CS 4610/5335: Robotic Science and Systems (Spring 2022) — Lawson L.S. Wong
Northeastern University — Due April 15, 2022

Exercise 5: Optimization and Perception

Please remember the following policies:

- Exercises are due at **11:59 PM** Boston time (ET).
- **Submissions should be made electronically on Canvas, as a single .pdf file for written parts, and as a single .zip file for code.** Check your submission and make sure you have included all files! You can make as many submissions as you wish, but only the latest one will be considered, and late days will be computed based on the latest submission.
- Each exercise may be handed in up to two days late (24-hour period), penalized by 5% per day. Submissions later than this will not be accepted. There is no limit on the total number of late days used over the course of the semester.
- Written solutions may be handwritten or typeset. For the former, please ensure handwriting is legible. If you write your answers on paper and submit images of them, that is fine, but please put and order them correctly in a single .pdf file. One way to do this is putting them in a Word document and saving as a PDF file.
- For programming questions, we recommend leaving sufficient comments to explain the logic of your solution.
- Some questions are intended for CS 5335 students only. CS 4610 students may complete these for extra credit.
- You are welcome to discuss these problems with other students in the class, but you must understand and write up the solution yourself, *and* indicate who you discussed with (if any).
- Contact the teaching staff if there are *extenuating* circumstances.

If you are taking the undergraduate version of the course (CS 4610), you may go directly to V2 and V3 (starting on page 3). However, observe that V1 and OPT have similar point values respectively; you may choose to do one or both of those instead if you prefer. We expect a completion of 8 points, but you are welcome to exceed that for extra credit.
All students (both CS 4610 and CS 5335) may also continue completing the Ex4/Ex5 extra credit questions until 4/30. Submit answers on Canvas in the Ex4/Ex5 extra credit assignment.

OPT. **[CS 5335 only.] 5 points.** In this question, you will use trajectory optimization (direct transcription) to find trajectories that satisfy various constraints while minimizing costs.

Consider the 2-D displacement system from Ex3 Q1(b).

The robot is situated on a 2-D plane, with position $\mathbf{x} = \begin{bmatrix} x_x \\ x_y \end{bmatrix} \in \mathbb{R}^2$.

The robot is commanded via a displacement in each axis independently, $\mathbf{u} = \begin{bmatrix} u_x \\ u_y \end{bmatrix} \in \mathbb{R}^2$.

The underlying system dynamics are linear with zero process noise: $f(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{x}_t + \mathbf{u_t}$.

The control cost at each time step is $\mathbf{u}_t^\intercal R \mathbf{u}_t$, where $R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ (i.e., the sum of squares $u_{x,t}^2 + u_{y,t}^2$).

(a) Suppose the robot starts at $\mathbf{x}_{\text{start}} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$ and needs to get to $\mathbf{x}_{\text{goal}} = \begin{bmatrix} 4 \\ 0 \end{bmatrix}$.

Use `fmincon` to find a cost-minimizing trajectory. Specifically, the direct transcription formulation is:

$$\begin{aligned} \min_{\mathbf{x}_0,\ldots,\mathbf{x}_T,\mathbf{u}_0,\ldots,\mathbf{u}_{T-1}} \quad & \sum_{t=0}^{T-1} \mathbf{u}_t^\intercal R \mathbf{u}_t \\ \text{subject to} \quad & \mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t) \qquad \forall t \in [0, T-1] \\ & \mathbf{x}_0 = \mathbf{x}_{\text{start}} \\ & \mathbf{x}_T = \mathbf{x}_{\text{goal}} \end{aligned}$$

Use $T = 20$. If done correctly, this should return a straight-line path between start and goal.
For more on trajectory optimization: `http://underactuated.csail.mit.edu/trajopt.html`

(b) By changing *only* the cost function, find and plot 3 different trajectories (different from part (a)). Include your cost function, the plotted trajectory, and a brief justification for why the cost function should produce a different trajectory.

(c) We will now modify the optimization problem to find trajectories in the *light-dark domain* (Platt et al., RSS 2010) from Ex3 Q1(e). Recall that in this problem, the robot's position is *unknown*, and can only be observed via a noisy observation function $h(\mathbf{x}_t) = \mathbf{x}_t + \mathbf{w}_t$,
where $\mathbf{w}_t \sim \mathcal{N}\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}\sigma_w^2(\mathbf{x}_t) & 0\\ 0 & \sigma_w^2(\mathbf{x}_t)\end{bmatrix}\right)$ is zero-mean position-dependent Gaussian observation noise,
and $\sigma_w^2(\mathbf{x}) = \frac{1}{2}(5 - x_x)^2 + 0.1$ (the light source is located at $x_x = 5$, giving more accurate observations).
Our *initial belief* of the robot's position is $\mathcal{N}\left(\mu_{\text{start}}, \sigma_{\text{start}}^2 I\right)$, where $\mu_{\text{start}} = \begin{bmatrix}2\\2\end{bmatrix}$ and $\sigma_{\text{start}}^2 = 5$.
We would like the robot to reach $\mu_{\text{goal}} = \begin{bmatrix}4\\0\end{bmatrix}$, but with much decreased uncertainty $\sigma_{\text{goal}}^2 \leq 0.01$.

To accomplish this, we need to find a trajectory in *belief space*, taking into account the effects of information-gathering and Kalman filtering – specifically, how observations decrease uncertainty.
We expect that the robot may first need to go closer to the light source to decrease its uncertainty, before heading back to the desired goal; i.e., the optimal path may not be a straight-line path as in part (a).

Formulate this as a trajectory optimization (direct transcription) problem. Steps:

- Include uncertainty as part of the state: $\tilde{\mathbf{x}} = [\mu_x, \mu_y, v]^\intercal$, where v is the belief variance.
- Derive the *belief-space dynamics* of the system, i.e., $\tilde{\mathbf{x}}_{t+1} = \tilde{f}(\tilde{\mathbf{x}}_t, \mathbf{u}_t)$.
  We assume that μ and v are being updated using Kalman filtering. Recall that how μ gets updated depends on the actual observation obtained (via the innovation term), which is of course not known during planning. Instead, assume that we will always get the *most-likely* observation, i.e., the zero-noise case (leading to zero innovation). The observation step therefore does not change the mean μ. However, it will still decrease the uncertainty v, which is what we are after.
- Insert additional constraints based on the derived belief-space dynamics and the goal specification.

For more context, see the original paper: `http://www.roboticsproceedings.org/rss06/p37.html`

(d) Use `fmincon` to solve the trajectory optimization problem you defined in part (c).
Plot the resulting trajectory, along with the associated variance at each state (e.g., as a circle).
Hint: For numerical stability, include the constraint $v \geq \epsilon$ as well, where $\epsilon > 0$ is a small positive number.

(e) Investigate what happens if you change T, and possibly other parameters / conditions in the optimization problem. What trends do you observe? Are there any surprising results? Include some examples.

(f) **Extra credit.** Read Section VI.A of the Platt et al. paper (RSS 2010) linked above. Replicate the direct transcription solution shown in Figure 1 (in that section). Implement the replanning strategy shown in Figure 2 (described in Section V.B). Understand and replicate the B-LQR strategy (Section IV) as well.

V1. **[CS 5335 only.] 3 points.** (RVC Ch. 14 Q10.)
In this question, you will track the pose of a moving book cover using a fixed camera.

Choose a book that you have a physical copy of. Find a reference image of the cover (e.g., from a website).

(a) Using a fixed camera (e.g., webcam), capture a sequence of images that includes the book's front cover.

(b) Compute SIFT or SURF features in each image, match them,
and use RANSAC to estimate an homography between consecutive views of the book cover.

(c) Decompose the homography to estimate the rotation and translation between consecutive images.

(d) Include examples of your output (sequence of images, example matches, and detected transformations).

(e) *Optional*: Put this in a real-time loop and continually display the pose of the book relative to the camera.

You will likely find it helpful to first read RVC Ch. 14.1 (Feature correspondence) and Ch. 14.2.4 (Planar homography), and follow their examples (you will need to obtain Peter Corke's Machine Vision Toolbox).
For this question only, you may use the toolbox functions described (including `isift`, `isurf`, `match`, `homography`, `homtrans`, `ransac`), but you are invited to re-implement as many of these as you can for pedagogical purposes.

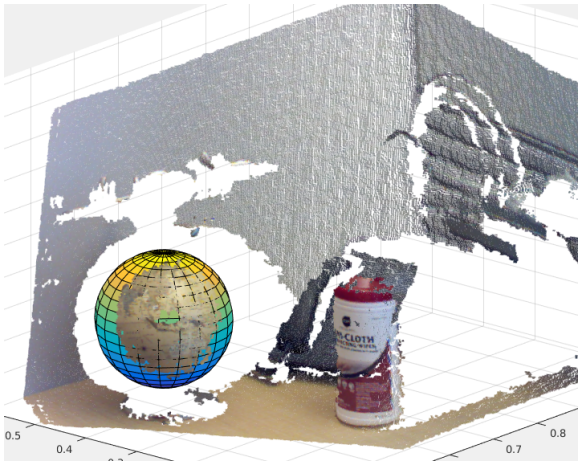
V2. **3 points.** (RVC Ch. 14 Q21.) In this question, you will implement and investigate the ICP algorithm.

(a) Implement the iterative closest point (ICP) algorithm.

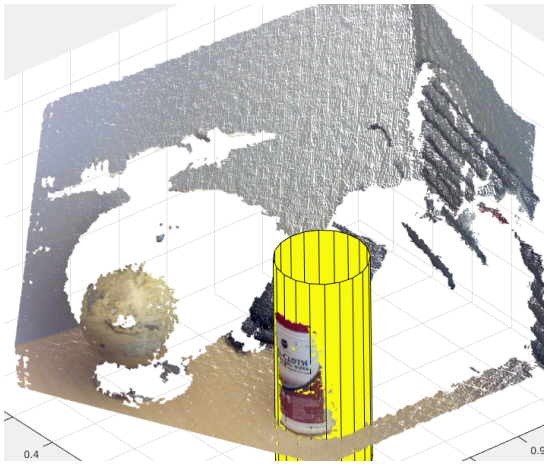
(b) Load `bunny.mat` in `ex5_data.zip`. Create a second point cloud by applying a random transformation. Run your ICP implementation on these two point clouds to estimate the transformation between the two point clouds. Were you able to recover the correct solution?

(c) Investigate the robustness of ICP. Do this by perturbing the initial transformation between the two point clouds, adding noise to the point locations, and adding spurious points / removing existing points. Find examples where ICP is able to recover the solution, and cases where it fails to do so.

(d) **Extra credit.** Can you find point clouds that are particularly easy or difficult for ICP? What makes point clouds easy/difficult to align? Generate some hypotheses and investigate them.



Sphere found in point cloud.



Cylinder found in point cloud.

V3. **5 points.** In this question, you will implement and use RANSAC to identify various geometric primitives in the given point cloud data (`ptcloud.mat` in `ex5_data.zip`), shown in the figure above. Notes:

- Do **not** use the point cloud processing/fitting functions in the MATLAB Computer Vision Toolbox. However, you may try them to compare against your solution.
- You may find it helpful to write helper routines to visualize your solution, as shown above.
- Try to make your solution run reasonably efficiently. Under a minute is ideal; a few minutes is okay; an hour is not. Indicate how long it takes on your computer to find a solution so we have a rough estimate. The bottleneck is typically the number of RANSAC iterations; try a small number first.
- If your solution is too slow, you may specify a rough region-of-interest to crop the point cloud to limit the number of points being considered, or possibly downsample the point cloud (can be risky). If you do this, indicate very clearly in your code comments that you are doing so.

(a) Write a function to compute the surface normal at a given point in the point cloud.

(b) Using this function and RANSAC, identify the planes in the point cloud. Which one is the table?

(c) The point cloud contains a ball that is only partially visible to the sensor. The position of the center of the ball is unknown. The radius is also unknown, but is between 0.05m and 0.10m. Use RANSAC to determine these two quantities.

Note: In your code comments, describe the strategy you are using to fit the sphere. Randomly guessing the center/radius and checking is possible, but likely too inefficient. One way to generate sphere hypotheses is to sample a point in the point cloud, sample a radius within the specified range, and assume the sampled point is on the surface of the sphere. Then, if we compute the surface normal at the point, and follow the normal direction for a distance equal to the sampled radius, we arrive at the center of the sphere candidate. You are welcome to devise and implement your own strategy.

(d) The point cloud also contains a cylinder. The cylinder is defined by an unknown center, radius (between 0.05m and 0.10m), length, and orientation. The orientation should be returned in the form of a unit vector pointing along the axis of the cylinder (either direction is fine).
Use RANSAC to determine these unknown quantities.

Note: Similar to the previous part, you may devise your own strategy, and describe it in your code comments. One way to generate cylinder hypotheses is to sample *two* points in the point cloud, and assume both points are on the surface of the cylinder. The cylinder axis can then be found by taking the cross product between the surface normals at the two points (if they are not parallel). A strategy similar to that described in the previous part can be used to identify the radius and center. It may be helpful to project the points onto the plane orthogonal to the identified axis (multiply points by $I - \hat{a}\hat{a}^\intercal$, where $\hat{a}$ is the cylinder axis) so that the remainder of the problem reduces to identifying a circle in the plane. The length (and midpoint along the cylinder axis) can be identified separately by some form of thresholding, once the other parameters have been identified.

(e) **Extra credit.** Apply the above methods to detect geometric primitives in other point cloud data (e.g., download existing data, capture your own, or generate synthetic data).
Include the data you use and describe any additional assumptions made. Investigate robustness.

(f) **Extra credit.** Devise a strategy to detect cuboids in point clouds (potentially with arbitrary position, orientation, and extent). Implement and investigate your strategy on various point cloud data.