Planar Monocular SLAM

Damiano Imola[1], Omar Salem[1], Lorenzo De Rebotti[1], and Giorgio Grisetti[1]

[1]Department of Computer, Control, and Management Engineering "Antonio Ruberti", Sapienza University of Rome, Italy

This report presents an implementation of a planar monocular Simultaneous Localization and Mapping (SLAM) system for a differential drive robot equipped with a single camera. The system integrates wheeled odometry and a stream of point projections to estimate the robot's trajectory and reconstruct the landmarks map. Moreover, we built a robust version of Bundle Adjustment using three different M-estimators: Huber, Cauchy and Tukey, so to improve the robustness of the system against outliers, mitigating their importance. We will compare estimated trajectory and landmark positions against the initial guess, for each and every methodology and M-estimator.

Keywords: Visual SLAM, Monocular, Multi-point, M-estimators, Bundle Adjustment

1 Introduction

Monocular SLAM is a fundamental problem in robotics and computer vision, where a moving robot must estimate its trajectory and map the environment using only a single camera. In our case the environment mapping refers to the estimation of the landmarks positions. This challenge requires robust estimation techniques to infer 3D structure from 2D projections.

In this project, we implement a monocular SLAM system specifically designed for a differential drive robot. The system leverages odometry data and point projections, incorporating intrinsic and extrinsic camera parameters to improve localization and mapping accuracy.

A central contribution of this work is the implementation of a robust bundle adjustment technique. Unlike traditional least-squares approaches that are sensitive to outliers, our method employs three M-estimators: Huber, Cauchy, and Tukey; that allows us to dynamically adjust the influence of wrong measurements. The Huber estimator gives us a smooth transition between quadratic and linear loss, the Cauchy estimator suppresses larger errors in a faster way, and the Tukey estimator aggressively down-weights extreme outliers.

This allows our system to be both precise and resilient to data coming from noisy sensors.

By comparing every estimated result (coming from different approaches) against ground truth, we evaluate the system's performance in terms of trajectory accuracy and error reduction.

2 Triangulation

Triangulation is the process of estimating the 3D position of a point in space given its 2D projections in multiple images and the corresponding camera matrices. Given matching image points p_i and their respective camera projection matrices M_i for each view, the goal is to recover the 3D point P in homogeneous coordinates.

Is it worth mention that in our problem, the camera is rigidly mounted on the moving robot, meaning that its extrinsic transformation (i.e., its position and orientation relative to the robot's reference frame) is constant and known. Hence, what changes over time is the pose of the robot as it moves through the environment.

The triangulation inside the project is done following the tecnical report[1] from Slabaugh Et al. Greg Slabaugh (2001). The core idea is to solve a linear system derived from minimizing the sum of squared perpendicular distances from the estimated point $P(x, y, z)$ to each ray. This is expressed via (see equation 11 of the paper)

$$E(x,y,z) = \sum_{i=1}^{N} \left[(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2 - \left(a_i(x - x_i)^2 + b_i(y - y_i)^2 + c_i(z - z_i)^2 \right) \right] \tag{1}$$

with (x_i, y_i, z_i) the starting point of ray i (the center of the camera) and (a_i, b_i, c_i) the direction of the ray. The "second part" of the formula is the one needed for projecting the point into the ray direction. Note that the minimization is not done explicitly using an iterative approach but it's done analitycally by solving the linear system $\mathbf{x} = A^{-1}b$ (see equation 22 of the paper):

```
P = np.linalg.solve(A, B)
return P.flatten()
```

Then, the core part of the function used for triangulation (i.e. `triangulate_multiple_views(...)`) is the building of the matrix A and of the vector B (see equation 21 of the paper). In particular, this is done here

```
# looping over ray observations
for i in range(points.shape[1]):
    a, b, c = directions[:, i]
    x, y, z = points[:, i]

    # building matrix A
    A[0, 0] += 1 - a * a
    A[0, 1] += -a * b
    A[0, 2] += -a * c
    A[1, 1] += 1 - b * b
    A[1, 2] += -b * c
    A[2, 2] += 1 - c * c
```

[1] Found inside the *"Stereo Vision: Epipolar Geometry"* slides of Technincal University of Instanbul (ITU) for the course of 3D Vision held by Professor Gozde Unal.

```
# building vector B
B[0, 0] += (1 - a * a) * x - a * b * y - a * c * z
B[1, 0] += -a * b * x + (1 - b * b) * y - b * c * z
B[2, 0] += -a * c * x - b * c * y + (1 - c * c) * z
```

and since A is symmetric we mirror the matrix, although not explicitly mentioned in the paper, using

```
A[1, 0] = A[0, 1]
A[2, 0] = A[0, 2]
A[2, 1] = A[1, 2]
```

3 Bundle Adjustment (Total Least Squares)

Bundle Adjustment (BA) is a nonlinear optimization technique used in Simultaneous Localization and Mapping (SLAM) to refine camera poses and 3D feature positions by minimizing the reprojection error. I won't comment out the code since it is mainly based on the Professor Grisetti's code (i.e. `26_total_least_squares`) available on GitLab.

4 Robust Bundle Adjustment (Robust Total Least Squares)

In order to bring robustness to outliers, we decided to use M-estimators inside the Bundle Adjustment, so to obtain a **Robust Bundle Adjustment**. The idea is to control and damp the amount of importance of the outliers inside the Bundle Adjustment. To do so, I've decided to compare three different robustifiers:

- **Huber Loss**: hybrid between squared loss and absolute loss, it weights quadratically small errors and linearly outliers, so that it reduces the influence of large residuals, without ignoring them. The equation is

$$\rho(e) = \begin{cases} \frac{1}{2}e^2 & \text{if } |e| \leq \delta \\ \delta\left(|e| - \frac{1}{2}\delta\right) & \text{if } |e| > \delta \end{cases} \tag{2}$$

  where δ is the threshold where quadratic loss switch to linear. When using Huber robustifier, you can modify δ in the code by setting `robust_param` to your desired value.

- **Cauchy Loss**: this is a smooth alternative and suppresses large errors in a faster way. The equation is

$$\rho(e) = \frac{c^2}{2}\log\left(1 + \frac{e^2}{c^2}\right) \tag{3}$$

  where c is a scaling factor that controls the "outlier rejection": as the absolute value of e grows, the loss saturates, preventing that large errors dominates the cost function. When using Cauchy robustifier, you can modify c in the code by setting `robust_param` to your desired value.

- **Tukey (Bisquare) Loss**:strictly bounded loss, to to do not let large errors contribute to the optimization. The function is

$$\rho(e) = \begin{cases} \frac{c^2}{6}\left(1 - \left(1 - \frac{e^2}{c^2}\right)^3\right) & \text{if } |e| \leq c \\ \frac{c^2}{6} & \text{if } |e| > c \end{cases} \tag{4}$$

  where c is a threshold for ignoring residuals. It fully removes high-magnitude outliers; but can potentially discard useful data. When using Tukey robustifier, you can modify c in the code by setting `robust_param` to your desired value.

The visualization of these M-estimators is available in Figure 1.

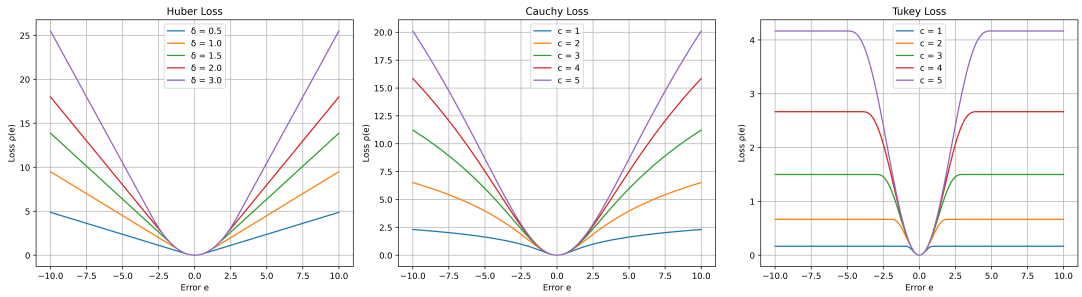

Figure 1: From left to right: Huber, Cauchy, and Tukey.

Inside the code, we are using these concepts in `build_robust_linear_system_poses` and `build_robust_linear_system_projections`, where the difference w.r.t. the plain implementation of these functions (done in the project for the classic Bundle Adjustment), is in:

- the computation of the residuals, using `residual = np.linalg.norm(e)`
- the computation of the weight of the residual using the chosen M-estimator, using `weight = robust_weight(residual, robust_method, robust_param)`
- the update of the H matrix and b vector, which now have a weighting term that multiplies the entries. Follows the code from `build_robust_linear_system_projections`.

```
# ===== H MATRIX =====
H[pose_matrix_idx:pose_matrix_idx + pose_dim,
   pose_matrix_idx:pose_matrix_idx + pose_dim] +=
   weight * Jr.T @ Jr

H[pose_matrix_idx:pose_matrix_idx + pose_dim,
   landmark_matrix_idx:landmark_matrix_idx +
   landmark_dim] += weight * Jr.T @ Jl

H[landmark_matrix_idx:landmark_matrix_idx +
   landmark_dim, landmark_matrix_idx:
   landmark_matrix_idx + landmark_dim] += weight * Jl.T
    @ Jl
```

```
H[landmark_matrix_idx:landmark_matrix_idx +
   landmark_dim, pose_matrix_idx:pose_matrix_idx +
   pose_dim] += weight * Jl.T @ Jr

# ===== b VECTOR =====
b[pose_matrix_idx:pose_matrix_idx + pose_dim] += weight
    * Jr.T @ e
b[landmark_matrix_idx:landmark_matrix_idx +
   landmark_dim] += weight * Jl.T @ e
```

5 Results

I've tested the system in two different scenarios:

- using the BA over the whole system (poses + landmarks) right after the triangulation
- pre-optimizing the guess using the BA in a "only-landmarks" fashion and then, from the refined guess, run the BA over the whole system (poses + landmarks)

5.1 Without pre-optimization

The visualization of the refined trajectories using various techniques, and of the evolution of the χ and of the number of inliers among iterations, without using the pre-optimization technique, can be seen respectively in Figure 2 and Figure 3.

5.2 With pre-optimization

The visualization of the refined trajectories using various techniques, and of the evolution of the χ and of the number of inliers among iterations, using the pre-optimization technique, can be seen respectively in Figure 4 and Figure 5.

5.3 Simple Bundle Adjustment

The results of the base project (i.e. of the simple Bundle Adjustment), can be visualized in Figures 6, 7, and 8.

6 Conclusions

The implemented planar monocular SLAM system successfully estimates the trajectory of a differential drive robot and reconstructs a 3D representation of its environment using a single camera.

The combination of odometry-based initialization, point triangulation, and Bundle Adjustment proves to be effective in reducing localization errors, leading to improved accuracy when compared to ground truth.

Moreover, when integrating robust bundle adjustment module, the system has shown enhancements in resilience to outlier measurements.

Further works on hyperparameter tuning for M-estimators can help the system to still increase it's robustness.

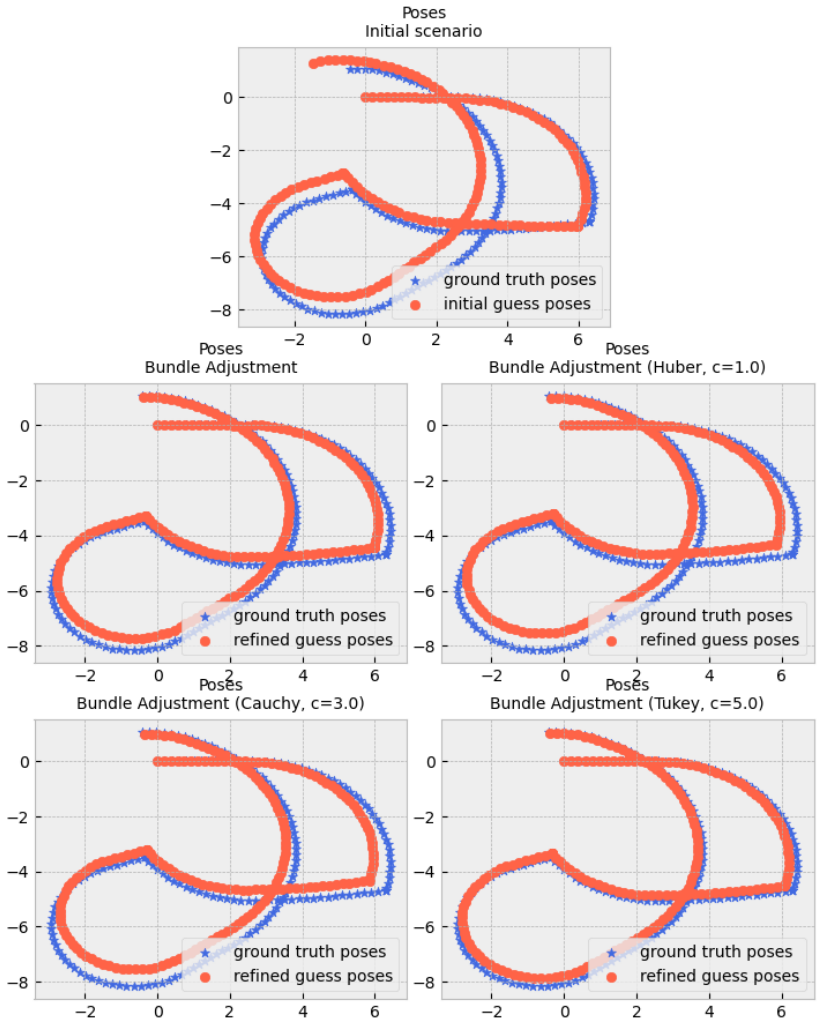

Figure 2: Estimated trajectories without pre-optimization.

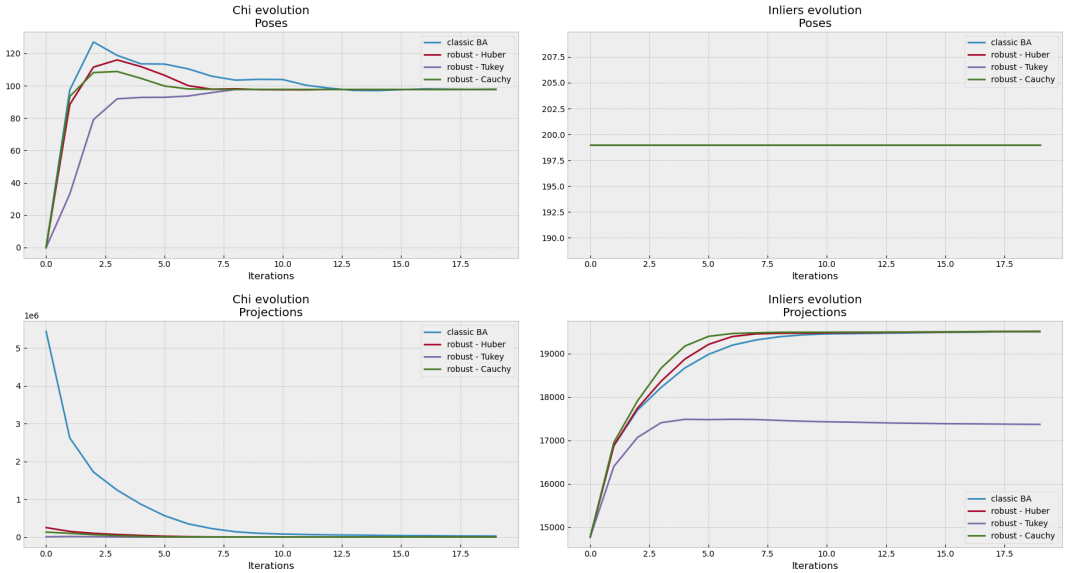

Figure 3: χ and inliers evolution over iterations without pre-optimization.

References

Mark Livingston Greg Slabaugh, Ron Schafer. Optimal Ray Intersection For Computing 3D Points From N-View Correspondences. 2001. URL `https://www.eecs.qmul.ac.uk/~gslabaugh/publications/opray.pdf`.

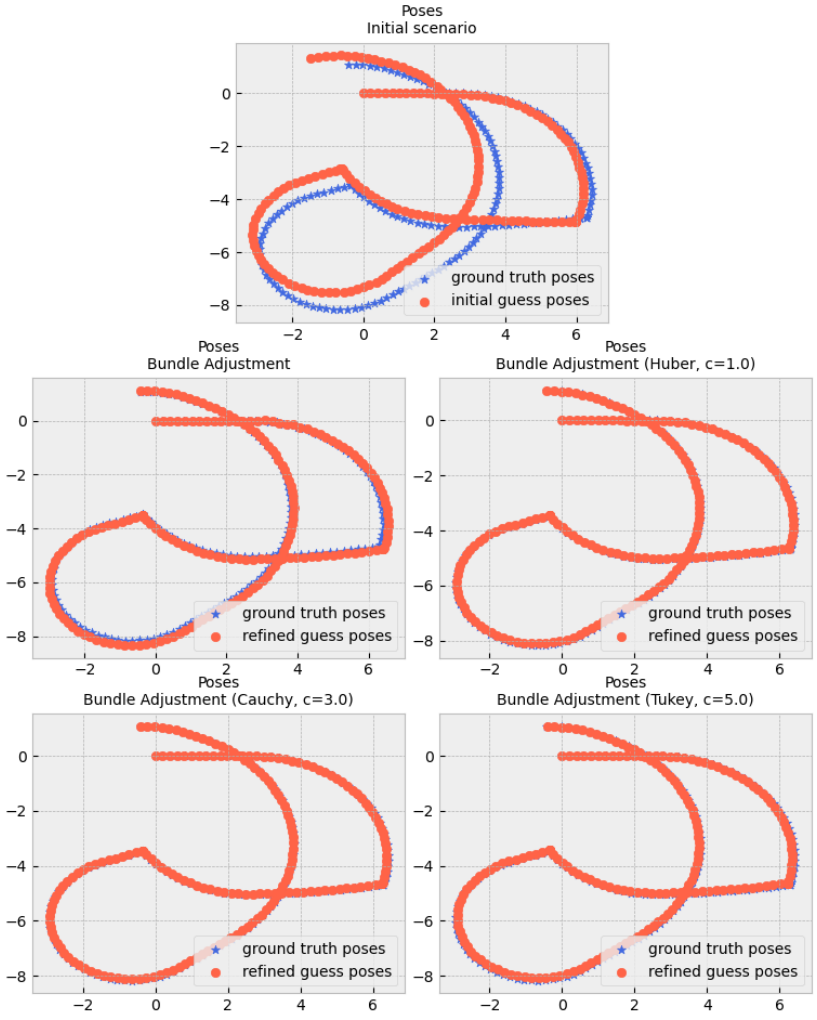

Figure 4: Estimated trajectories with pre-optimization.

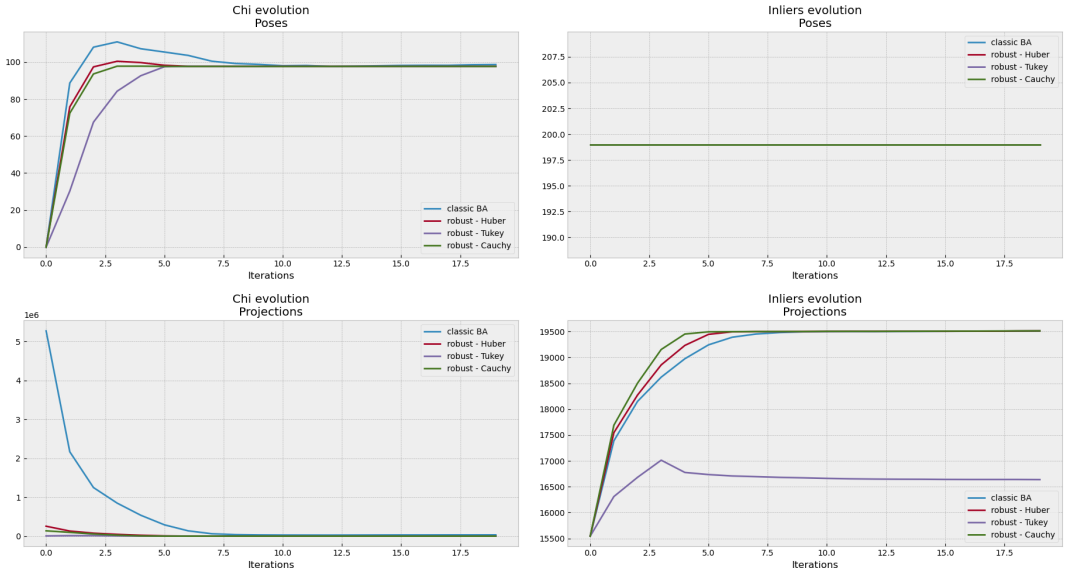

Figure 5: χ and inliers evolution over iterations with pre-optimization.

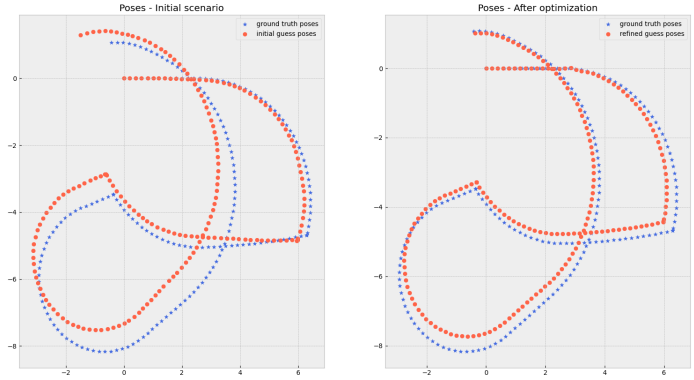

Figure 6: Estimated trajectories with simple BA.

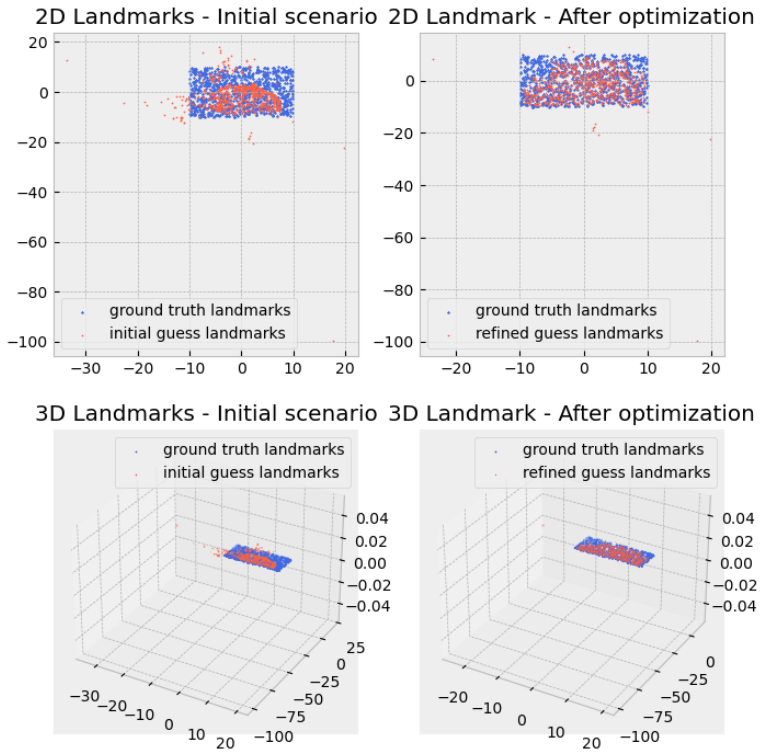

Figure 7: Landmarks optimization with simple BA.

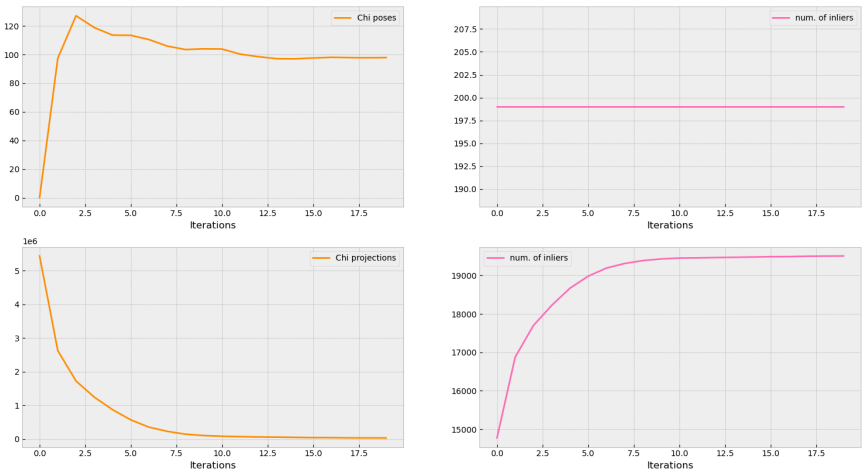

Figure 8: χ and inliers evolution over iterations of simple BA.