Perception in Robotics
Term 3, 2020. PS4

Gonzalo Ferrer
Skoltech

March 13, 2020

This problem set is a single task comprising 10% of your course grade, and it is to be done individually. You are encouraged to talk at the conceptual level with other students, discuss the equations and even the results, but you may not show/share/copy any non-trivial code.

Submission Instructions

Your assignment must be received by 11:59p on Friday, March 20th. You are to upload your assignment directly to the Canvas website as two attachments:

1. A PDF with the written portion of your document, solving the tasks proposed below. Scanned versions of hand-written documents, converted to PDFs, are perfectly acceptable. No other formats (e.g., .doc) are acceptable. Your PDF file should adhere to the following naming convention: `alincoln_ps4.pdf`.
2. A file containing code to solve this PS4. You will be using COLAB and the mrob library for $SE(3)$, so it will be required to export your notebook to `.ipynb` or `.py`

   `alincoln_ps4.ipynb`

Homework received after 11:59p is considered late and will be penalized as per the course policy. The ultimate timestamp authority is the one assigned to your upload by Canvas.

Task 1: Point Cloud Alignment by Gradient Descent

In this first task you should implement a solution to the point cloud alignment, that is, finding the RBT $T \in SE(3)$ that minimizes the following cost function:

$$C(T) = \frac{1}{2}\sum_{i=0}^{N-1} ||Tx_i - y_i||^2_{\Sigma_i} = \frac{1}{2}\sum_{i=0}^{N-1} ||r_i(T)||^2_{\Sigma_i} \to \min_T .$$

The function $C(T)$ resembles the derivation obtained for SAM, although all the residuals $r_i(T)$ are *observing* the same transformation. Also, note the difference w.r.t. the seminar, where the sign of the residual has been changed.

Data regarding X and Y is provided as well as the covariance for each observation Σ_i. This covariance is different for each point transformed and should be taken into account properly. Data association is given, so the same indexes in all arrays correspond to the same points observed from different frames.

A. (10 pts) Create a function that plots the PC Y and the transformed PC $T \cdot X$. Also plot a segment between each pair of points. Show a plot for the initial transformation estimation $T^{[0]} = I$.

   Hint: you might want to use mrob.SE3 and SE3.transform_array(), whose input is a matrix $3 \times N$

B. (10 pts) Create a function that outputs the current value of $C(T)$, that is, the alignment error.

C. (10 pts) Create a function that calculates the gradient $\nabla_T C = \sum_{i=0}^{N-1} r_i^\top \Sigma_i^{-1} \frac{\partial r_i}{\partial \Delta\xi}$, knowing that

$$\frac{\partial r_i}{\partial \Delta\xi} = \big[-(T \cdot p)^\wedge | I \big].$$

   Hint: In the first iteration $T^{[0]} = I$, the magnitude of the gradient should be around $6e6$.

D. (10 pts) Create a function that updates the current value of the transformation given the gradient just calculated. For that, we will use *Gradient Descent* with update sizes of $\alpha = 10^{-6}[0.1, 0.1, 0.1, 1, 1, 1]$. To test this function, provide the initial condition $T^{[0]} = I$ and calculate the error before and after the update. You can also plot the point clouds before and after.

$$T^{k+1} = \text{Exp}(-\alpha \nabla_T C) \cdot T^k$$

E. (20 pts) Optimize the function $C(T)$ until convergence. For that, we will define a convergence criterion to be the difference in errors $< 1e - 4$. Plot the results of the error w.r.t. the number of iterations. Show also a plot of the point clouds aligned after convergence. Comment on the results.

   Hint: for debugging you might want to use the plot function.

Task 2: Point Cloud Alignment by Gauss-Newton

The Newton method for optimization is a second order optimization method, in the sense that it uses the second derivative of the cost function, the Hessian, to update the current solution point:

$$x^{k+1} = x^k - \alpha^k (\nabla_x^2 f(x^k))^{-1} \nabla_x f(x^k)$$

The Newton method presents a better convergence rate than gradient decent, and the intuition is that instead of guessing (or line-searching) a value for the step-size, the Hessian automatically provides a good estimation. Calculating the Hessian can be prohibitively expensive, that is why one can approximate it by different means. One popular approximation, which only works for squared residuals is the Gauss-Newton method.

If each of the residuals can be rewritten in the following form

$$C(T) = \frac{1}{2} \sum_{i=0}^{N-1} r_i^\top \cdot \Sigma_i^{-1} \cdot r_i$$

its derivative is, as presented before

$$\nabla_T C = \sum_{i=0}^{N-1} r_i^\top \Sigma_i^{-1} \frac{\partial r_i}{\partial \Delta\xi}$$

and its second derivative

$$\nabla_T^2 C = \sum_{i=0}^{N-1} \frac{\partial r_i}{\partial \Delta\xi}^\top \Sigma_i^{-1} \frac{\partial r_i}{\partial \Delta\xi} + r_i^\top \Sigma_i^{-1} \frac{\partial^2 r_i}{\partial \Delta\xi^2}.$$

The Gauss-Newton method proposes to ignore the right hand part of the Hessian by:

$$\nabla_T^2 C \simeq \sum_{i=0}^{N-1} \frac{\partial r_i}{\partial \Delta \xi}^\top \Sigma_i^{-1} \frac{\partial r_i}{\partial \Delta \xi},$$

which is a good approximation as long as the values of the residual are close to zero (which is our aim). Also note that this Hessian is for each element of i, that means, that the overall Hessian is obtained after summing N times each component.

The final result, will be an iterative update of our transformation T by

$$T^{k+1} = \text{Exp}(-\alpha(\nabla_T^2 C)^{-1} \nabla_T C) \cdot T^k$$

In this task you will implement Gauss-Newton (GN)

A. (10 pts) Create a function, based on the previous function from Task 1.C, that returns the gradient vector and the Hessian matrix.

B. (10 pts) Create a function, based on function from Task 1.D that updates T by using $\alpha = 1$.

C. (20 pts) Optimize the function $C(T)$ until convergence using GN. Plot the error w.r.t. iterations. Comment on the results and compare with the result obtained in Task 1.