D points are sent to the bundle adjustment module to refine our existing transformation.

- The bundle adjustment module used for implementation is taken from the scipy implementation. Here

- The implementation was done for normal perspective cameras but the module has to be changed to PAL camera by making modification to how the reprojection in normal camera is different from PAL camera. This part of the project took a lot of time, update of the reprojection, checking the data association etc.

Challenges:

The main challenges in this part are to get the better initial transformation and Bundle adjustment part. Having a good initial estimate is necessary to make our code work better. Initially, a 3D transformation was calculated but due to noise there was deviation in Y in translation and rotation. So the transformation is not good to send it as an estimate. So I tried reducing ransac error, taking only good features, etc, but the results weren't satisfactory so 2D transformation is calculated for points whose Y co coordinates are nearly close. In the Bundle adjustment I took the camera as a normal camera and sent it to the bundle adjustment module which gave me poor results. Later the sub module of reprojection and the correspondences are updated to fit for the PAL camera.

Results:

Error comparison:
Initial error : 117.39661062563486
Final error : 0.9859466282930333

Initial 2D transform is :
[[9.98110366e-01 -8.05042166e-02 -9.49372889e-04]
 [8.05042166e-02 9.98110366e-01 -3.42397766e-02]]

After BA:
[[9.96843992e-01 -3.86940703e-04 -7.93845389e-02]
[3.88316008e-04 9.99999925e-01 1.88709613e-06]
[7.93845322e-02 -3.27074277e-05 9.96844068e-01]]

[-4.81066230e-04 6.91251202e-07 -4.48546964e-02]

Translation from odometry: [-0.01309162, 0, - 0.060730834]
And Rotation of 0.07 radians

2D reprojection of images:

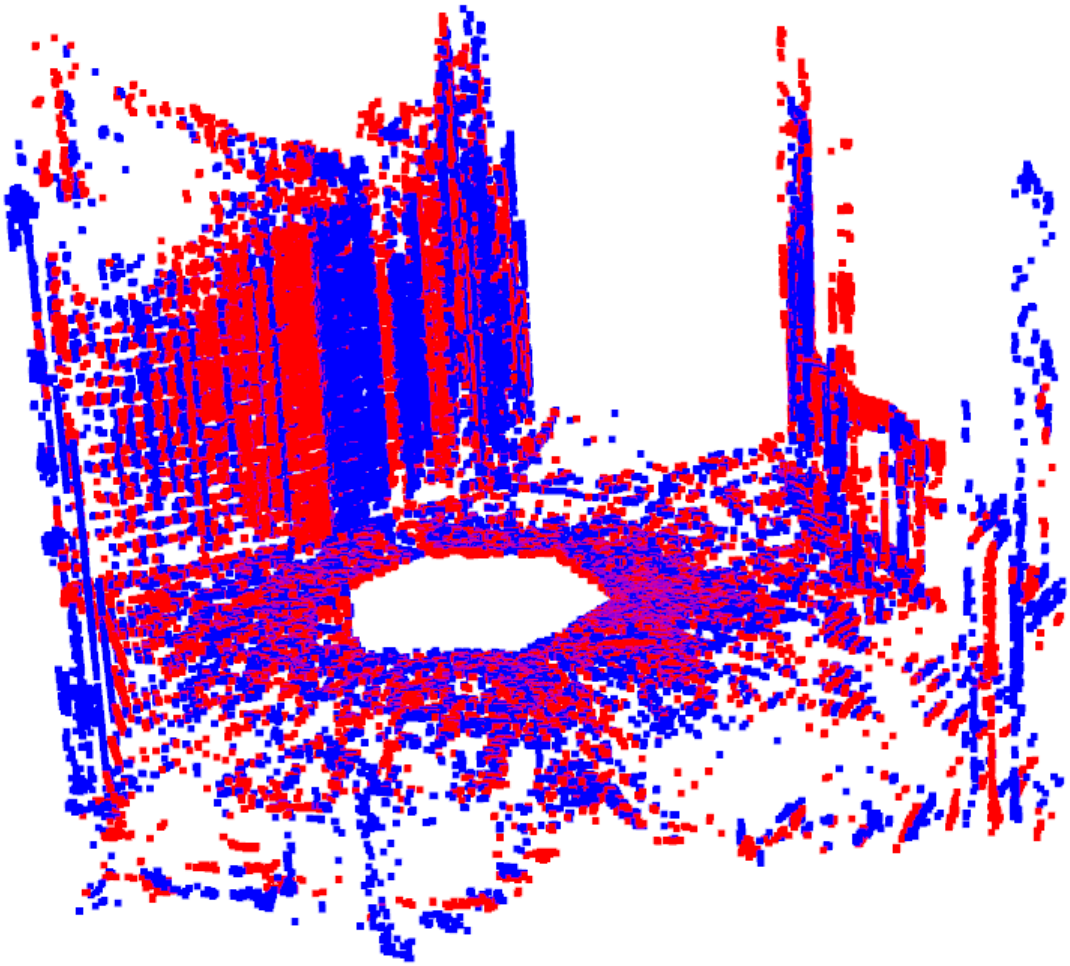
Original

Before BA

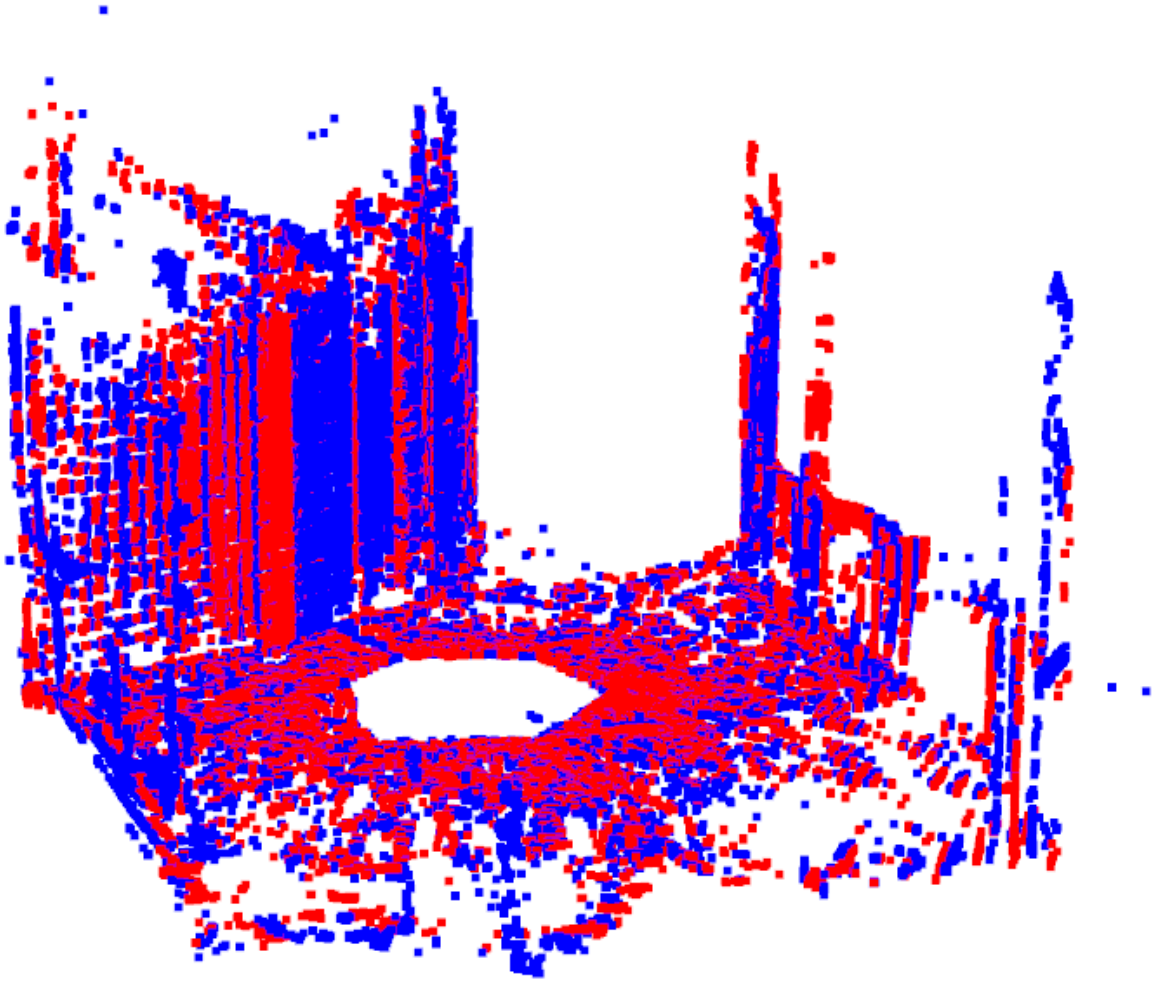
After BA

Point cloud of Map:

Before BA

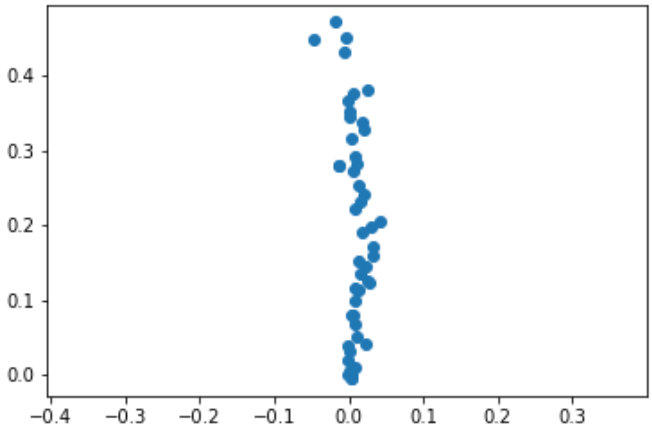


After BA:

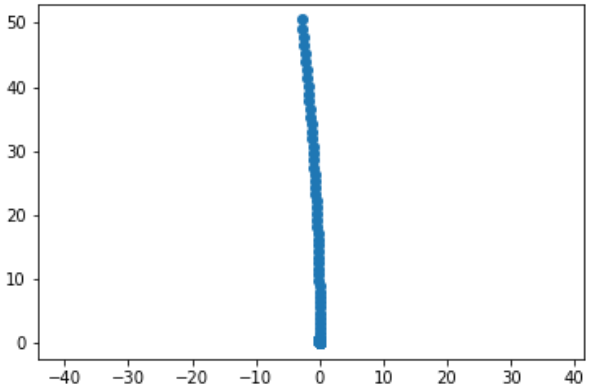


Tracking:

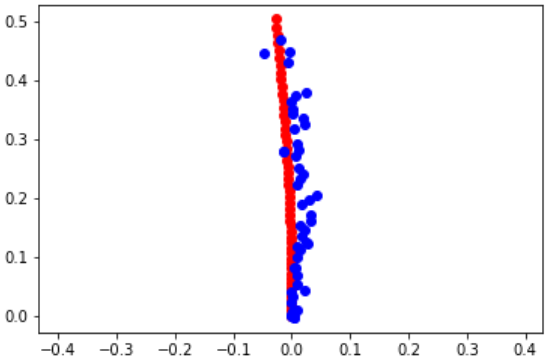
Now using the initialised map the images are tracked



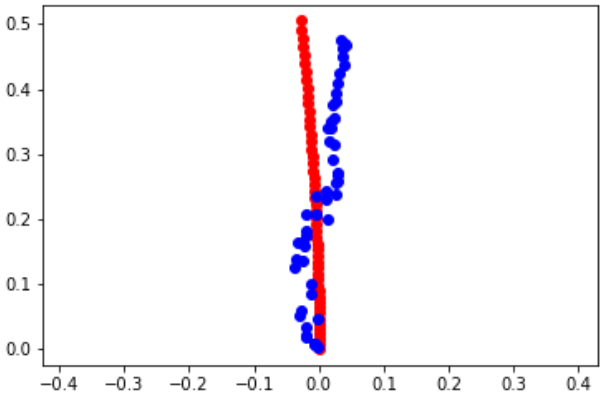
Tracking output



Ground truth



F2m vs ground truth



F2f vs ground truth

The implementation is working better in the initial cases because the map is generated using the 100 and 110th images. As the number increases there is slight deviation in the map this is because the features are reducing as the number is increasing.

Now the map is not updating so we are not having better results as we move forward. The next stage involves the map update where the key frames are updated so better results are expected. The rms error is around 3.569 cms.

Summary:

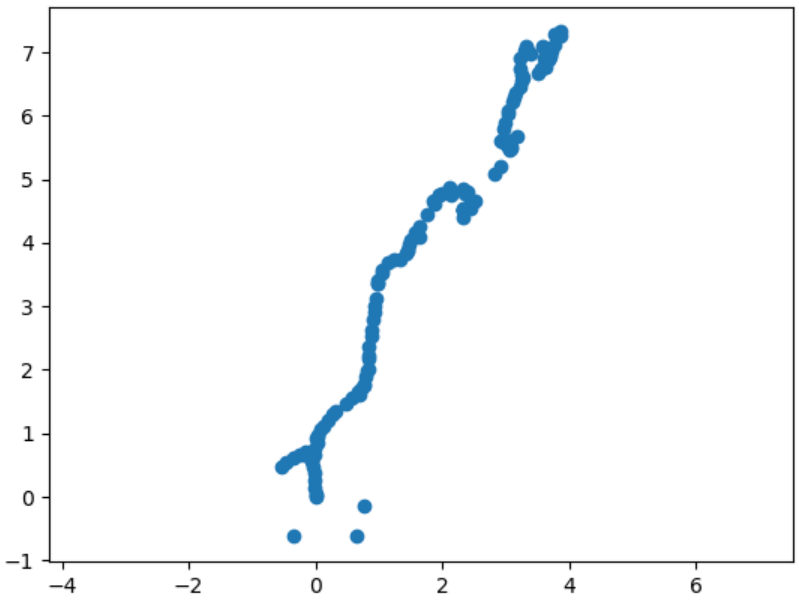
I have initialized the map using images 100 and 110...In the map initialization, a primary transformation is found using 3D point correspondences and later the transformation and the world points are refined using Bundle adjustment. Now using this initialized map I have tracked the pose for images 100 to 150...the pose is calculated by finding the common features between map points and image points unlike in f2f implementation this transformation is found using the previous frame... So now a comparison between these two are shown in the doc.In the tracking the pose is close for the images which are close to 100 and 110 because later the no of common features reduce. So, in the later stage I need to update the map with the new points from key frames.

Map update and Tracking:

- Performance

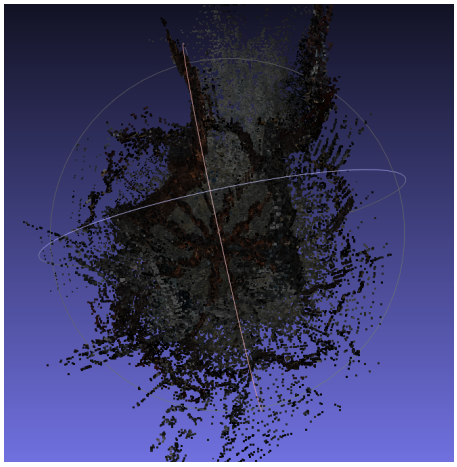
Time taking for computation of individual component:
Ran on Intel i5-10 8 GB RAM processor. No GPUS used.

image0 and image 10 are processed
time taken for reading image: 0.5584614276885986
time taken for getting features: 0.13435149192810059
time taken for reading image: 0.5445461273193359
time taken for getting features: 0.05550575256347656
time taken for feature matching: 0.08514976501464844
time taken for tracking - Initial estimate: 0.012339591979980469
time taken for adding new points: 1.0851740837097168
Load a ply point cloud, print it, and render it
time taken for whole process: 2.4979419708251953

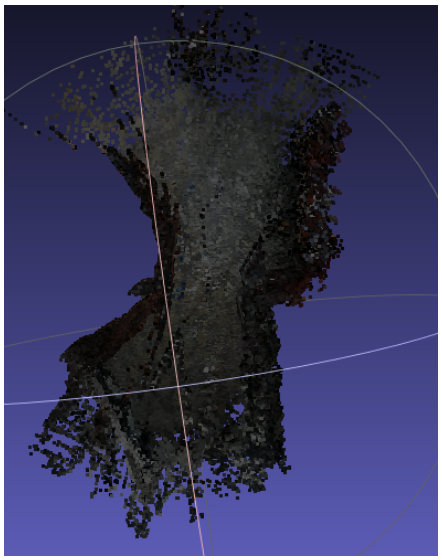


Quality:
The map generated is tested by tuning Lowe's ratio, no of features, clahe, sampling, resolution of the image, frequency for update, etc.

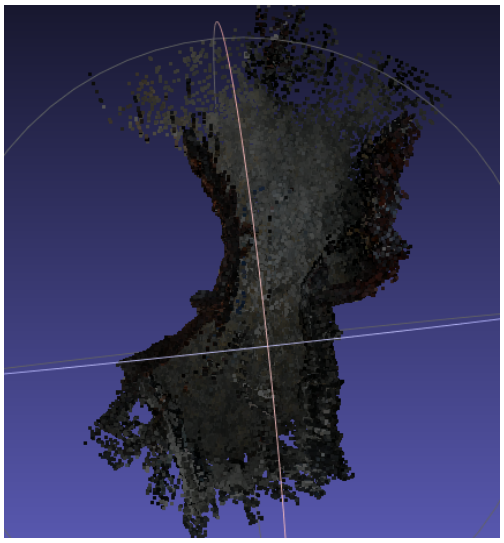
- When the Lowe's ratio is large there are more common features, because of loose matching there is mismatch in the pose



Lowe’s ratio : 0.9

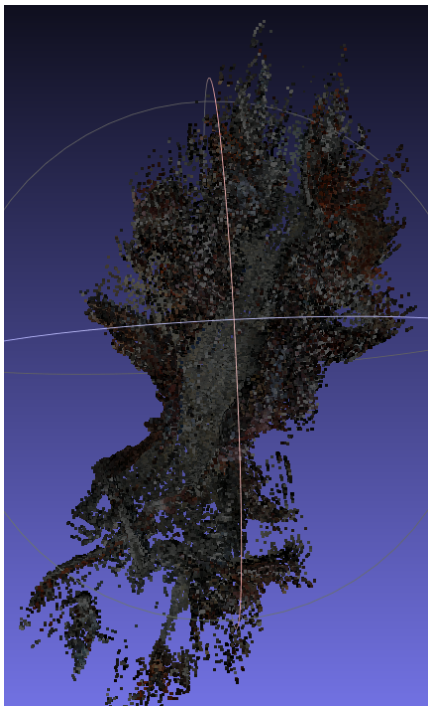


Lowe’s ratio : 0.6

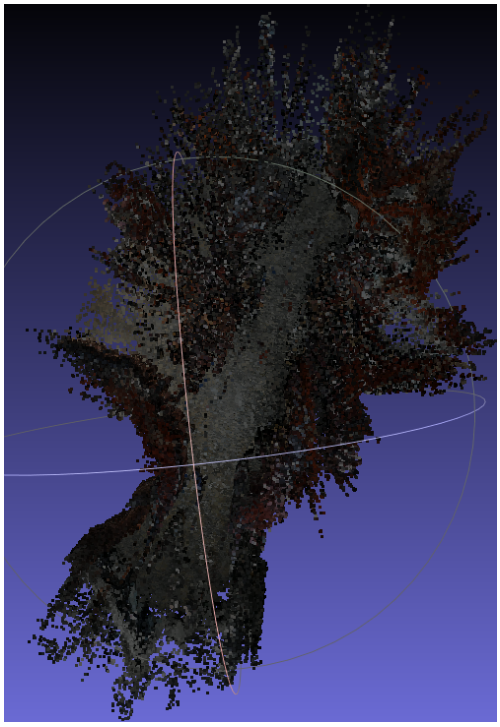


Lowe’s ratio : 0.7

- The feature number are to to 100000 features
- When tested without clahe the output is not good and much more noisy

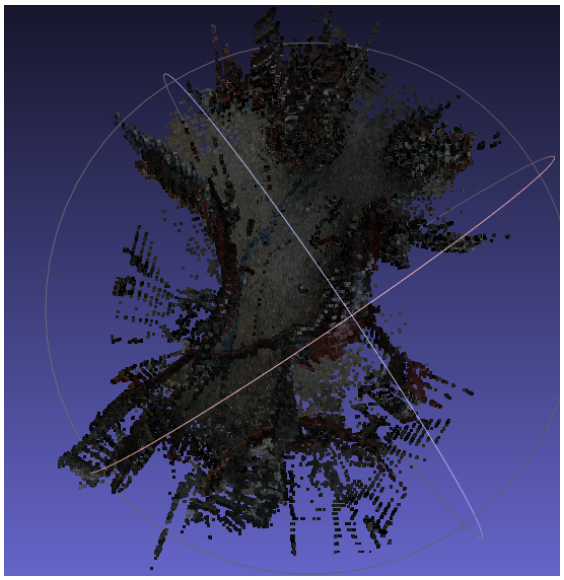


Without Clahe



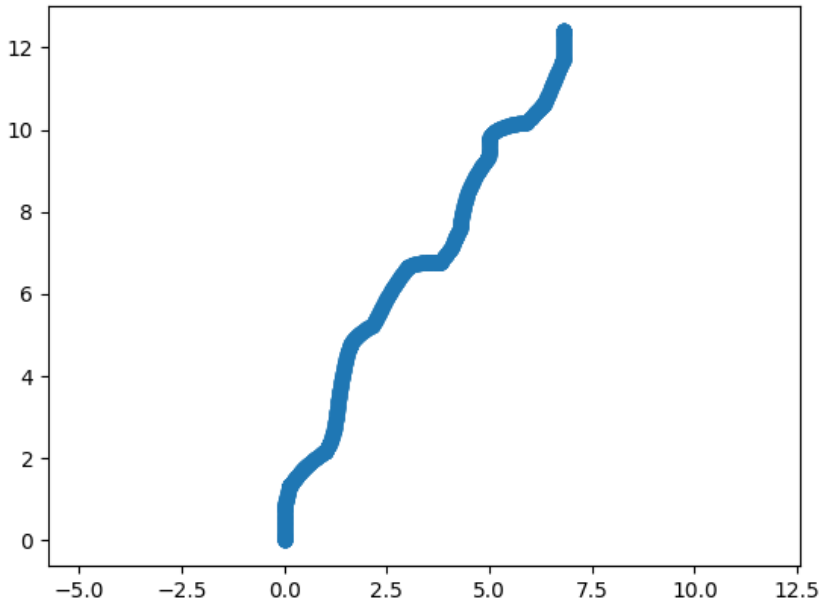
With Clahe

- The most affected parameter is resolution of image which drastically increased the performance and quality of the map.

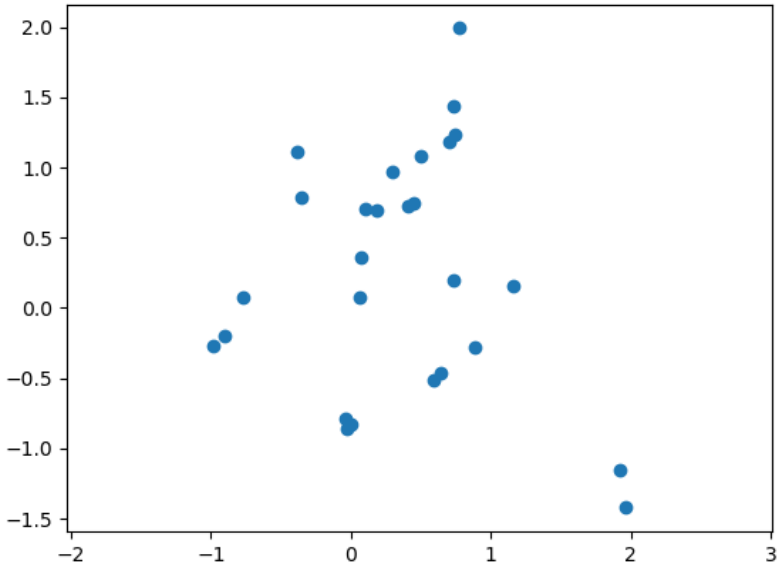


With resolution = 1 early clustering

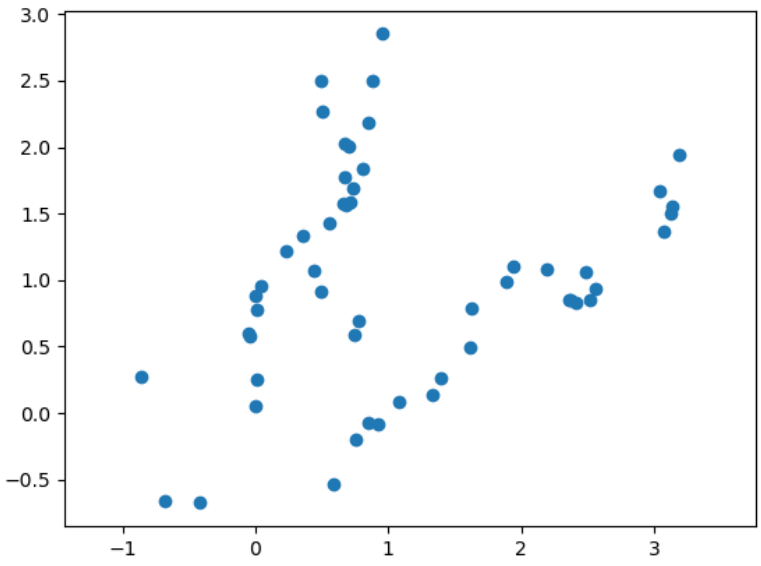
- The frequency of update is set to 10 which guarantees better performance and quality. The higher the sampling more the mismatch. Even the variation with 20,25 failed to show better results



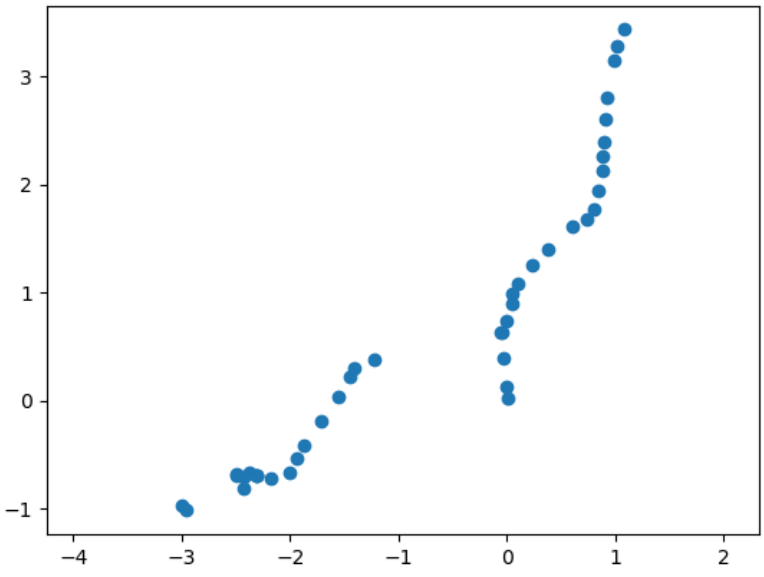
Ground Truth



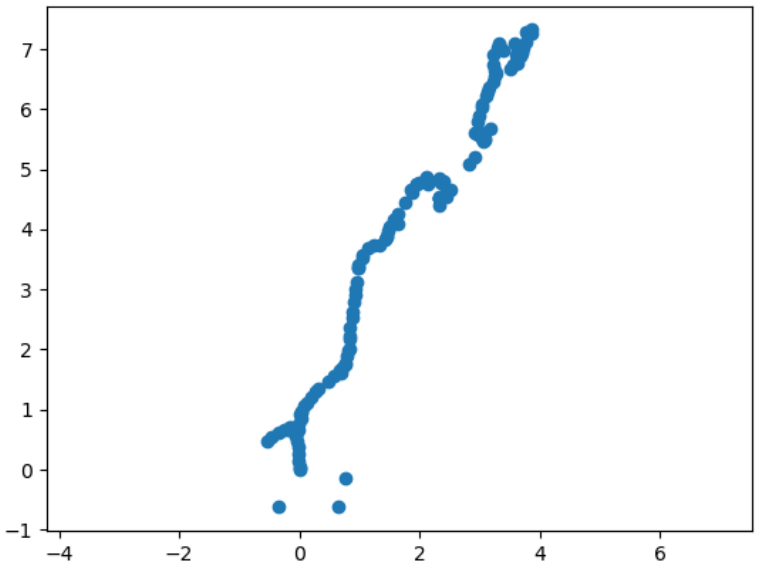
Sampling = 50



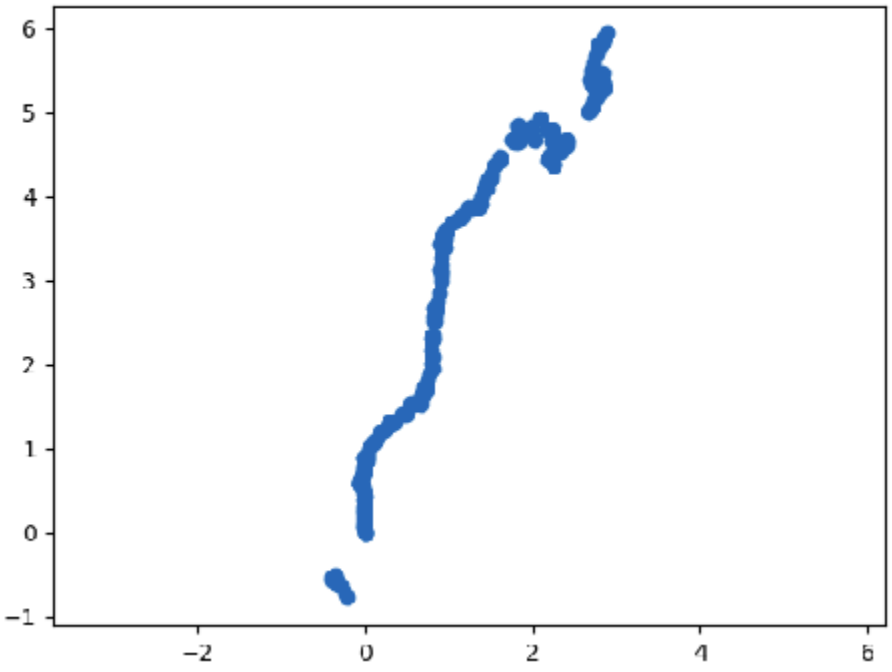
Sampling = 25



Sampling = 20



Sampling = 10



Sampling = 5

Parameters variation:

Firstly I varied Lowe's ratio from 0.6 to 0.9, with a gap of 0.1...then there was an error in the computation of initial transformation at some stage. For 0.9 its in between 240-250 while the value is reduced the range varied from 340-370.

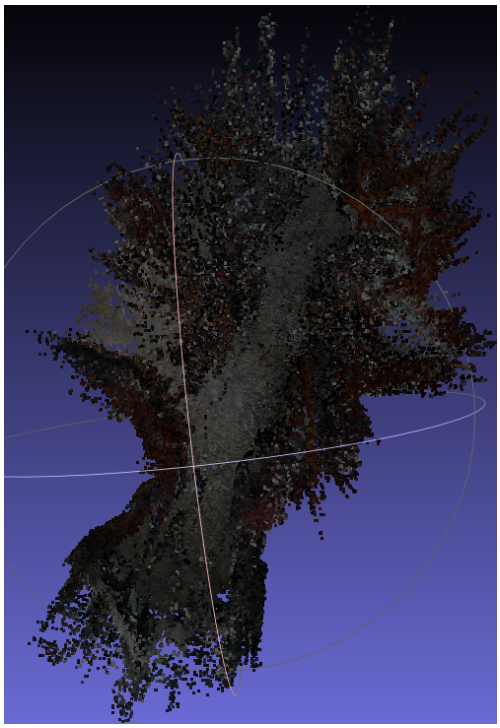
Later the resolution of the image is changed to 0.5 times and 0.25 times.

The 0.25 resolution change with the Lowe's ratio of 0.7 with appling clahe for 100000 features looks working fine till 1150 images later it seems deviated but the overall map looks fine.

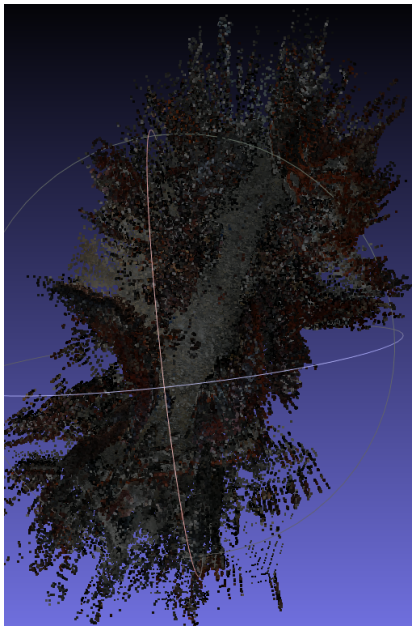
But when the same is repeated with 0.9 as Lowe's ratio then the clustering happened near 500 itself.

One more variation of 0.25 with 0.9 ratio is done for lesser features an the results look close to 0.75 with 100000 features.

The one with 0.5 resolution also shown good results, but sometimes failing in between 500-700 and computationally larger than 0.25 resolution



Map after 1250



Map after 1370

No parameters are giving significant difference…..the main difference in the output is observed when resolution and the matcher are changed. So few more experiments with matcher may lead better results.

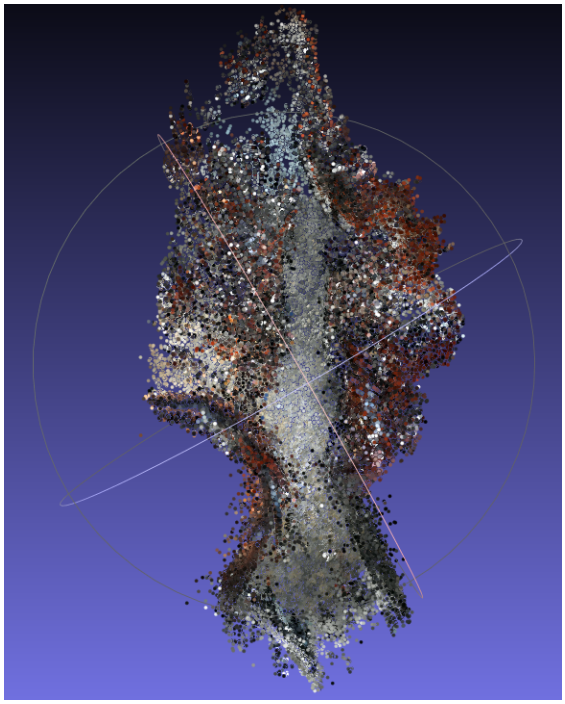
Tunable parameters in matcher:

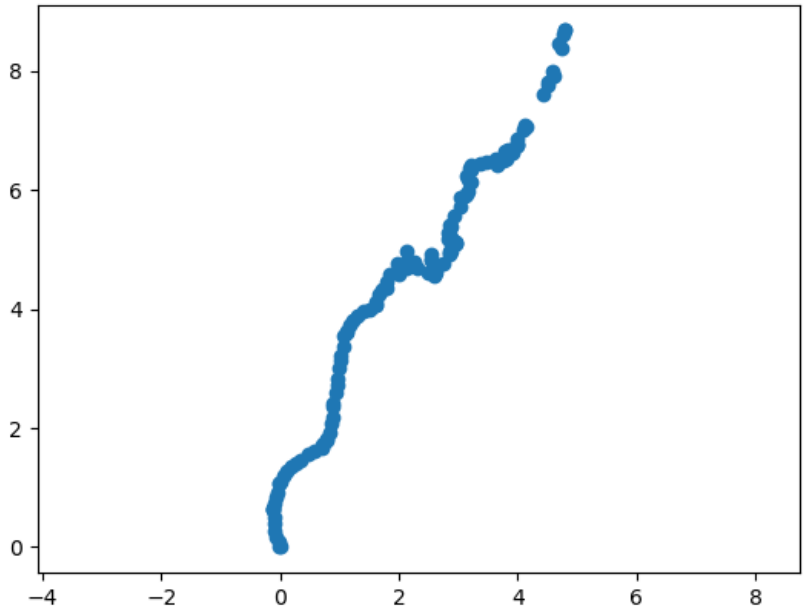
```
index_params= dict(algorithm = FLANN_INDEX_LSH,
                   table_number = 10, # 12
                   key_size = 12,     # 20
                   multi_probe_level = 1) #2
```

```
search_params = dict(checks=100)
```

OR change the Matcher to BF matcher and see how it works

Changing Index param made computational time bit more and there is a significant improvement in the result.



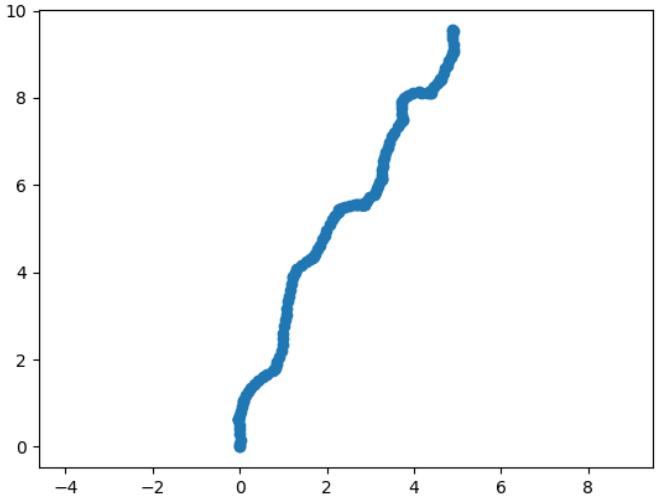


Output after changing the feature matcher to right one

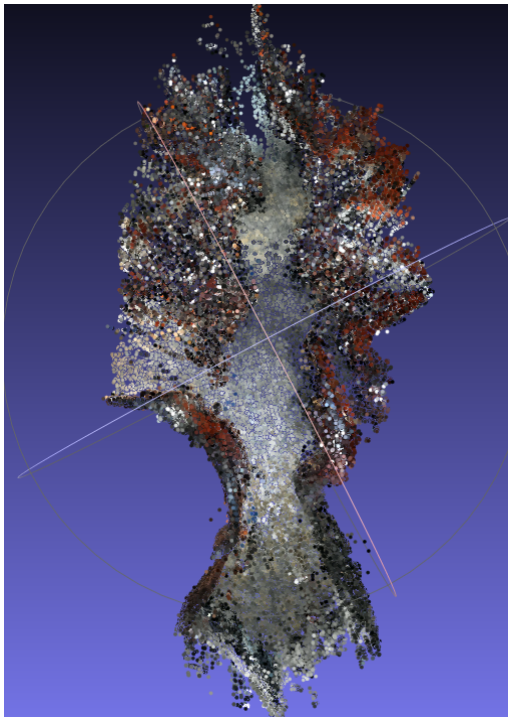
The output still looks discontinuous and scaled. So again the parameters like Lowe's ratio, resolution, clahe, sampling gap, etc were varied to make the trajectory smooth. But varying those didn't improve the results significantly.

The problem was when the map size increases there is an increase of clustering points in the map. This might be due to common feature matching between the images.

So to make it right, Map is divided into sub parts and those sub parts are connected together. So for every N frame, we shall start initialising the map and the pose of these next N frames will be calculated w.r.t. Nth Frame. As we know the pose of Nth frame, we shall convert the next N frames to the world by applying the transformation of the Nth frame and This process continues.

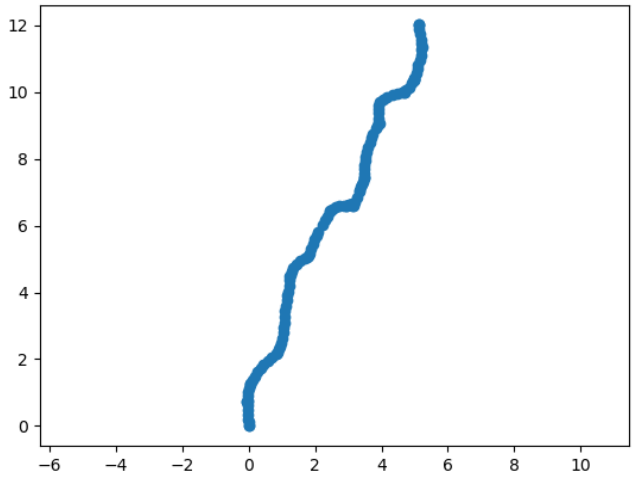


Using Vertical obstacles

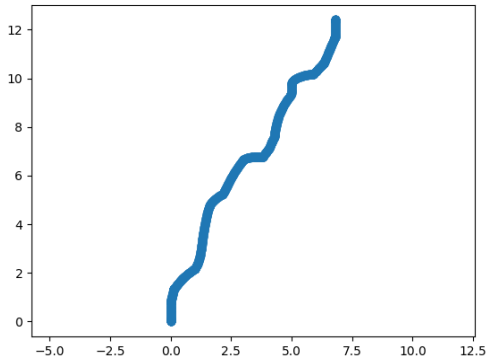


This is the trajectory generated by dividing the map into sub parts using the vertical obstacles. There seems to be still some scale issues but better than the previous one. The generation of the map is done by sampling the sub parts to avoid the overlap between the frames.

The same experiment is performed on the floor points and that result is significantly good.



Floor points



Ground Truth

The above attached is the trajectory using the floor points. The scale is much better than the previous one. This approach can be used to keep track of the pose of the robot. To generate better map algorithms like ICP need to be applied on the key map frames where the sampling of the map takes place.

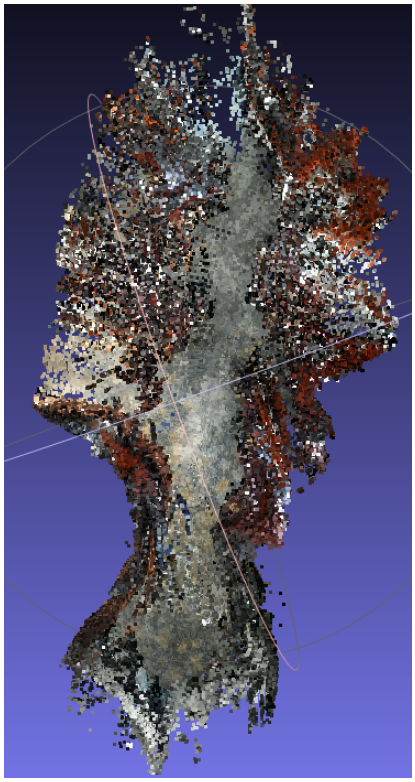
Summary:

- The code takes about 15-20 minutes to run on whole dataset
- After the RTAB map implementation, various parameters were experimented to make the trajectory look smooth and accurate. Some key parameters are:
    - Resolution of Image
    - Lowe's ratio
    - Sampling frequency for Map update
    - Clahe
    - RANSAC threshold
    - Feature Matcher
- Even after tuning the parameters in various combinations and the best result still seems to have few discontinuities in the trajectory.
- The main problem is the matching of the features in a large map is going wrong and leading to incorrect translation.
- So the map is made into parts and a trajectory is generated. The generated trajectory is smooth and better than previous trajectories. Out of vertical obstacles and floor points, the floor points seem to give better estimation of the pose.
- The map is generated by stitching the sub parts. To get a better map those subparts are sampled. To get more better shape if the map ICP has been applied.

ICP map:

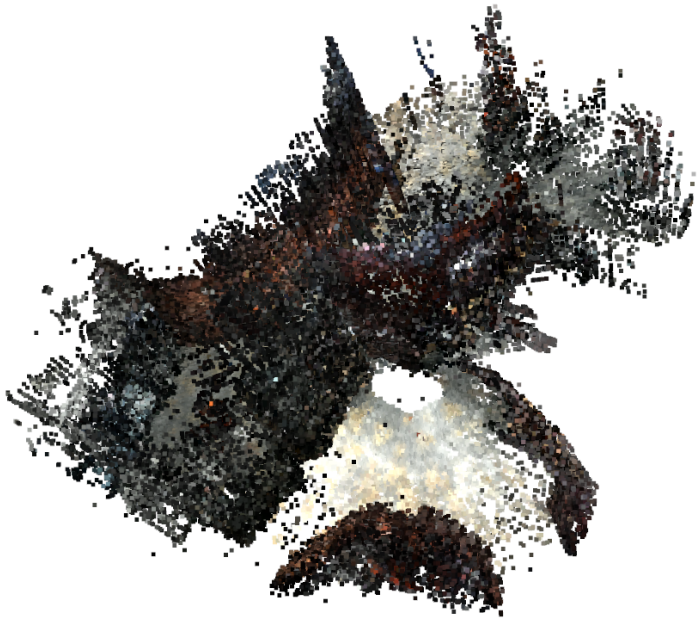
Coloured Point cloud registration is applied to the individual Maps to stitch together.

In the Filtering Voxelization and removing of outliers are applied. Along with this the parameters of rmse and fitness of ICP needs to be adjusted.



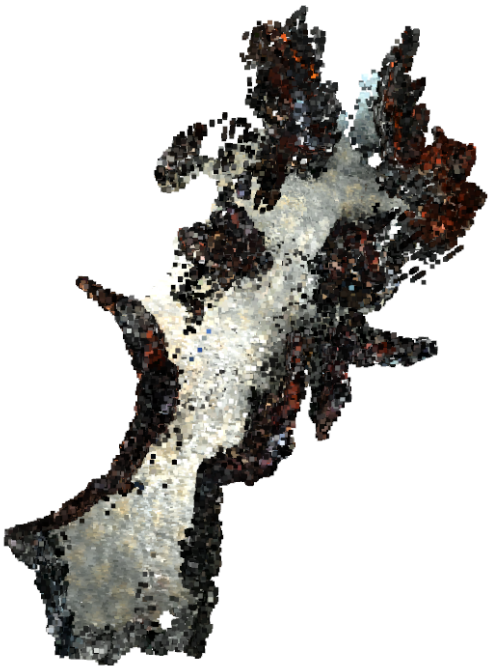
Initially only voxelization is done with voxel_radius of 4cms. The normals are calculated at 20 times of the voxel radius. By taking relative fitness of 1e-2 and rmse 1e-2.

If the values of relative fitness and rmse are reduced then there is cross matching for the images after 600 they are matching to the previous walls and floor.

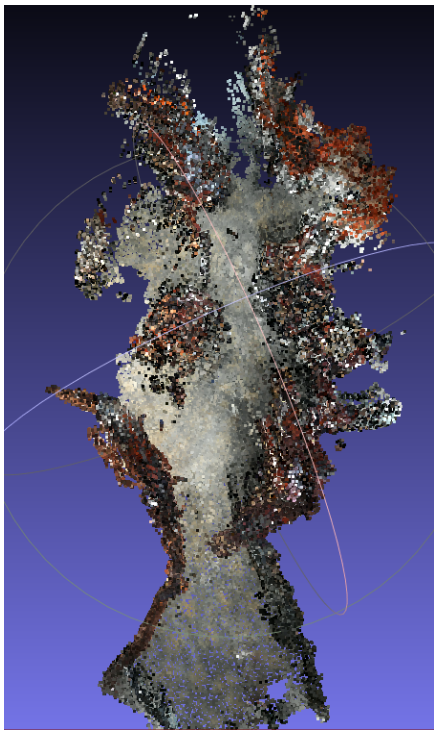


Even after changing the method, rmse, voxelization radius, normal calculation, nothing affected the quality of the map significantly.

The main reason there are few points in the previous frames which are overlapping with the points in the next frame which are not in the right position and affecting the correspondences. So to resolve this problem, outlier filtering is applied. There are 2 methods, one is statistical and another is radial. The statistical one has neat results compared to radial.



Here the outlier removal is applied with st_ratio: 2 and neighbours: 50, with fitness of 1e-6 and rmse 1e-4. With normal radius 2*voxel_radius. All the graphs are running for 5000 iterations.



Reducing the rmse more is trying to shrink the map, like an additional wall layer is formed at the bottom right part.

By making the outlier condition stricter, there is loss of needed information in the map. The above map is when the std_ratio is made 1. This has cut the starting parts of the mao as the map builds.

The radial outliner wasn't able to remove the points which are away from the map like in the bottom left. And the points are denser than earlier.