2D Pose Graph optimization
(Weighted Non-linear least sqaures)

Subramanian Krishnan

andrew ID: subramak

Abstract—**Robot state estimation is an extremely common problem in autonomous navigation and control. More generally state estimation is a subset of the simultaneous localization and mapping (SLAM) problem. State information comes in mainly from odometry and sensor measurements. As odometry measurements are noisy, the state estimate deteriorates over time. To compensate for this, global graph optimization is performed to refine states to better explain the sensor measurements. In this project, I implement one such graph optimization procedure as described in [1] for the problem of 2d planar poses (3 DOF - x,y,θ). The method is a modified Stochastic Gradient Descent algorithm implemented in python and tested on the M3500 and Intel datasets.**

Keywords—**SLAM, Pose graph, State estimation, Stochastic Gradient Descent, optimization, M3500**

I. Introduction

Autonomous navigation of mobile robots require planning from a known state in a map to a goal state. In order to execute a set of actions created from a plan, the current position and orientation of the robot is necessary. This is the need for state estimation for mobile robots.

In order to estimate the state of the robot, the best way would be to observe it externally using another system, for eg. Vicon motion capture. Markers on the robot can be tracked to know exactly where the robot it. However, this idea doesn't scale well. It's incredibly expensive to set up a large area Vicon system. Plus, you'd like the robot to go into unexplored areas too where you'd still want a state estimate.

The next alternative is to set up sensors on the robot to measure motion. Easy ideas are wheel encoders for wheeled mobile robots, IMU for drones. Given encoder readings, we can estimate pose of the robot by using the robot motion model and set up a system of odometry equations. Similarly acceleration information of IMU can be double integrated to get 3D position and orientation estimates. Although simple, these methods are prone to loss of accuracy over time. Why? Noise at every step of integration accumulates over time producing poorer state estimates.

Can we overcome this? Yes. Using what? Sensors that indirectly output motion. Example? Take laser scans. Two planar scans can be aligned using scan matching to give a 2D motion transformation. Take camera images. Motion between images taken from two different poses can be recovered using epipolar constraints (perform feature matching, estimate fundamental matrix, estimate rotation and translation). Thus at any time, we may have two sources of information telling us the state of the system: 1. Odometry 2. Sensors (cameras, lidars). How do we combine these two sources? That is what this project answers by implementing an old solution proposed by Edwin Olson in his 2006 ICRA paper [1]. More commonly the problem is called loop closure in SLAM.

II. Background and Notation

In this project I would be dealing with a planar mobile robot to keep things simple. The states (or commonly called pose) of a planar mobile robot are the (x,y) positions and orientation with respect to a frame of reference (let's call this world frame F_w). Let us denote state at a time discrete timestep k as p_k where k varies varies from 0 to n. This implies $p_0 = \begin{bmatrix}0 & 0 & 0\end{bmatrix}^T$ is the starting pose.

In the introduction section, we defined motion being estimated from other sensor sources. This motion can be represented as a 3x3 2D frame transformation made of a rotation and translation. A frame is just a coordinate system in 2D space (x-axis vector, y-axis vector and origin). So robot motion can be written mathematically using frame motion. If a frame at time k, F_k, rotates by an angle θ and translates by $\mathbf{t}$ to become F_{k+1}, the motion can be represented as F^k_{k+1} given by,

$$F^k_{k+1} = \begin{bmatrix} cos(\theta) & -sin(\theta) & t_x \\ sin(\theta) & cos(\theta) & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

As as aside, point represented in k+1th frame, q^{k+1}, can be transformed into the k-th frame, q^k as,

$$q^k = F^k_{k+1} q^{k+1} \tag{2}$$

Thus we can represent states as global poses, $p_k = [x_k, y_k, \theta_k]^T$ in the world frame or as a frame transformation to the world frame P^w_k or in short hand P_k,

$$\begin{aligned} P_k &= \begin{bmatrix} cos(\theta_k) & -sin(\theta_k) & x_k \\ sin(\theta_k) & cos(\theta_k) & y_k \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{R_k} & \mathbf{t_k} \\ 00 & 1 \end{bmatrix} \end{aligned} \tag{3}$$

An example pose graph is presented in Fig. 1 [2]. The notation is butchered here. The poses, $x \Longleftrightarrow p$, in this figure. Thus p_0 in our notation is same as x_0 in this figure. Thus a pose graph, G, can be defined as the set (N, E) where N refers to the set of nodes which are the poses at different times k, and E refers to set of edges which are measurements. An edge

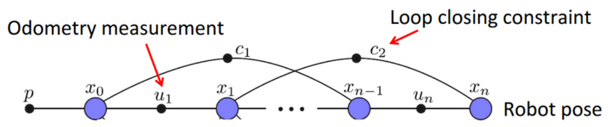

Figure 1. Pose graph

between two consecutive nodes is an odometry measurement. In Fig. 1, u_k refer to odometry measurements. For purely odometry reliant pose graph (which would be the case when we start solving the optimization) we can write,

$$u_k = p_k - p_{k-1} \tag{4}$$

At the start of the optimization, Eq. 4 is true for all k as the states were estimated only using odometry. As the optimization proceeds, this fails to hold as we minimize the net error.

Loop closure or non-consecutive pose constraints are represented by c_i in Fig. 1. Due to noisy odometry (using which initial poses were computed), the measurement c_i may not be equal to the difference in the nodes that the edge connects. This difference is what we'd try to minimize. At the start of optimization, $c_{ij} \neq p_j - p_i$.

Each measurement, be it odometry or loop closure, has an associated covariance. The covariance is a measure of how accurate that measurement is. More specifically, the inverse of the covariance can be used as a weighting factor for the error. We'll represent this covariance as, Σ_i, for the i-th measurement. Thus E consists of all u_k and c_i along with their covariances, Σ.

III. Problem definition

The quirk in this problem is that the measurements are in relative frames and the poses are in global frames. Say a measurement, u_{ij} connects poses, p_i and p_j. We need a function f to transform from state space (poses in global reference) to measurement space (in reference frame of F_i). We can write f as,

$$f(p_i, p_j) = \begin{bmatrix} \mathbf{R_i}^T(\mathbf{t_j} - \mathbf{t_i}) \\ \theta_j - \theta_i \end{bmatrix} \tag{5}$$

This is where the complication begins. **f** is a non-linear function due to the presence of the rotation. Thus the system cannot be solved linearly. The problem can be phrased as minimizing the following objective,

$$\min_{p \in N} \sum_E (f(p_i, p_j) - u_{ij})^T \Sigma_{ij}^{-1} (f(p_i, p_j) - u_{ij}) \tag{6}$$

If all the states and measurements were to be flattened and stacked together, h() transfers these states to measurement space, we can write more generally as,

$$\min_p (h(p) - u)^T \Sigma^{-1} (h(p) - u) \tag{7}$$

IV. Methodology

This problem is traditionally solved iteratively. The cost can be linearized about the current estimate of the state.

$$h(p) = H|_{p\_current} + J|_{p\_current} \Delta p \tag{8}$$

J has a 3-row block for every constraint u_{ij} with only 2 sets of 3x3 blocks (corresponding to x_i and x_j) populated in that 3-row block matrix. We can define the residual error and search direction as,

$$\begin{aligned} r &= u - H|_{p\_current} \\ d &= \Delta p \\ cost &= (Jd - r)^T \Sigma^{-1} (Jd - r) \end{aligned} \tag{9}$$

This quadratic minimization can be solved by differentiating w.r.t d and setting it to zero which would give,

$$(J^T \Sigma^{-1} J) d = J^T \Sigma^{-1} r \tag{10}$$

This is where the sheer size of the problem comes into the picture. The M3500 dataset used in this project results in a J of dimension 16359 x 10,500 and Σ of dimension 16359 x 16359. The multiplication and inverse cannot be done in reasonable time for online operation.

The alternative method suggested in [1] is presented here. Let us consider the cost (scalar) of a single constraint whose residual, r_i = u - f(p), is 3x1 and covariance (Σ_i) of the measurement is 3x3.

$$c_i = r_i^T \Sigma_i^{-1} r_i \tag{11}$$

Let us find the gradient of the cost w.r.t descent direction d (3nx1 where n poses are optimized) using chain rule,

$$\begin{aligned} \nabla c_i &= \frac{\partial c_i}{\partial d} \\ &= \frac{\partial c_i}{\partial r_i} \frac{\partial r_i}{\partial d} \\ &= -2 r_i^T \Sigma^{-1} J_i \end{aligned} \tag{12}$$

where J_i would be a [3 x state_vars] row block matrix. This results in Δc_i to be a row vector 1 x state_vars with only 6 values filled in the a long zero row vector. These six values would be at the poses corresponding to the nodes connected by the measurement under consideration. The descent direction is given by factoring a step to the negative gradient.

$$d = 2\alpha J_i^T \Sigma_i^{-1} r_i \tag{13}$$

If we add all the ∇c_is we'd get RHS of Eq. 10 with a scale factor of 2α. To solve for the exact d in Eq. 10, we need to get a scaling matrix M such that,

$$\begin{aligned} M \approx J^T \Sigma^{-1} J &= diag(J^T \Sigma^{-1} J) \\ d &= 2\alpha M^{-1} J_i^T \Sigma^{-1} r_i \end{aligned} \tag{14}$$

V. The Jacobian

To do the whole process efficiently without having to invert huge matrices at any stage, the authors propose a state space representation called incremental pose. In the problem definition section, I defined f() as a mapping between global pose to measurement space. Now, instead of the global pose, we'd use the incremental pose. Hence f() would change. If the original global poses are denoted by (x_i, y_i, θ_i), then

$$p_{incremental} = \hat{p} = \begin{bmatrix} \hat{x}_0 \\ \hat{y}_0 \\ \hat{\theta}_0 \\ \hat{x}_1 \\ \hat{y}_1 \\ \hat{\theta}_1 \\ \hat{x}_2 \\ \hat{y}_2 \\ \vdots \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ \theta_0 \\ x_1 - x_0 \\ y_1 - y_0 \\ \theta_1 - \theta_0 \\ x_2 - x_1 \\ y_2 - y_1 \\ \vdots \end{bmatrix} \tag{15}$$

Let a measurement u_{ij} connect two poses represented by incremental pose representation, $\hat{p}_i, \hat{p}_j$. To transform the pose information (in state space) to measurement space (u_{ij} refers to pose at j w.r.t i), we can write f() as,

$$f(\hat{p}_i, \hat{p}_j) = \begin{bmatrix} \sum_{k=i+1}^{j} \hat{x}_k \\ \sum_{k=i+1}^{j} \hat{y}_k \\ \sum_{k=i+1}^{j} \hat{\theta}_k \end{bmatrix} \tag{16}$$

Note that f() outputs the difference in global pose of the two input incremental poses, i.e, this is not to be compared with the measurement directly as the measurement is in local frame of reference *i*. So, to actually compute the residual, one would have to transform the measurement to global frame (by doing $u_{ij}P_i$).

So the jacobian, J_{ij}, of **f** corresponding to a measurement u_{ij} now has a much simpler structure with 3 x 3 blocks of zeros and identity matrices,

$$\begin{aligned} J_{ij} &= \frac{\partial f_{ij}}{\partial \hat{p}} \\ &= [\mathbf{0}_0 | \mathbf{0}_1 | \ldots | \mathbf{I}_{i+1} | \mathbf{I}_{i+2} | \ldots | \mathbf{I}_j | \mathbf{0}_{j+1} | \ldots | \mathbf{0}_n] \end{aligned} \tag{17}$$

In order to define the step size, the authors suggested it to be the inverse of current iteration number along with a factor,

$$\alpha = \frac{1}{\gamma k} \tag{18}$$

where k is the current iteration number and γ is given as smallest covariance in the measurements for each type of variable (x,y,θ),

$$\begin{aligned} \Sigma_i^{-1} &= \begin{bmatrix} \sigma_{ix} & 0 & 0 \\ 0 & \sigma_{iy} & 0 \\ 0 & 0 & \sigma_{iz} \end{bmatrix} \\ \gamma_{x,y,z} &= \min_{\sigma_{x,y,x}} \Sigma_i^{-1} \qquad \forall i = [0, m] \end{aligned} \tag{19}$$

Remember that the relation between global pose, p_i and incremental pose, $\hat{p}_i$ are related as,

$$\begin{aligned} p_i &= \hat{p}_i + \hat{p}_{i-1} \\ \hat{p}_i &= p_i - p_{i-1} \end{aligned} \tag{20}$$

VI. The algorithm

I reuse the algorithm section described in [1], which describes a succinct implementation of the equations written before. The notation is consistent: p refers to 3 vector global pose (x,y,θ), P refers to the 2D transformation matrix version of the 3 vector pose, *diag* refers to vectorizing the diagonal of the matrix, *mod2pi* refers to forcing input radian angle to lie withing $-\pi$ and π.

Algorithm 1 SGD Optimize Graph

```
1: iters = 0
2: loop
3:   iters++
4:   $\gamma = [\infty\ \infty\ \infty]^T$
5:
6:   {Update approximation $M = J^T\Sigma^{-1}J$}
7:   $M$ = zeros($numstates, 3$)
8:   for all {$a$, $b$, $t_{ab}$, $\Sigma_{ab}$} in $Constraints$ do
9:     $R = Rot(P_a)$
10:    $W = (R\Sigma_{ab}R^T)^{-1}$
11:    for all $i \in [a+1, b]$ do
12:      $M_{i,1:3} = M_{i,1:3} + diag(W)$
13:      $\gamma = \min(\gamma, diag(W))$
14:
15:  {Modified Stochastic Gradient Descent}
16:  for all {$a$, $b$, $t_{ab}$, $\Sigma_{ab}$} in $Constraints$ do
17:    $R = Rot(P_a)$
18:    $P_b{}' = P_a T_{ab}$
19:    $r = p_b{}' - p_b$
20:    $r_3 = mod2pi(r_3)$
21:    $d_{1:3} = 2(R^T\Sigma_{ab}R)^{-1}r$
22:
23:    {Update $x$, $y$, and $\theta$}
24:    for all $j \in [1, 3]$ do
25:      $alpha = 1/(\gamma_j * iters)$
26:      $totalweight = \sum_{i\in[a+1,b]} 1/M_{i,j}$
27:      $\beta = (b-a)d_j\alpha$
28:      if $|\beta| > |r_j|$ then
29:        $\beta = r_j$
30:      $dpose = 0$
31:      for all $i \in [a+1, N]$ do
32:        if $i \in [a+1, b]$ then
33:          $dpose = dpose + \beta/M_{i,j}/totalweight$
34:        $p_{i,j} = p_{i,j} + dpose$
```

Figure 2. Pseudo-code

A small quirk in implementation is that they've proposed to solve for the small change in pose (dpose), to reduce cost, in the **global frame**. This means that at each stage, all values are considered in global frame. Seen from line 34 that *dpose* is added to *p*.

W is the inverse of the measurement covariance transformed to the global frame, i.e information matrix of that measurement in the global frame. To propagate the covariance it is multiplied by R_a and R_a^T [line 10]

Every measurement is defined between two poses/frames A and B. The residual is defined as moving pose A to a hypothetical pose B by applying the motion T_{ab} as given by the measurement. The difference between actual pose B and this motion model based pose B is the error (after correcting euclidean warping)

Line 12 tracks only the diagonal elements of the M matrix resized to n x 3 where n is the number of poses. Say there's only one measurement that connects pose 1 and pose 3 (remember pose numbering starts at 0 with p_0 being [0,0,0]). There are n poses. This implies J is of dimension 3 x 3n, Σ^{-1} is 3 x 3, resulting M is 3n x 3n. We know that we're trying to approximate only the diagonal elements of M which would be 3n elements which can thus be resized as n x 3 [line 7]. An example M would like this,

$$
\begin{aligned}
J &= [0|0|I|I|--0--] \\
J^T\Sigma^{-1} &= \begin{bmatrix} 0_{3\times3} \\ 0_{3\times3} \\ I_{3\times3} \\ I_{3\times3} \\ 0_{3\times3} \\ \vdots \end{bmatrix} \Sigma^{-1} = \begin{bmatrix} 0_{3\times3} \\ 0_{3\times3} \\ \Sigma^{-1} \\ \Sigma^{-1} \\ 0_{3\times3} \\ \vdots \end{bmatrix} \\
J^T\Sigma^{-1}J &= \begin{bmatrix} 0_{3\times3} \\ 0_{3\times3} \\ \Sigma^{-1} \\ \Sigma^{-1} \\ 0_{3\times3} \\ \vdots \end{bmatrix} [0|0|I|I|--0--] \qquad (21) \\
&= \begin{bmatrix} 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & \cdots \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & \cdots \\ 0_{3\times3} & 0_{3\times3} & \Sigma^{-1} & \Sigma^{-1} & 0_{3\times3} & \cdots \\ 0_{3\times3} & 0_{3\times3} & \Sigma^{-1} & \Sigma^{-1} & 0_{3\times3} & \cdots \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & \cdots \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}
\end{aligned}
$$

In that, we only care about the diagonal elements of that matrix. Thus we add the diagonal of Σ^{-1} to the every 3x3 block diagonal stored in M from i+1 th row to jth row [line 12]

Sometimes, the resulting d could potentially be very large. This is because M^{-1} helps make a weighted sum of the residuals. If the constraint edge is across two nodes that are far apart, then the resulting sum could be high and throw us out of the minima. Hence, line 27 performs a clamping of the step using a heuristic factor (b-a) which is representative of the length of the constraining edge.

Line 33 does something that is not in the mathematical derivation. Instead of directly adding the M^{-1} weighted residual, they normalize the M^{-1} matrix such that each variable axis (x, y, θ) sums to one. Essentially, this takes the whole step and proportionally splits it into different sized steps for the intermediate nodes making the trajectory and uses that for correcting the node's state values. I couldn't figure out why they did this and not directly add un-normalized weights.

VII. Experiments

I used two data: M3500 and Intel sets to validate the algorithm. They were obtained from [3]. The M3500 is an artificial dataset created by Olson et. al to test graph optimization on a scale that was not available from real world data as ground truth for such long trajectories were not viable. The Intel dataset is a more real-world version of the data where Hahnel et al. mapped the Intel research lab in Seattle using wheel odometry and laser range finder measurements.

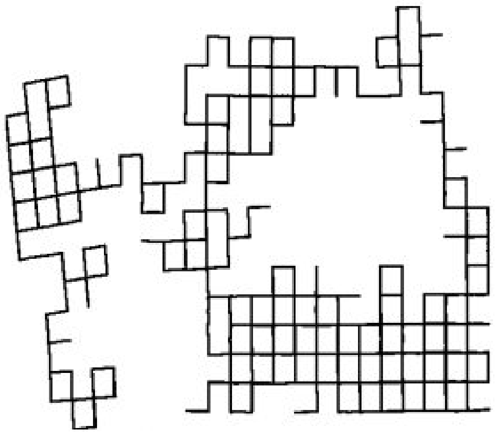
Figure 3. M3500 Ground truth trajectory

The M3500 is chosen as it provides a clear visual cue of whether the algorithm is working. The ground truth trajectory is shown in Fig. 3. It looks like somebody walked along the streets of upper Manhattan. There are 3500 nodes in the given graph. Hence the name M3500. To account for measurements, there are 5454 measurements along with their covariances given as edges of the graph. I used the TORO version of the dataset whose definition is presented in [4].

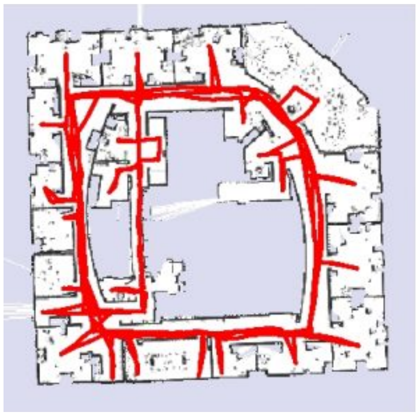
Figure 4. Intel Ground truth trajectory

The Intel dataset4 was chosen to show transfer-ability to real-world data. It has 1228 poses, 1505 constraints/edges.

VIII. Results

I ran each dataset for 200 iterations. Code was written in python and is available at: https://github.com/SubramanianKrish/poseGraphOptimization_SGD The following figures 5 and 6 show initial estimate passed into the optimization. The initial cost is 23318533685.31057 for M3500 and 18321178.717 for intel.

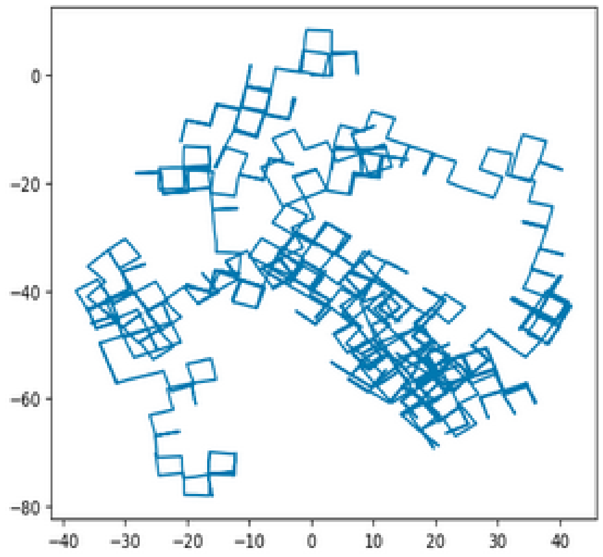

Figure 5. M3500 initial odometry estimate pre-optimization

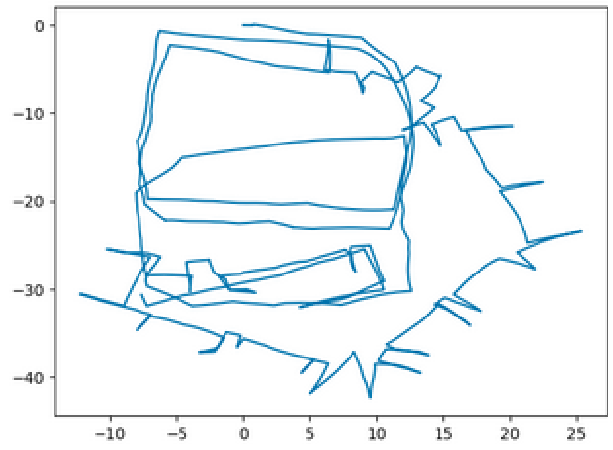

Figure 6. Intel initial odometry estimate pre-optimization

The final optimized trajectory and cost over iterations are plotted in the following figures. The final optimized costs are 65258908.22 for M3500 and 27400.5695 for Intel. For 100 iterations, M3500 took 85.42 secs and Intel took 22.53 secs. The paper cited that 100 iterations for M3500 was 2.9 secs but this was in C++ with a tree implementation for pose update (which is O(log N) as compared my python O(N) implementation).

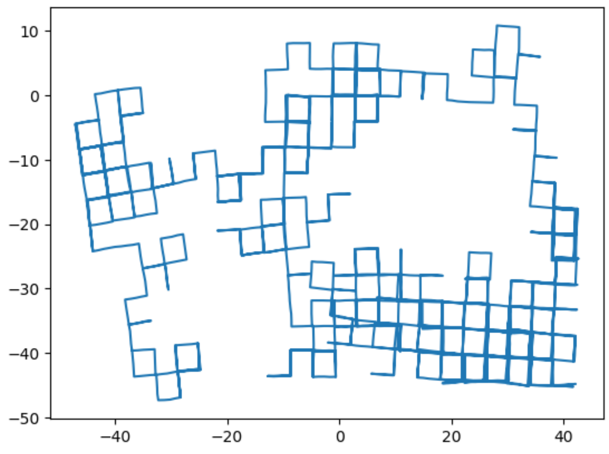

Figure 7. M3500 after optimization - 200 iterations

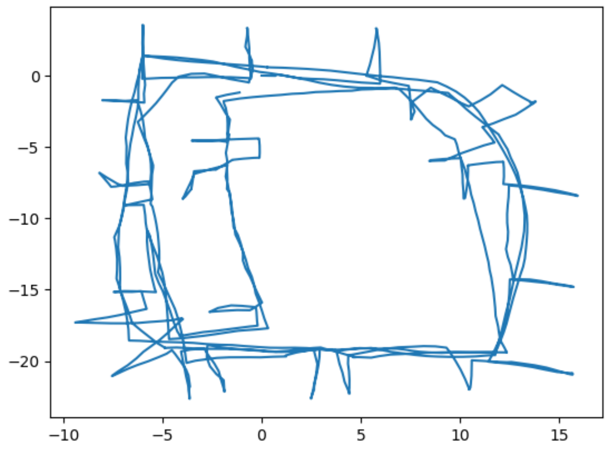

Figure 8. Intel after optimization - 200 iterations

IX. Conclusion

I had a lot of fun working on this project! My implementation clearly lacks efficiency in terms of speed. Need to vectorize more of the math there. There was a suggestion in the paper regarding different learning rate schedulers instead of the way alpha is defined earlier. That is a possible avenue to explore in the future.

References

[1] E. Olson, J. Leonard, and S. Teller, "Fast Iterative Optimization of Pose Graphs with Poor Initial Estimates," *ResearchGate*, pp. 2262–2269, Jan 2006.

[2] M. Kaess, "Slides of SLAM:Least Squares," *CMU*.

[3] L. Carlone, R. Aragues, J. Castellanos, and B. Bona, "A fast and accurate approximation for planar pose graph optimization," *IJRR*, 2014.

[4] C. L., "g2oVStoro.pdf," Dec 2020. [Online]. Available: https://www.dropbox.com/s/uwwt3ni7uzdv1j7/g2oVStoro.pdf?dl=0