ROB 530 - HW7

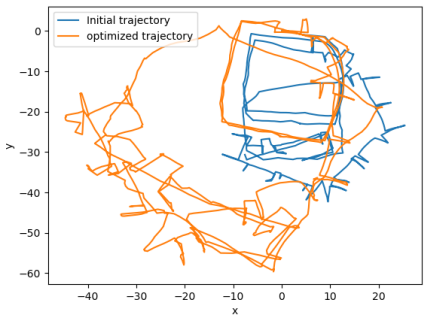
※ 50036744

Kuan-Ting Lee

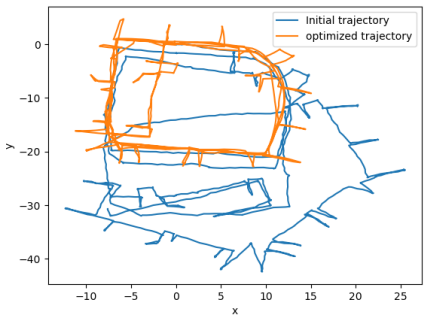
(1A) See code

(1B)

(w/o perturbation)



(w/ perturbation)



- Parameters : (1) prior Noise : $\begin{bmatrix} 0.3 & & \\ & 0.3 & \\ & & 0.1 \end{bmatrix}$

(2) GaussNewton_relativeErrorTol = 10^{-5}

(3) GaussNewton_maxIterations = 100

- Observation : When optimizing w/o perturbation, GaussNewton solver is easy to fall into local minimum. After adding SE2 perturbation of $\begin{bmatrix} 0.05 & & \\ & 0.05 & \\ & & 0.05 \end{bmatrix}$, The optimized path will become similar to the iSAM2's result.

- Graph construction:

```
NonlinearFactorGraph graph;
Values initialEstimate;
for vertex : initialEstimate.insert(i, Pose2(x, y, theta))
for edge : graph.emplace_shared<BetweenFactor<Pose2>>(i, j, Pose2(x,y,theta), noise)
```

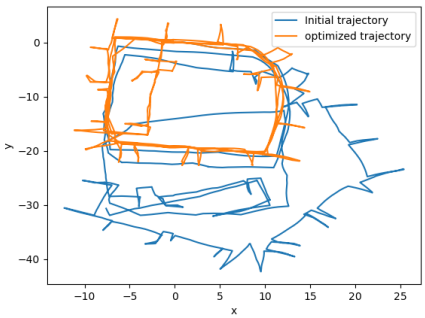
1c

- Parameters:

  iSAM2.relinearizeThreshold = 0.01

  iSAM2.relinearizeSkip = 1

  prior Noise = $\begin{bmatrix} 0.3 & & \\ & 0.3 & \\ & & 0.1 \end{bmatrix}$



- Observation

Compared with the Batch approach, iSAM2 seems not to fall into local minimum. My guess is that because Batch approach solves the entire system at one shot, the system is larger and more complicated. Therefore, it has more local minimum.

- Graph construction

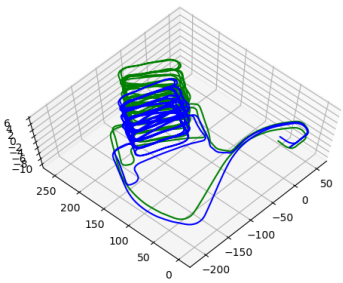
Algorithm 1 `incremental_solution_2d`(poses, edges)

Require: poses: a N × 4 array that each row is $pose = (id_p, x, y, \theta)$; edges: a M × 11 array that each row is $edge = (id_{e1}, id_{e2}, dx, dy, d\theta, info)$

```
isam ← gtsam.ISAM2()                                   ▷ Initialize isam solver
for every pose in poses do
    graph ← NonlinearFactorGraph                       ▷ Initialize the factor graph
    initialEstimate ← Values                           ▷ Initialize the initial estimation
    (id_p, x, y, θ) ← pose                             ▷ Extract information from the current pose
    if id_p == 0 then                                  ▷ The first pose
        priorNoise ← some noiseModel                   ▷ Use a predefined noise model
        graph.add(PriorFactorPose2(0, Pose2(x, y, θ), priorNoise))
        initialEstimate.insert(id_p, Pose2(x, y, θ))
    else                                               ▷ Not the first pose
        prevPose ← result.at(id_p − 1)                 ▷ Use last optimized pose
        initialEstimate.insert(id_p, prevPose)
        for every edge in edges do
            (id_e1, id_e2, dx, dy, dθ, info) ← edge    ▷ Extract information from the current edge
            if id_e2 == id_p then
                cov = construct_covariance(info)       ▷ Construct a covariance matrix from the
information vector.
                Model ← noiseModel.Gaussian.Covariance(cov)
                graph.add(BetweenFactorPose2(id_e1, id_e2, Pose2(dx, dy, dθ), Model))
            end if
        end for
    end if
    isam.update(graph, initialEstimate)
    result = isam.calculateEstimate()
end for
```

(2A) See code

(2B)



- Parameters : (1) prior Noise : $\begin{bmatrix} 0.3 & & & & & \\ & 0.3 & & & & \\ & & 0.3 & & & \\ & & & 0.1 & & \\ & & & & 0.1 & \\ & & & & & 0.1 \end{bmatrix}$

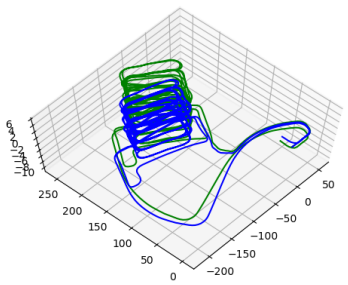
(2) GaussNewton_relativeErrorTol = 10^{-5}

(3) GaussNewton_maxIterations = 100

- Graph construction :

```
NonlinearFactorGraph graph;
Values initialEstimate;
for vertex : initialEstimate.insert(i, Pose3)
for edge : graph.emplace_shared<BetweenFactor<Pose3>>(i, j, Pose3, noise)
```

2C



- Parameters:

iSAM2.relinearizeThreshold = 0.01

iSAM2.relinearizeSkip = 1

$$\text{prior Noise} = \begin{bmatrix} 0.3 & & & & & \\ & 0.3 & & & & \\ & & 0.3 & & & \\ & & & 0.1 & & \\ & & & & 0.1 & \\ & & & & & 0.1 \end{bmatrix}$$

Graph construction:

Algorithm 1 `incremental_solution_2d`(poses, edges)

Require: poses: a N × 4 array that each row is $pose = (id_p, x, y, \theta)$; edges: a M × 11 array that each row is $edge = (id_{e1}, id_{e2}, dx, dy, d\theta, info)$

```
isam ← gtsam.ISAM2()                                   ▷ Initialize isam solver
for every pose in poses do
    graph ← NonlinearFactorGraph                       ▷ Initialize the factor graph
    initialEstimate ← Values                           ▷ Initialize the initial estimation
    (id_p, x, y, θ) ← pose                             ▷ Extract information from the current pose
    if id_p == 0 then                                  ▷ The first pose
        priorNoise ← some noiseModel                   ▷ Use a predefined noise model
        graph.add(PriorFactorPose2(0, Pose2(x, y, θ), priorNoise))
        initialEstimate.insert(id_p, Pose2(x, y, θ))
    else                                               ▷ Not the first pose
        prevPose ← result.at(id_p − 1)                 ▷ Use last optimized pose
        initialEstimate.insert(id_p, prevPose)
        for every edge in edges do
            (id_e1, id_e2, dx, dy, dθ, info) ← edge    ▷ Extract information from the current edge
            if id_e2 == id_p then
                cov = construct_covariance(info)       ▷ Construct a covariance matrix from the information vector.
                Model ← noiseModel.Gaussian.Covariance(cov)
                graph.add(BetweenFactorPose2(id_e1, id_e2, Pose2(dx, dy, dθ), Model))
            end if
        end for
    end if
    isam.update(graph, initialEstimate)
    result = isam.calculateEstimate()
end for
```