16-833: Robot Localization and Mapping, Spring 2025

Homework 3 - Linear and Nonlinear SLAM Solvers

Due: Friday Mar 28, 11:59pm, 2025

Your homework should be submitted as a **typeset PDF file** along with a folder including **code only (no data)**. The PDF and code must be submitted on **Gradescope**. If you have questions, post them on Piazza or come to office hours. Please do not post solutions or codes on Piazza. This homework must be done **individually**, and plagiarism will be taken seriously. You are free to discuss and troubleshoot with others, but the code and writeup must be your own. Note that you should list the name and Andrew ID of each student you have discussed with on the first page of your PDF file.

1 2D Linear SLAM

In this problem you will implement your own 2D linear SLAM solver in `linear.py`. Data are provided in `2d_linear.npz`, `2d_linear_loop.npz` with loaders. Everything you need to know to complete this problem was covered in class, so please refer to your notes and lecture slides for guidance.

We will be using the least squares formulation of the SLAM problem, which was presented in class:

$$\begin{aligned} x^* &= \arg\min_x \Sigma \left\| h_i\left(x\right) - z_i \right\|^2_{\mathbf{\Sigma}_i} \\ &\vdots \\ &\approx \arg\min_x \left\| \mathbf{A}x - b \right\|^2, \end{aligned} \tag{1}$$

where z_i is the i-th measurement, $h_i\left(x\right)$ is the corresponding prediction function, and $\|a\|^2_{\mathbf{\Sigma}}$ denotes the squared Mahalanobis distance: $a^T\mathbf{\Sigma}^{-1}a$.

In this problem, the state vector x is comprised of the trajectory of robot positions and the landmark positions. Both positions are simply (x, y) coordinates. For a sanity check, we visualize the ground truth trajectory and landmarks in the beginning.

There are two types of measurements: odometry and landmark measurements. Odometry measurements give a relative $(\Delta x, \Delta y)$ displacement from the previous position to the next position (in global frame). Landmark measurements also give a relative displacement $(\Delta x, \Delta y)$ from the robot position to the landmark (also in global frame).

1.1 Measurement function (10 points)

Given robot poses $\mathbf{r}^t = [r_x^t, r_y^t]^\top$ and $\mathbf{r}^{t+1} = [r_x^{t+1}, r_y^{t+1}]^\top$ at time t and $t+1$, write out the measurement function and its Jacobian. (5 points)

$$h_o(\mathbf{r}^t,\ \mathbf{r}^{t+1}) : \mathbb{R}^4 \to \mathbb{R}^2,$$
$$H_o(\mathbf{r}^t,\ \mathbf{r}^{t+1}) : \mathbb{R}^4 \to \mathbb{R}^{2\times 4}.$$

Similarly, given the robot pose $\mathbf{r}^t = [r_x^t,\ r_y^t]^\top$ at time t and the k-th landmark $\mathbf{l}^k = [l_x^k,\ l_y^k]^\top$. (5 points)

$$h_l(\mathbf{r}^t, \mathbf{l}^k) : \mathbb{R}^4 \to \mathbb{R}^2,$$
$$H_l(\mathbf{r}^t, \mathbf{l}^k) : \mathbb{R}^4 \to \mathbb{R}^{2\times 4}.$$

1.2 Build a linear system (15 points)

Use the derivation above, please complete the function `create_linear_system` to construct the linear system as described in Eq. 1. (15 points)

- Note in this setup, you will be filling the blocks in the large linear system that is aimed at batch optimizing the large state vector stacking all the robot and landmark positions. Please carefully select indices for both measurements and states when you fill in Jacobians.
- Use `int` to convert `observation`[:, 0] and `obsevation`[:, 1] into pose and landmark indices respectively.
- In addition, you will have to add a prior to the first robot pose, otherwise the system will be underconstrained and the state will be subject to an arbitrary global transformation.
- Please refer to the function document for detailed instructions.

1.3 Solvers (20 points)

Given the data and the linear system, you are now ready to solve the 2D linear SLAM problem. You are required to implement 5 solvers to solve $Ax = b$ where A is a sparse non-square matrix.

1. `pinv`: Use pseudo inverse to solve the system. You may only use `scipy.sparse.linalg.inv` and matrix multiplication in this function. Return x and a placeholder None. (5 points)
2. `lu`: Use LU factorization (Cholesky is one variant of LU) to solve the system. You may use `scipy.sparse.linalg.splu` to factorize the relevant matrices, and use the resulting `SuperLU`'s `solve` method to obtain the final result. Specify ordering `permc_spec` with `NATURAL` in `splu`. Return both x and U. (5 points)
3. `qr`: Use QR factorization to solve the system. You may use `sparseqr.rz` to obtain `z, R, E, rank` from A, b, where R, z are the factors used for efficiently solving $||Ax-b||^2 = ||Rx-z||^2 + ||e||^2$ (for details please check the lecture note *Sparse Least Squares*). You may then use `scipy.sparse.linalg.spsolve_triangular` to get the solution. Specify ordering `permc_spec` with `NATURAL` in `rz`. You may NOT directly use `sparseqr.solve`. Return both x and R. (10 points)

We provide you with a default solver for sanity check. After obtaining the state vector x, You may decode the state to trajectory and landmarks, and visualize your results using functions `devectorize_state` and `plot_traj_and_landmarks`. Check if they match the ground truth before you proceed to the next step.

1.4 Exploit sparsity (30 points + 10 points)

Now we want to exploit sparsity in the linear system in QR and LU factorizations.

1. `lu_cholmod`. Change the ordering from the default `NATURAL` to `COLAMD` (Column approximate minimum degree permutation) and return x, U. (5 points)

2. (Bonus) Instead of LU's built-in solver, write your own forward/backward substitution to compute x. Note because of reordering (permutation), you need to manipulate both rows and columns. Please check online documents for more details. (10 points)

3. `qr_cholmod`. Change the ordering from the default `NATURAL` to `COLAMD` (Column approximate minimum degree permutation) and return x, R. Note now you have to use E from `sparseqr.rz`.
   `sparseqr.permutation_vector_to_matrix` can be useful for permutation. (5 points)

4. Now proceed with `2d_linear.npz`, visualize the trajectory and landmarks, and report the efficiency of your method in terms of run time. Attach the visualization and analysis of corresponding factor for `qr`, `qr_colamd`, `lu`, `lu_colamd`. What are your observations and their potential reasons (general comparison between QR, LU, and their reordered version)? (10 points)

5. Similarly, process `2d_linear_loop.npz`. Are there differences in efficiency comparing to `2d_linear.npz`? Write down your observations and reasoning. (10 points)

2 2D Nonlinear SLAM

Now you are going to extend the linear SLAM to a nonlinear version in `nonlinear.py`.

The problem set-up is exactly the same as in the linear problem, except we introduce a nonlinear measurement function that returns a bearing angle θ and range d (in robot's body frame, notice we assume that this robot always perfectly facing the x-direction of the global frame), which together describe the vector from the robot to the landmark:

$$h_l\left(\mathbf{r}^t, \mathbf{l}^k\right) = \begin{bmatrix} \operatorname{atan2}\left(l_y^k - r_y^t, l_x^k - r_x^t\right) \\ \left(\left(l_x^k - r_x^t\right)^2 + \left(l_y^k - r_y^t\right)^2\right)^{\frac{1}{2}} \end{bmatrix} = \begin{bmatrix} \theta \\ d \end{bmatrix}. \tag{2}$$

2.1 Measurement function (10 points)

In your nonlinear algorithm, you'll need to predict measurements based on the current state estimate.

1. Fill in the functions `odometry_estimation`, and `bearing_range_estimation` with corresponding measurement functions. The odometry measurement function is the same linear function we used in the linear SLAM algorithm, while the landmark measurement function is the new nonlinear function introduced in Eq. 2. Please carefully check indices and offsets in the state vector. (5 points)

2. Derive the jacobian of the nonlinear landmark function in your writeup

$$H_l(\mathbf{r}^t, \mathbf{l}^k) : \mathbb{R}^4 \to \mathbb{R}^{2\times 4},$$

   and implement the function `compute_meas_obs_jacobian` to calculate the jacobian at the provided linearization point. (5 points)

2.2 Build a linear system (15 points)

Use the derivation above, implement `create_linear_system`, now in `nonlinear.py`, to generate the linear system A and b at the current linearization point. (15 points) In addition to the notes for the linear case, please remember to

- Use the provided initialization of x as the linearization point.
- Error per observation is the *difference* of measurements and estimates because of linearization.
- Use `warp2pi` to normalize the difference of angles.

2.3 Solver (10 points)

Process `2d_nonlinear.npz`. Select one solver you have implemented, and visualize the trajectory and landmarks before and after optimization. Briefly summarize the differences between the optimization process of the linear and the non-linear SLAM problems. (10 points)

3 Code Submission

Instructions:

- Use `conda` to create an environment and run `./install_deps.sh` to install dependencies. If you encounter failures, please install SuiteSparse to your system (usually already installed), see dependencies for sparseqr.
- Read documents and source code for the packages (`scipy.sparse.linalg` and `sparseqr`). It is a good exercise to learn to use libraries you are not familiar with.
- Use command line arguments. For instance, you can run
  python linear.py ../data/2d_linear.npz –method pinv qr qr_colamd lu lu_colamd
  to check the results of all methods altogether on the `2d_linear` case.

Please upload your code to gradescope excluding the `data` folder.