Problem Set 4

Due May 10, 2024 at 11:59 PM Pacific

Turn in your problem set by Gradescope. Include any code you write in your solutions as well. Mark down how much time you spent on the homework set.

1. **Kalman Filter update from an Optimization Perspective:** In this problem you will derive the update step of the Kalman filter as the solution to an optimization problem. Consider the Maximum A Posteriori (MAP) optimization problem, $x_t^{\mathrm{MAP}} = \arg\min_{x_t} J(x_t)$. Also, consider a previously computed Gaussian prediction distribution over the state at time t give measurements up to time $t-1$

$$p(x_t \mid y_{1:t}) = \mathcal{N}(\mu_{t|t-1}, \Sigma_{t|t-1}),$$

where $\mu_{t|t-1}$ and $\Sigma_{t|t-1}$ are known. Consider a linear-Gaussian measurement model $Y_t = C_t X_t + V_t$, where $V_t \sim \mathcal{N}(0, R_t)$. A sensible objective function to estimate x_t given the new measurement y_t is to minimize both the squared error of the measurement and the state. This formulation becomes a convex quadratic in x_t,

$$J(x_t) = \frac{1}{2}\|y_t - C_t x_t\|^2_{R_t^{-1}} + \frac{1}{2}\|x_t - \mu_{t|t-1}\|^2_{\Sigma^{-1}_{t|t-1}}.$$

   (a) Analytically find x_t^{MAP} by setting the gradient of the objective function to zero $\frac{\partial J(x_t)}{\partial x_t}\,|_{x_t = x_t^{\mathrm{MAP}}} = 0$ and solving for x_t^{MAP}. Show that your MAP estimate is $x_{t|t} = x_{t|t-1} + K_t(y_t - C_t\mu_{t|t-1})$ where $K_t = \Sigma_{t|t-1}C_t^T\left[C_t\Sigma_{t|t-1}C_t^T + R_t\right]^{-1}$. This is the update step of the Kalman filter.

   (b) Analytically find the Hessian, $H = \frac{\partial^2 J(x_t)}{\partial x_t^2}\,|_{x_t = x_t^{\mathrm{MAP}}}$. Show that the Hessian has the form $H = C_t^T R^{-1} C_t + \Sigma^{-1}_{t|t-1}$.

   (c) Invert H and simplify to get $\Sigma_{t|t} = H^{-1} = \Sigma_{t|t-1} - KC_t\Sigma_{t|t-1}$. This is the covariance update for the Kalman filter.

Hints: Matrix inverse lemmas will come in handy for this problem.

2. **Point Cloud Alignment:** Consider two point clouds $(r_1, \ldots, r_N)$ and $(\hat{r}_1, \ldots, \hat{r}_N)$, where $r_i, \hat{r}_i \in \mathbb{R}^3$ for all i. We would like to translate and rotate the second point cloud so that it best matches with the first. Assume both clouds have the same number of points and we know which point in the first cloud corresponds to which point in the second (these assumptions greatly simplify the problem and are the focus of current research). We would like to find a rotation R and translation τ that minimizes the following loss

$$J(R, \tau) = \frac{1}{N}\sum_{i=1}^{N} ||R\hat{r}_i + \tau - r_i||_2^2$$

subject to

$$RR^T = I$$

This is a non-convex optimization problem due to the orthogonality constraint $RR^T = I$. However, we can still solve this problem optimally. Let's seek to solve this optimization problem through two ways.

   (a) Solve for the optimal $\tau*$ as a function of $R, \hat{r}_i, r_i$. This should just be the solution to an ordinary least-squares problem. In terms of properties of the two point clouds, what does τ physically represent?

   (b) Solve for the rotation R^* by treating it as a linear optimization problem. In the process, also show that the rotation matrix you solved is globally optimal (i.e. there doesn't exist another rotation matrix that can have a lower loss).
   Hint: The Singular Value Decomposition $A = U\Lambda V^T$, where U, V are orthogonal and Λ is diagonal, may come in handy. All matrices have an SVD decomposition.

(c) Show that your answer in part (b) is equivalent to taking the SVD of a simpler 3×3 matrix $H = \sum_j^N (\hat{r} - \hat{r}_c)(r - r_c)^T$.

Remark: Part (c) is important because you will need a fast implementation of getting the rotation for Problem 3.

(d) Align two point clouds, one the ground-truth `ground-truth.ply` and the other, a rotated, translated point cloud with additive noise `transformed-associated.ply`. What's the loss $J(R_t^*, \tau_t^*)$?

(e) What's the SE(3) error between your optimized pose and the ground-truth pose (located in `gt-pose.csv`)? The CSV file stores the pose as a quaternion (x, y, z, w) and the translation (px, py, pz) in that order. Report both the rotational error (in degrees), and the translational error.

(f) Is it possible that the program is optimally solved, but the R matrix of the solution does not correspond to a physical rotation? In which case, what does R physically do to the point cloud? (Aside: even when R is not a rotation matrix, there are methods to still extract a valid rotation).

3. **Iterative Closest Point (ICP):** Let's extend the optimization infrastructure we developed in Problem 2 to problems where associations are unknown. The most common method is Iterative Closest Point (ICP), which iteratively performs a naive data associate followed by a point cloud alignment, and repeats until convergence. ICP starts with source point cloud, and returns a rotation and translation to align it with a target point cloud. The steps are as follows:

   - Step 1: For every point x_i in the source point cloud, find the closest point y_j in the target point cloud and form an association (x_i, y_j). It is fine if multiple x_i points are associated to the same y_j point.
   - Step 2: Find R and τ using the method from Problem 2.
   - Step 3: Transform the source point cloud by this optimized transformation.
   - Step 4: Go back to Step 1. Stop if the difference in losses between consecutive iteration steps falls below ϵ.

   Note that you already have all the ingredients to implement ICP. The closest point query you achieved using a KD-tree in Homework 3. The computation of the relative transformation was done in Problem 2. Use $\epsilon = 10^{-5}, \alpha = 0.2$.

   (a) Using the point cloud `transformed-nonassociatedX.ply` as the source point cloud(s) and `ground-truth.ply` as the target, perform ICP to align the two point clouds for $X = 1, 2, 3$. Plot the Mean Squared Error (the loss in Problem 2) and the SE(3) error (using the ground-truth pose provided in the file `gt-pose-nonassociatedX.csv`) over the iterations.

   (b) This basic form of ICP may not be particularly robust. However, our point cloud data typically contains more information than just locations of points. One such example is the color of each point, the projection of an RGB image to 3D using depth measurements. Note that if you visualize the two point clouds, they are **colored** point clouds. Therefore, let's add an additional step (Step 1.5) where after we form an association, we throw out any associations in which the color of x_i and y_j do not match $||c(x_i) - c(y_j)|| > \alpha$. The act of throwing out potential associations is called outlier rejection. Implement this colored ICP for the same point clouds as in (a). Plot the MSE and the SE(3) error on the same plots as in (a). In other words, you should have 3 plots: MSE, Rotation Error, and Translation Error. On each plot should be 6 curves: 3 per method, with each curve corresponding to a particular source point cloud `transformed-nonassociatedX.ply` for $X = 1, 2, 3$.

4. **Bundle Adjustment SLAM in 2D:** In this problem you will use a standard open source SLAM package, GTSAM, to perform a bundle adjustment SLAM. (Recall, bundle adjustment SLAM solves for both landmark locations and robot poses at the same time. Alternatively, Pose Graph Optimization (PGO) SLAM pre-processes raw sensor measurements in a "front end", and optimizes only robot poses in a "back end.") We will use the Python interface to the GTSAM library, which is implemented in C++. We use a sparse 2D problem setup to ensure quick optimization in Google Colab. If you choose to consider dense or 3D SLAM for your final project, you will likely not want to use Google Colab, and you would likely use a PGO style SLAM.

(a) Draw a simple factor graph by hand or with a computer program, such as PowerPoint, that meets the following criteria.

- Three pose states x_0, x_1, and x_2.
- Two landmarks, ℓ_0 and ℓ_1, marked by a 2D position.
- Four range and bearing measurements as follows:
  - $y^{rb}_{0,0}$: between pose x_0 and landmark ℓ_0
  - $y^{rb}_{0,1}$: between pose x_0 and landmark ℓ_1
  - $y^{rb}_{1,1}$: between pose x_1 and landmark ℓ_1
  - $y^{rb}_{2,1}$: between pose x_2 and landmark ℓ_1
- Odometry measurements $y^{o}_{i,i+1}$ between successive poses.
- Loop closure constraint as odometry-like measurement $y^{lc}_{0,2}$ between pose states x_0 and x_2.

We now consider a more complex scenario. We will use the robot trajectory and landmarks in Figure 1. You will see three data files on Canvas. In the file `robot_history_5.csv`, we provide the ground truth history of the robot pose and measurements for a maximum sensor range of 5 meters. We will add noise and reconstruct the robot pose and landmark locations. The following steps will guide you in constructing the factor graph in GTSAM.

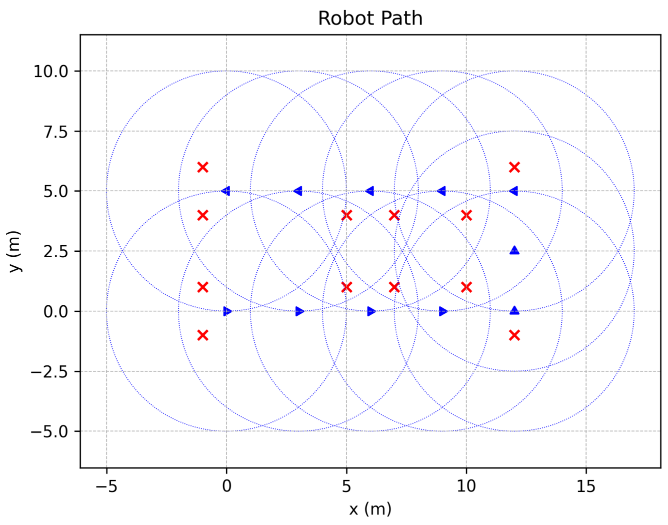

Figure 1: Robot Ground Truth Path. The landmarks are indicated as a red '×', the robot is a blue triangle oriented in the direction of travel, and the dotted blue circles indicate the maximum sensor range.

Please refer to the provided Google Colab at the link below (using your Stanford login), though you do not need to use Google Colab if you prefer to install GTSAM locally.

`https://colab.research.google.com/drive/1tJqy1jNUeGC2V235yTdWqyOElQdIclMp?usp=sharing`

(b) Verify that you can pip install GTSAM. On Google Colab, you can run `!pip install gtsam` at the top. Please read through the provided code to load and process the files. Instantiate the factor graph with:

```
graph = gtsam.NonlinearFactorGraph()
```

This is an empty factor graph. To *add* a factor to the factor graph, we will call `graph.add(...)`. To simplify the notation, GTSAM offers shorthand variables `X` and `L` for state and landmark, respectively, to organize the SLAM factors. In the provided Google Colab, we have initialized lists for `X` and `L`. The first factor we want to add is the prior factor to initialize the map at the origin. Fill in the required code based on the factor description below.

`gtsam.PriorFactorPose2(X_start, pose, noise)`

For `X_start`, use the zero-th element of the `X` list. For `pose`, use the `gtsam.Pose2(x, y, theta)` function specifying the origin and zero angle (i.e., aligned with the x-axis).

(c) Iterate through the provided data (`Pandas` is recommended). Use the factor type below to add the relative motion factors (i.e., from the odometry or relative state between successive states).

`gtsam.BetweenFactorPose2(X_prev, X_curr, rel_pose, motion_noise)`

Use the factor type below to add the bearing and range factors (i.e., from the sensor model) per each landmark in view at the given time step.

`gtsam.BearingRangeFactor2D(X_curr, L_curr, bearing, range, noise`

For the bearing measurement, use `gtsam.Rot2.fromDegrees(bearing_angle)`. Lastly, we will use the `BetweenFactorPose2` method to add a loop closure constraint between the second and eighth elements (zero-indexed) of the `X` list with position offset of 0 horizontally, 5 vertically, and 180 degrees in rotation. How many factors are in your factor graph (print `graph` to verify)?

(d) We will initialize the values from the motion alone. Instantiate the initial estimate with

`initial_estimate = gtsam.Values()`

You can input an initial estimate with `initial_estimate.insert(X_or_L_var, pose_or_point)`. Be sure to use `gtsam.Pose2(x, y, theta)` for the pose and `gtsam.Point2(x, y)` for the landmark. Plot the initial estimates for `X` and `L` using the functions below:

`gtsam_plot.plot_pose2(...)` and `gtsam_plot.plot_point2(...)`.

Include the plot in your write-up.

(e) Run the Levenberg Marquardt in GTSAM as listed in the Google Colab. GTSAM will provide estimates of `X` and `L` through the optimizer and will provide the marginal Gaussian distributions in `gtsam.Marginals`. Plot the optimized estimates for `X` and `L` along with the 2D Gaussian Confidence Ellipse (1-sigma). Include the plot in your write-up.

(f) What do you notice about the orientation of the solution and the shape and orientation of the confidence ellipses? Why does this occur?

(g) Switch out `robot_history_5.csv` for `robot_history_3.csv` (maximum sensor range of 3 meters) and `robot_history_10.csv` (maximum sensor range of 10 meters). Using the same initialization for `X` and `L`, reconstruct the factor graphs. What are the factor graph sizes in each case?

Hint: You should only have to change the file input. All the other functions you wrote should still be applicable!

(h) Plot the optimized estimates for `X` and `L` along with the 2D Gaussian Confidence Ellipse (1-sigma) in each case. What do you notice about the final results at different sensor ranges? Explain your result with the connectivity of the resulting factor graphs.