Principles of Robot Autonomy: Homework 4

Wei-Lin Pai
11/12/24

Other students worked with:
Time spent on homework: 6 hrs

Problem 1: Bundle Adjustment SLAM in 2D

1.1 Simple Factor Graph

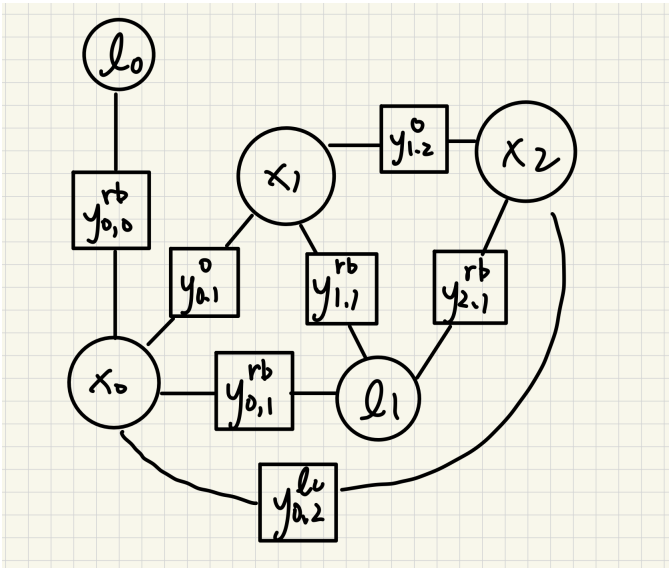

Figure 1: Factor graph

1.3 Factors in the Graph

There are a total of 60 factors in the graph.

1.4 Initial Estimation

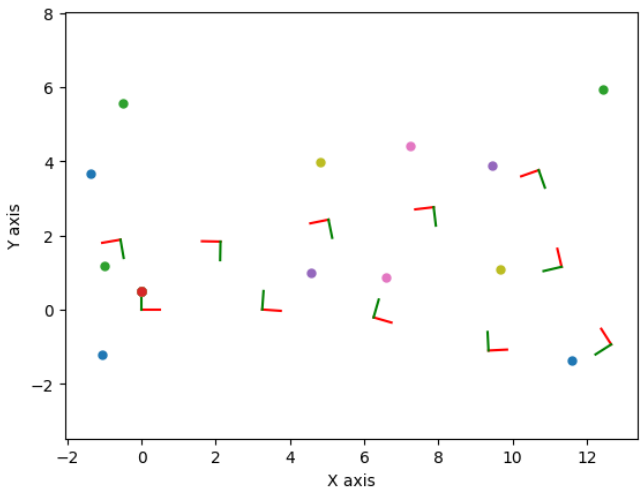

Figure 2: Initial estimation of the factor graph

1.5 SLAM Output

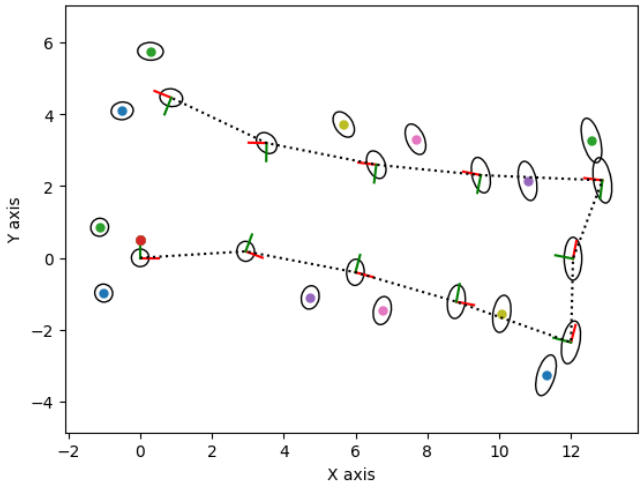

Figure 3: Slam output for robot history 5

1.6

It appears that some ellipses are more elongated than others, which might mean there is greater uncertainty in those poses along the elongated direction. This could be due to weaker constraints or fewer observations of that landmark.

1.7

Factor Graphs For robot history 3, the factor graph size is 32. For robot history 10, the factor graph have size 114.

1.8

SLAM Output The SLAM result of history 3 is as below:

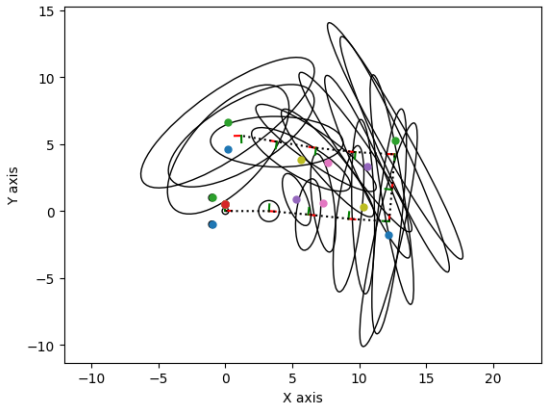

Figure 4: Slam output for robot history 3

It is obvious that this slam result have a relative higher covariance for each position estimation. This is because with shorter sensor range, it is more unlikely to get the precise data, or even getting the data. Hence, there is less factor to conduct slam in factor graph and the result will have high uncertainty and low confidence.
On the other hand, if we have higher sensor range, like history 10, we can have more sensor measurement data. With more factor in a factor graph, we have have a better result on SLAM optimization.

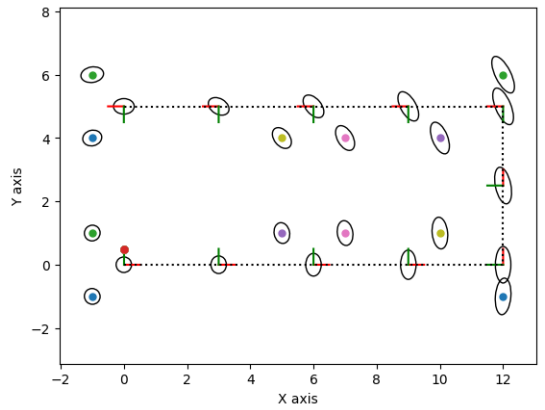

Figure 5: Slam output for robot history 10

Problem 2: Frontier Exploration

2.1 Explore

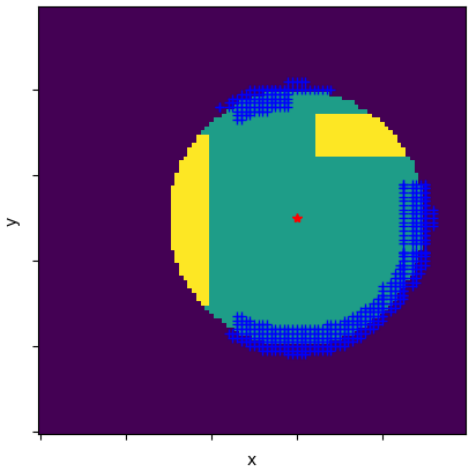

Figure 6: Frontier Exploration

2.2 Explore

Closest Distance value: 2.596

Appendix A: Code Submission

P1 GTSAM.ipynb
(Q4.b-c) Building the factor graph

```
# Re-import here so that you can rerun this cell to debug without rerunning
# everything
from gtsam.symbol_shorthand import L, X

# Create an empty nonlinear factor graph
### TODO (Q4.b) ###
graph = gtsam.NonlinearFactorGraph()
###

Xs = [X(i) for i in range(n_time_steps)]
Ls = [L(i) for i in range(n_landmarks)]

# Add a prior factor at the origin to set the origin of the SLAM problem
### TODO (Q4.b) ###
inital_pose = robot_noisy_df[["x", "y", "theta"]].iloc[0].to_numpy()
graph.add(gtsam.PriorFactorPose2(Xs[0],gtsam.Pose2(inital_pose),PRIOR_NOISE))
###

# Loop through the noisy data (i.e., `robot_noisy_df`)
# In the loop,
#    get the relative motion with pose_{i} - pose_{i - 1}
#    the measurements are in the noisy data
# Don't forget the loop closure constraint
### TODO (Q4.c) ####
for i in range(n_time_steps):
    for j in range(n_landmarks):
        if not np.isnan(robot_noisy_df.iloc[i][f"bearing_{j}"]) and not
            np.isnan(robot_noisy_df.iloc[i][f"range_{j}"]):
            bearing = gtsam.Rot2.fromDegrees(robot_noisy_df.iloc[i][f"bearing_{j}"])
            range_ = robot_noisy_df.iloc[i][f"range_{j}"]
            graph.add(gtsam.BearingRangeFactor2D(Xs[i],Ls[j],bearing,range_,MEASUREMENT_NOISE))
    if i > 0:
      X_prev = gtsam.Pose2(robot_noisy_df.iloc[i-1][['x']],
      robot_noisy_df.iloc[i-1][['y']],
      robot_noisy_df.iloc[i-1][['theta']])
      X_curr = gtsam.Pose2(robot_noisy_df.iloc[i][['x']],
      robot_noisy_df.iloc[i][['y']],
      robot_noisy_df.iloc[i][['theta']])
      rel_pose = X_prev.between(X_curr)
      graph.add(gtsam.BetweenFactorPose2(Xs[i-1],Xs[i],rel_pose,ODOMETRY_NOISE))

loop_closure_offset = gtsam.Pose2(0.0,5.0,np.pi)
graph.add(gtsam.BetweenFactorPose2(Xs[2], Xs[8], loop_closure_offset, ODOMETRY_NOISE))
###

# Print the factor graph to see all the nodes
### TODO (Q4.c) ####
```

```
graph
###
```

(Q4.d) Initialize the optimization problem

```
# Set-up a values data structure for the initial estimate
### TODO (Q4.d) ###
initial_estimate = gtsam.Values()
###

# Set the initial poses from the noisy odometry alone
# Hint, you already have this in `robot_noisy_df`
### TODO (Q4.d) ###
for i in range(n_time_steps):
    x = robot_noisy_df.iloc[i]['x']
    y = robot_noisy_df.iloc[i]['y']
    theta = robot_noisy_df.iloc[i]['theta']
    # Create Pose2 with individual values
    initial_estimate.insert(Xs[i], gtsam.Pose2(x, y, theta))

###

# Sample random values for the initial landmark positions
# In reality, you would have to estimate these from odometry,
# but to not over-complicate the problem, just use noisy
# ground-truth.
# In reality, the initialization is very important for the
# graph optimization to converge to a good solution!!
l_init_vec = [
    (-1, -1),
    (-1, 1),
    (5, 1),
    (7, 1),
    (10, 1),
    (12, -1),
    (12, 6),
    (10, 4),
    (7, 4),
    (5, 4),
    (-1, 4),
    (-1, 6)
]

for l_ind, L in enumerate(Ls):
    l_hat = l_init_vec[l_ind]
    point_init = (np.random.normal(l_hat[0], ODOMETRY_NOISE_NUMPY[0]),
                  np.random.normal(l_hat[1], ODOMETRY_NOISE_NUMPY[0]))
    # Add the initial estimates of the landmarks
    ### TODO (Q4.d) ###
    initial_estimate.insert(L,gtsam.Point2(point_init[0],point_init[1]))
    ###
```

```
# Print the initial estimates to verify
### TODO (Q4.d) ###
initial_estimate
###
```

(Q4.e) Graph Optimization

```
### TODO (Q4.e) ###
marginals = gtsam.Marginals(graph, result)
for x_ind, x_key in enumerate(Xs):
    print(marginals.marginalCovariance(x_key))
for l_ind, l_key in enumerate(Ls):
    print(marginals.marginalCovariance(l_key))
```

HW4 P2 explorer.ipynb

```
    ############################ Code starts here ############################
    probs = occupancy.probs
    window = np.ones((window_size, window_size))

    unknown_mask = (probs == -1).astype(float)
    occupied_mask = (probs >= 0.5).astype(float)


    unknown_count = convolve2d(unknown_mask, window, mode='same', fillvalue = 1)
    occupied_count = convolve2d(occupied_mask, window, mode='same')
    unoccupied_count = window_size**2 - occupied_count - unknown_count

    unknown_percentage = unknown_count / (window_size**2)
    occupied_percentage = occupied_count / (window_size**2)
    unoccupied_percentage = unoccupied_count / (window_size**2)

    condition1 = unknown_percentage >= 0.2
    condition2 = occupied_count == 0
    condition3 = unoccupied_percentage >= 0.3

    x_,y_ = np.meshgrid(np.arange(occupancy.size_xy[0]), np.arange(occupancy.size_xy[1]))
    frontier_grid = np.stack([x_, y_],axis = -1)

    valid_pt = condition1 & condition2 & condition3
    Frontier = frontier_grid[valid_pt]
    frontier_states = occupancy.grid2state(Frontier)

    distances = np.linalg.norm(frontier_states - current_state, axis=1)
    min_distance = np.min(distances)
    print(min_distance)

    ############################ Code ends here ############################
    return frontier_states
```