Sapienza University of Rome

Planar Monocular SLAM
Probabilistic Robotics

Ibis Prevedello

April 28, 2019

1 Introduction

The goal of this project is to develop a SLAM system using Total Least Square algorithm to determine the trajectory of a robot moving in a given environment. This robot perceives the environment only using a stereo camera mounted on its base and its motion by the odometry sensor.

Before the optimization, the information known is the odometry for the trajectory and the landmarks position estimated by triangulation (first column of figure 1 and the goal is to make the prediction match the ground truth values.

The program was written in Python and it succeeds in estimating the right trajectory of the robot and also most of the landmarks, with the exception of some that were bad initialized due to lack of information collected by the camera.

2 Development

First it is necessary to create a feature association, because the robot perceives the landmark appearance and do not know its id, it is necessary to determine what id each appearance corresponds to, matching with the descriptors of each landmark using l^2-norm.

Then, for the landmarks initialization, they need to be triangulated using Direct Linear Transformation (DLT) [1] for each landmark all the times that the robot perceived it. This process depends on the premise that the robot sees the landmark to know that it exists, it can also cause a bad accuracy triangulating its position if there is not enough information.

And finally, run the total least square algorithm with the pose-pose constraints give by the odometry and the pose-landmark constraints given the the images.

3 Dataset

The dataset for this project was generated using the **runba_sim** simulator [2] and is composed of four different class of files:

3.1 camera.dat

Information about the camera.

CAMERA MATRIX
CAMERA TRANSFORMATION
Z NEAR
Z FAR
WIDTH
HEIGHT

3.2 world.dat

Information about the map.

ID $-$ X $-$ Y $-$ Z $-$ APPEARANCE

3.3 trajectory.dat

Information about the trajectory of the robot, containing ground truth values and odometry values.

POSE ID $-$ GROUND TRUTH$\{$X $-$ Y $-$ $\theta\}$ $-$ ODOMETRY$\{$X $-$ Y $-$ $\theta\}$

3.4 meas-xxxxx.dat

Information about the data collected by the robot's camera, with image's position and appearance of each landmark.

POSE ID
GROUND TRUTH$\{$X $-$ Y $-$ $\theta\}$
ODOMETRY$\{$X $-$ Y $-$ $\theta\}$
MEASUREMENT ID $-$ LANDMARK GT ID $-$ ROW $-$ COLUMN $-$ APPEARANCE

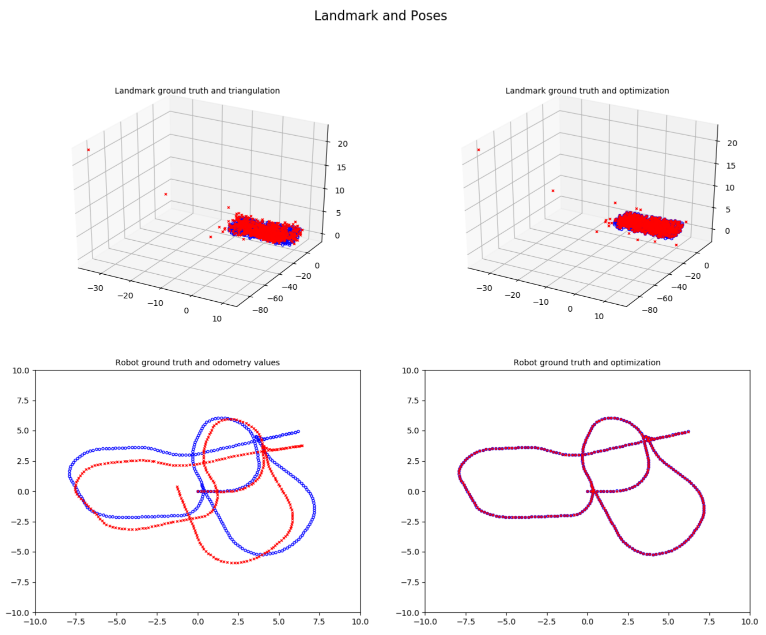

Figure 1: Landmark and poses before and after optimization

4 Results

The total least square algorithm runs in order to optimize the given constraints. This optimization ran for 28 iterations and it is possible to see the final result in image 1. On the bottom part of the image we can notice that the trajectory of the robot after optimization matches exactly the ground truth values.

In image 1 it is possible to notice some outliers landmarks badly initialized due to lack of information for the triangulation. In order to get a better view of the landmarks before and after optimization the outliers were removed and it can be seen in image 2. From this image we can notice that most of the landmarks match the ground truth position after the optimization with the exception of some in the border of the world.

Image 3 shows the chi and number of inliers for the poses and projections.

5 Conclusion

The algorithm implemented provided great results on trajectory and map optimization. Some of the landmarks that did not converge to the ground truth position were due to lack of information in the initialization, this could easily be solved with more data collected by the robot's camera.

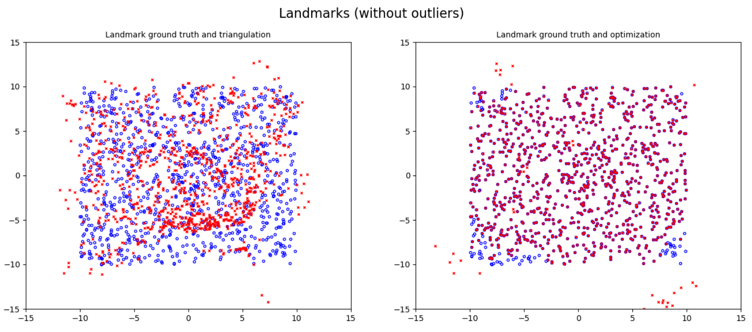

Figure 2: Landmark before and after optimization

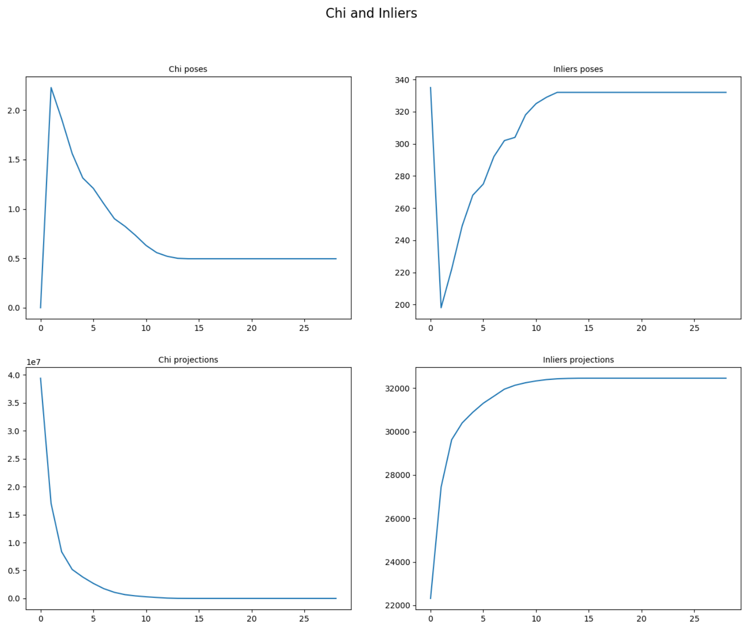

Figure 3: Chi [left column] and inliers [right column] of the poses [first row] and the projections [second row]

This code was implemented in Python because the goal was just to understand how total least squares works, but another way to improve the performance of this algorithm would be converting the code to C++.

References

[1] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. New York, NY, USA: Cambridge University Press, 2 ed., 2003.

[2] "Runba simulator." `https://github.com/ibiscp/Planar-Monocular-SLAM/tree/master/runba_sim`.