Nonlinear Kalman Filter

Project #3

Introduction

In this project, you will implement a nonlinear Kalman Filter (your choice of EKF or UKF) to estimate the pose of a quadcopter drone. We will leverage our knowledge of the dynamics of the quadcopter to construct an accurate process model and create a computer-vision based measurement model to provide position and orientation measurements from a known grid of AprilTags (example show in Figure 1). With this model we will then determine values for the covariance matrices in the process and measurements models (tuning).

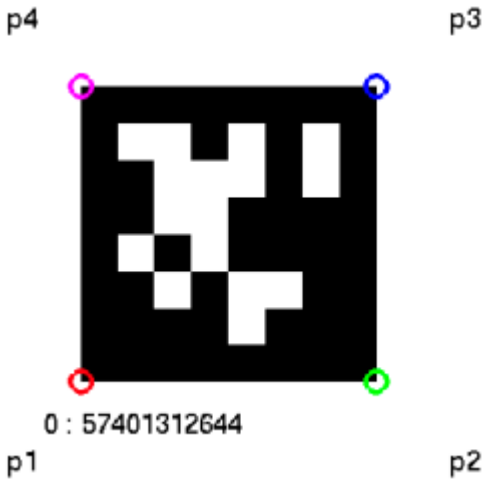

Figure 1: AprilTag example.

In this problem, we will model the pose using a typical fifteen-state model used in inertial navigation. This state vector will consist of the three-axis components of the linear position, orientation, linear velocity, gyroscope bias, and accelerometer bias as shown below:

$$\mathbf{x} = \begin{bmatrix} \mathbf{p} \\ \mathbf{q} \\ \dot{\mathbf{p}} \\ \mathbf{b}_g \\ \mathbf{b}_a \end{bmatrix}$$

Additionally, we will define the orientation using a Z-X-Y Euler angle parameterization corresponding to rolling, pitching, and yawing ($\mathbf{q} = [\phi, \theta, \psi]$). In rotation matrix form these yield:

$$R(\mathbf{q}) = \begin{bmatrix} \cos\psi\cos\theta - \sin\phi\sin\psi\sin\theta & -\cos\phi\sin\psi & \cos\psi\sin\theta + \cos\theta\sin\phi\sin\psi \\ \cos\theta\sin\psi + \cos\psi\sin\phi\sin\theta & \cos\phi\cos\psi & \sin\psi\sin\theta - \cos\psi\cos\theta\sin\phi \\ -\cos\phi\sin\theta & \sin\phi & \cos\phi\cos\theta \end{bmatrix}$$

Input Data

The data for this phase 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 Figure 3.

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. 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 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

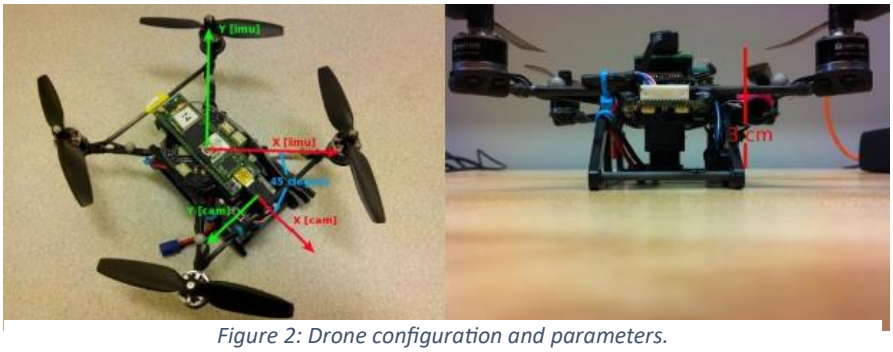

Figure 2: Drone configuration and parameters.

The camera calibration matrix and distortion parameters are given in the file `parameters.txt`. 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 is also given in the file `parameters.txt`.

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 *radians* per second by the IMU.
7. `acc` – 3x1 linear acceleration vector measured in $\frac{\mathrm{m}}{\mathrm{s}^2}$ 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.

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.

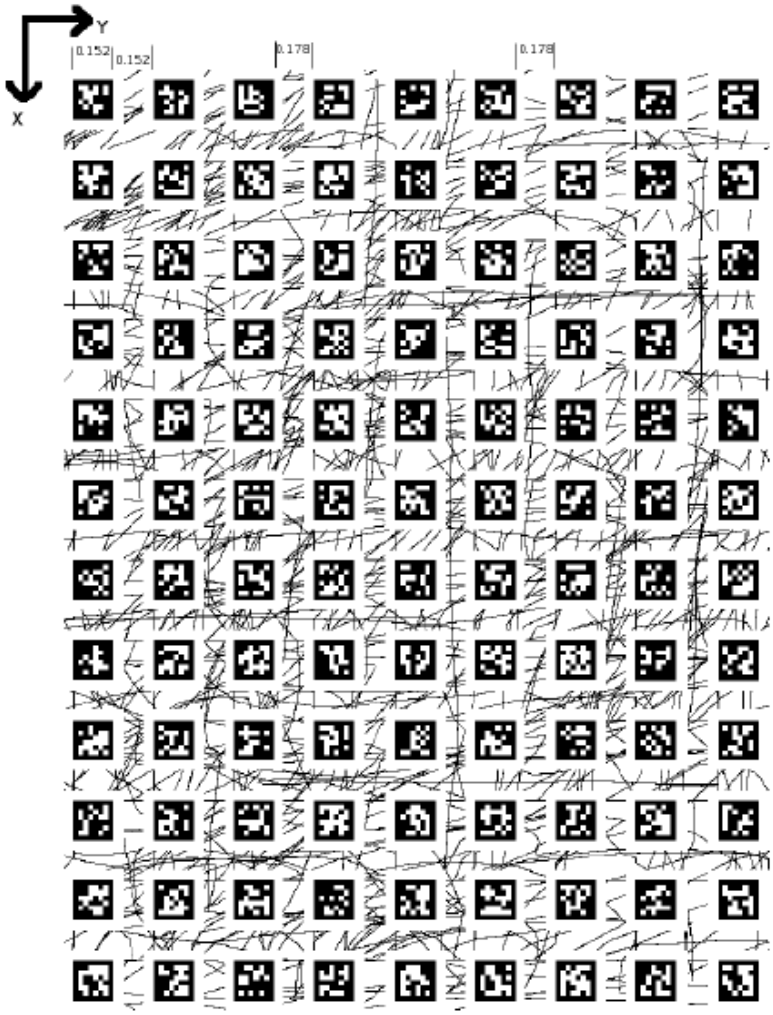

Figure 3: map of AprilTag layout and locations.

System Models

Process Model

We will base our process model on the acceleration output from the IMU. Recall that IMU output is notoriously noisy. We will start our investigation of inertial navigation by learning how to account for the bias terms and account for the change in the IMU out with respect to the gravity vector. In this scenario, we need to account for this due to the "higher" order dynamics of the drone (pitching and rolling). In larger scale scenarios we will need to do similar corrections due to the curvature of the Earth.

We'll start by modeling the gyroscopes and accelerometers by assuming they give a noisy estimate of the true values (angular velocities $\boldsymbol{\omega} = [\omega_x, \omega_y, \omega_z] = \mathbf{u}_\omega$ and linear accelerations $\boldsymbol{a} = [\ddot{p}_x, \ddot{p}_y, \ddot{p}_z] = \mathbf{u}_a$).

$$\hat{\boldsymbol{\omega}} = \boldsymbol{\omega} + \mathbf{b}_g + \mathbf{n}_g$$

$$\hat{\boldsymbol{a}} = R(\mathbf{q})^T(\mathbf{a} - \mathbf{g}) + \mathbf{b}_a + \mathbf{n}_a$$

These values correspond to the values stored in the `omg` and `acc` fields. Additionally, we'll assume that the drift on the IMU biases is described by a normal white noise process.

$$\begin{bmatrix} \dot{\mathbf{b}}_g \\ \dot{\mathbf{b}}_a \end{bmatrix} = \begin{bmatrix} \mathbf{n}_{bg} \\ \mathbf{n}_{ba} \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{Q}_g \\ \mathbf{Q}_a \end{bmatrix}\right)$$

Putting this all together we get the continuous time process model:

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{\mathbf{p}} \\ G(\mathbf{q})^{-1}\mathbf{u}_\omega \\ \mathbf{g} + R(\mathbf{q})\mathbf{u}_a \\ \mathbf{n}_{bg} \\ \mathbf{n}_{ba} \end{bmatrix}$$

where $\mathbf{u}_\omega$ and $\mathbf{u}_a$ are the commanded angular velocity and linear acceleration, $\mathbf{g}$ is the gravity vector, and $G(\mathbf{q})$ is defined as:

$$G(\mathbf{q}) = \begin{bmatrix} \cos\theta & 0 & -\cos\phi\, sin\theta \\ 0 & 1 & \sin\phi \\ \sin\theta & 0 & \cos\phi\cos\theta \end{bmatrix}$$

Observation Model

Our observation model is relatively simple. We will be using a computer vision-based technique that measures the position and orientation:

$$\mathbf{z} = \begin{bmatrix} \mathbf{p} \\ \mathbf{q} \end{bmatrix} + \boldsymbol{\nu}$$

Fortunately, this relates back to the state space linearly:

$$\mathbf{z} = \begin{bmatrix} \mathbf{I} & 0 & 0 & 0 & 0 \\ 0 & \mathbf{I} & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{p} \\ \mathbf{q} \\ \dot{\mathbf{p}} \\ \mathbf{b}_g \\ \mathbf{b}_a \end{bmatrix} + \mathcal{N}(\mathbf{0}, \mathbf{R})$$

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 as well as a nonlinear Kalman Filter. 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 *.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.

As a final point, as the main goal and deliverable of this project is to build a nonlinear Kalman Filter, my version of the observation model is provided as a compiled Python library with documentation. You may use this to develop the EKF/UKF separately from the observation model deliverables.

Task 1: 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 $\cos(\theta)$.)

Task 2: 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.

You do not need to implement the observation model in order to create the visualizations. This section serves both as a sanity check to see if your implementation of the observation model is working correctly, and for you to explore the datasets. If you are unable to implement the observation model, please plot the requested data using the results from your implementation and note in your report that you used the provided implementation of the observation model.

Task 3: Covariance Estimation

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.

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

To get the value of $\boldsymbol{\nu}_t$ at t rearrange the observation model to solve for $\boldsymbol{\nu}_t$. For the state, use the ground truth data.

You must submit a function call `estimate_covariances` with the following input/output structure:

- Inputs
  - `filename` – a string with the name of the data file to use.
- Outputs
  - `R` – a 6x6 matrix containing the estimates observation model covariance.

Task 4: Nonlinear Kalman Filter

Your next task is to implement a nonlinear Kalman filter of your choice, the EKF or UKF, to track the pose of the quadrotor. You should use the covariance values from Task 1 as a starting point for the observation model. However, you will likely need to tweak (increase) these parameters a bit to get better tracking results. I encourage you to test the covariance values across all of the datasets to compare how reliable the values are before you select the values you want to use in your filter. Note: if you have trouble with Task 1, you can still complete Task 2. You can simply guess and check to find appropriate covariance values that yield good results.

For the process model, you will need to guess and check. I recommend starting with a diagonal covariance matrix with values in the range 0.001 – 0.01. Again, you will likely need to adjust these values slightly. Your code should use a single set of covariance values for all the datasets. The goal is to get an estimator that works well across many different scenarios, not to over-tune it to each specific situation.

For each trajectory, compare the ground truth data from the motion capture system, the raw position and orientation observations, and the filtered state. The primary comparison point should be the base position states $[p_x, p_y, p_z]$, but you should describe all your findings and anything interesting you note about the trajectory. Provide at least a three-dimensional plot of the position truth, observations, and estimates.

To give yourself an objective bar to compare performance I would suggest that you calculate the root-mean-square error ($RMSE = \sqrt{\frac{1}{n}\sum_i^n (x_i - \bar{x}_i)}$) of the position states with respect to time as well. This is not precisely a required deliverable but is a useful tool to measure if your implementation is performing correctly.

Grading Rubric

Task 1: Drone Observation Model	**Excellent – 4 pts** The student correctly implemented the observation model and verified its performance.	**Acceptable – 3 pts** The student implemented the observation model with few or minor errors.	**Marginal – 2 pts** The student attempted to implement the observation model but there are major errors.	**Unacceptable – 0 pts** The student did not attempt to implement the observation model and relied.
Task 2: Trajectory visualization	**Excellent – 3 pts** The required figures are plotted, visually distinct and appealing and discussed.	**Acceptable – 2 pts** The required figures are plotted.	**Marginal – 1 pts** Some but not all the required figures and data are plotted.	**Unacceptable – 0 pts** None of the required figures or data is plotted.
Task 3: Covariance Estimation	**Excellent – 6 pts** The student estimates the measurement model covariance and provides a	**Acceptable – 4 pts** The student estimates the measurement model covariance	**Marginal – 2 pts** The student attempts to estimate the measurement model	**Unacceptable – 0 pts** The student fails to estimate the covariance for the observation model.

	mathematical justification for the results and any subsequent rounding.	through iterative guesses.	covariance unsuccessfully.	
Task 4: Nonlinear Kalman Filter	**Excellent – 6 pts** The student successfully implements a nonlinear Kalman filter that accurately tracks position with analysis given to how it compares to the raw observations and motion capture data.	**Acceptable – 4 pts** The student successfully implements a nonlinear Kalman filter that tracks position. Some further manual tuning of the process model may be required to improve performance.	**Marginal – 2 pts** The student attempts to implement a nonlinear Kalman filter, but with many minor errors or some major errors.	**Unacceptable – 0 pts** The student fails to implement a nonlinear Kalman Filter.
Code and Report Quality	**Excellent – 6 pts** The student has intuitive, concise, well written code that follows a logical and organized structure. Comments are provided and document the functionality of the code. The student's report provides a very detailed description of their work. It addresses the questions and deliverables posed by the assignment and provides some discussion as to their results.	**Acceptable – 4 pts** The student's code makes sense given the context of the problem. Some comments or documentation is provided. The student's report provides a description of their work and addresses the questions and deliverables posed by the assignment.	**Marginal – 2 pts** The student's code is poorly structured and not intuitive. Little to no comments or documentation is provided. The student's report provides little to no description of their work or fails to address the questions and deliverables of the assignment beyond a basic answer.	**Unacceptable – 0 pts** The student's code fails to run, runs incorrectly, or otherwise fails to address the problem. The student's report fails to adequately describe their work or address the questions and deliverables of the assignment.