Perspective-N-Point Observation Model

Project Overview

In this project, you will implement and analyze a computer-vision based observation model using the perspective-n-point method. This will serve as a measurement model for the subsequent nonlinear filters module project.

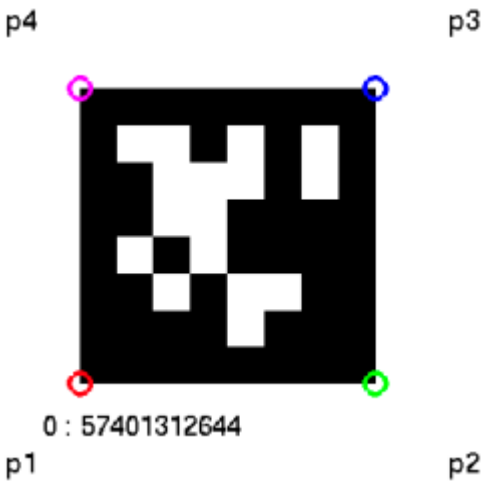

Figure 1: AprilTag example.

Input Data

The data for this project was collected using a Nano+ quadrotor that was either held by hand or flown through a prescribed trajectory over the mat of AprilTags shown in **Error! Reference source not found.**.

Map

Each tag in the map has a unique ID that can be found in the file `parameters.txt`. The tags are arranged in a 12 x 9 grid. The top left corner of the top left tag as shown in the map image below should be used as coordinate (0, 0) with the x coordinate going down the mat and the y coordinate going to the right as oriented in the image. The z coordinate for all corners is 0. Each tag is a 0.152 m square with 0.152 m between tags, except for the space between columns 3 and 4, and columns 6 and 7, which is 0.178 m. Using this information, you can compute the location of every corner of every tag in the world frame.

Camera Calibration

The camera calibration matrix and distortion parameters are given in the file `parameters.txt` and shown again below.

$$\mathbf{M} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 314.1779 & 0 & 199.4848 \\ 0 & 314.2218 & 113.7838 \\ 0 & 0 & 1 \end{bmatrix}$$

Camera Distortion Parameters		
Radial	k_1	-0.438607
	k_2	0.248625
	k_3	-0.0911
Tangential	p_1	0.00072
	p_2	-0.000476

Together these make up the intrinsic parameters of a camera. See the explanations here for more details about what the specific parameters mean, how they are organized, and how one could determine them in practice. The camera is located on the bottom of the drone and points directly downwards, as Figure 2 shows. You will need to transform your camera-based pose estimate from the camera frame to the robot frame so that you can compare it against the ground truth motion capture (`vicon`) data. The parameters for this transformation are also given in the

file `parameters.txt`. The translation vector from camera's coordinate frame to the drone's coordinate frame is $t = [-0.04 \quad 0 \quad -0.03]^T$. There is also a rotation about the camera's z-axis of $\pi/4$ rad and about the x-axis of π rad.

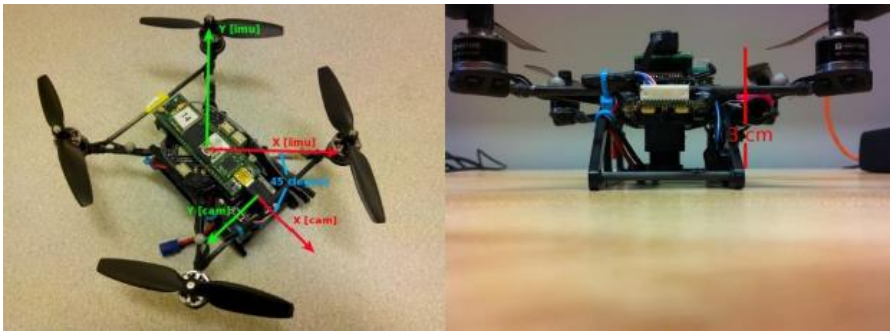

Figure 2: Drone configuration and parameters.

Pose Estimation

The data for each trial is provided in a `.mat` file. The file contains a struct array of image data called `data`, which holds all the data necessary to do pose estimation. In Python you can use the `loadmat` function from the `scipy.io` module to load in the source data. Be sure to use the argument `simplify_cells=True`. This ensures that nested structures are read in properly. The `data` struct contains the following fields:

1. `img` – Rectified image
2. `id` – Array with the ID of every AprilTag that is observed in the image.
3. p1, p2, p3, p4 – Arrays with the four corners of every AprilTag in the image. The corners are given in the order bottom left (p1), bottom right (p2), top right (p3), and top left (p4), (see Figure 1). The ith column in each of these fields corresponds to the ith tag in the ID field. The values are expressed in image (pixel) coordinates.
4. `id` – Time stamp measured in seconds.
5. `rpy` – 3x1 orientation vector with the roll (ϕ), pitch (θ), and yaw (ψ) measured in *radians* by the IMU.
6. `omg` – 3x1 angular velocity vector measured in rad/s by the IMU.
7. `acc` – 3x1 linear acceleration vector measured in $\mathrm{m/s^2}$ by the IMU.

Note that for some packets no tags are observed, you therefore do not need to compute the pose for those packets. Using the camera calibration data, the corners of the tags in the frame, and the world frame location of each tag you can compute the measured pose of the drone for each packet of data.

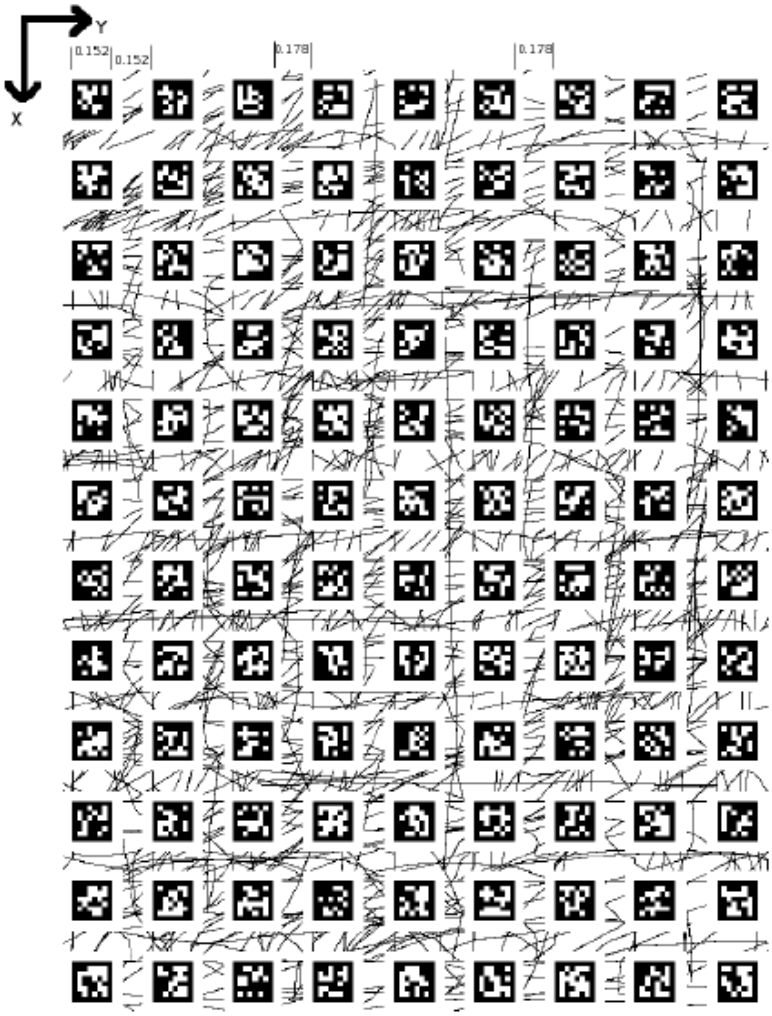

Figure 3: AprilTag map

Ground Truth Data

The `.mat` file also contains motion capture data taken at 100 Hz, which will serve as our ground truth measurements. The motion capture data is stored in two matrix variables, `time` and `vicon`. The `time` variable contains the timestamp in seconds while the `vicon` variable contains the motion capture data in the following format: $[x, y, z, \phi, \phi, \psi, v_x, v_y, v_z, \omega_x, \omega_y, \omega_z]$

You may use this ground truth data to observe how well your estimated pose lines up with the true pose, but your observation model code should not directly use this data in any way.

Project Structure and Guidance

This will be the first of two projects using this data set. This project will involve building both a detailed observation model that will be used as the observation model for a nonlinear estimation algorithm in the subsequent project. You will re-use the same observation model in the subsequent Particle Filter project. I suggest that you develop the code to complete the tasks and deliverables in the project as a self-contained file (ex: as a single Python module `obs.py`) and use a Jupyter notebook to function as a UI/CLI and document to help you build your report. Again, ultimately, I will ask you to submit both your code and a written report detailing your process and task deliverables. Writing a self-contained code library or module will help you re-use the code and be a cleaner submission to grade.

Project Requirements

Pose Estimation

Using the information outlined above you need to write code to estimate the pose of the robot. The basic approach you should use is to solve the PnP (Perspective-n-Point) problem, which estimates the pose of a (calibrated) camera using n 3D points in the world (here, the AprilTag corners from the map layout) and their corresponding 2D projections in the image plane (here, the AprilTag corners from the image). At least 3 correspondences are necessary to make a match. Luckily, each AprilTag gives us 4 points. However, the more points you use the more accurate the resulting estimate should be. In Python, you can use the `solvePnP` method from the OpenCV library. It is up to you to determine how to take the provided data and input it into the corresponding function.

You must submit a function called `estimate_pose` with the following input/output structure:

Inputs

- `data` – a single element from the array of data described above.

Outputs

- `position` – a 3x1 array showing the estimated position of the robot.
- `orientation` – a 3x1 array containing the Euler angles.

Note that in either programming language you should take the resulting orientation data ($\mathbf{q} = [\phi, \theta, \psi]$, [roll, pitch, yaw]) convert it to a rotation matrix, and then use the matrix below to solve for the three Euler angles ($c\theta$ is shorthand for $\cos(\theta)$.)

Visualization

Your next task is to use the pose estimation function from Task 1 to visualize a trajectory. Your code should take in a `.mat` file name and loop over the data to estimate the pose at each time step. You should generate a 3d plot showing both the estimated and ground truth position of the drone over time. I recommend using the image style for the axis to ensure equal scaling so that you can see the true shape of the trajectory. Your code should also generate a second figure with 3 subplots that plot both the estimated and truth value for the roll, pitch, and yaw. Provide some analysis for this observation model commenting on its overall performance.

Covariance Estimation

Ultimately, we want to use this observation model for a navigation estimation filter (Kalman, Partilce, etc.) However, we're using a camera to sense position as it relates to a map and manufacturers typically don't do quality assurance testing for such a use case. Therefore, we'll have to do it ourselves! We'll use the physics and geometry from the perspective-n-point method to form the basis of our measurement, but now we need to check the reliability of the measurement and develop a noise parameter.

For this task the goal is to estimate the covariance matrix in the observation model: $\mathcal{N}(\mathbf{0}, \mathbf{R}) \rightarrow \mathbf{R}$. We will assume that the noise is zero-mean. Next, we can use the formula for sample covariance.

$$\mathbf{R} = \frac{1}{n-1}\sum_{t=1}^{n} \mathbf{v}_t \mathbf{v}_t^T$$

where $\boldsymbol{v}_t$ is the difference between the true position and orientation and the position an orientation reported by your measurement model, and n is the number of samples.

Grading Rubric

Category	Unacceptable	Marginal	Acceptable	Excellent
Task 1: Drone Observation Model	No attempt. ***0 Points***	The student attempted to implement the observation model but there are major errors. ***7 points***	The student implemented the observation model with few or minor errors. ***8 points***	The student correctly implemented the observation model and verified its performance using a suitable error metric. ***10 points***
Task 2: Trajectory visualization	No attempt. ***0 Points***	Some but not all the required figures and data are plotted. ***7 points***	The required figures are plotted. ***8 points***	The required figures are plotted, visually distinct and appealing and discussed. ***10 points***
Task 3: Covariance Estimation	No attempt. ***0 Points***	The student attempts to estimate the measurement model covariance unsuccessfully. ***7 points***	The student estimates the measurement model covariance through iterative guesses. ***8 points***	The student estimates the measurement model covariance and provides a mathematical justification for the results and any subsequent rounding. ***10 points***
Code	The student does not submit their code, or their code does not run. ***0 Points***	The code is poorly organized, documented, or lacking architecture and would require significant explanation to a collaborator but is still sufficient to provide results. ***7 Points***	The code is detailed with a clear structure. Code is broken out into logical and intuitive segments with clear functionality. ***8 points***	Code is comprehensive and well written. Detail is put into the architecture such that it is readily apparent how to use the code. When not, additional comments and documentation are provided such that a collaborator can easily understand it. ***10 points***
Report Quality	The student fails to submit a report that adequately describes their work or address the questions and deliverables of the assignment, is	The report is incomplete or poorly written. Lacks clear organization and sufficient detail. ***7 points***	The report is detailed with a clear structure: introduction, methodology, results, and conclusion. Thorough documentation of	Comprehensive and well-written report with exceptional clarity. In-depth analysis and discussion of results and challenges. Includes additional sections on potential improvements and future work. Professional presentation

	poorly organized, unprofessional, or is otherwise unacceptable. ***0 points***		algorithms, simulation setup, and performance evaluation. Includes figures to support the analysis. ***8 points***	with high-quality figures and visuals. ***10 points***