# Nonlinear Navigation Filters

## Project Overview

In this project you will implement two nonlinear navigation filters that make use of the observation and measurement model you implemented in project 2. These will be paired together to estimate the pose of a quadcopter drone. 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 this yields:

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

Please refer to the description in project 1 for information about the map and observation model.

### 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:

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

## Measurement Model

Our measurement model is relatively simple. We will be using the computer vision-based technique developed in the previous assignment 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:

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

You should also use the covariance estimate developed from the previous assignment.

# Project Requirements

In this project you will implement a nonlinear Kalman filter (your choice of either an Extended or Unscented) as well as a Particle filter.

## Nonlinear Kalman filter

### Implementation

You will likely need to tweak (increase) the noise 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. 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.

### Analysis

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 and a discussion of these results.

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.

## Particle Filter

### Implementation

The algorithm for a particle filter is readily implemented by using an object-style range-based `for` loop. However, such loops tend to be computation inefficient, particularly in Python. Therefore, I recommend using a vectorized approach and storing the particle list in an array and making use of linear algebra libraries (NumPy, Eigen, et cetera) to perform the actual computation. Again, despite this linear algebra-based implementation, I still strongly recommend not using MATLAB. If you choose to use a vectorized approach, be careful and understand how the library orders operations. Python is row-wise (ex: N x 15).

In the prediction step, for each particle you need to take the original particle, sample from the noise distribution, and use the measured inputs plus noise to determine the future state of that particle using the process model. You should use the same noise covariance values you used for the Nonlinear Kalman Filter as a starting point for the process noise.

In the update step, you need to use the observation model and the measurements to calculate its importance weight for each particle and then use the low variance resampling algorithm to find your updated particle set. Again, you should use the covariance values you found in the previous assignment as a starting point for the observation model.

### Analysis

One notable drawback of the particle filter is the lack of a ready navigation solution. While a Kalman filter provides a mean and covariance, the particle filter has only the particle distribution. Therefore we must introduce some method of sampling the particle field to determine a singular position estimate. There are a couple ways of doing this, please examine the error from truth by taking the position estimate as: the highest weighted particle, the average over all particles, and the weighted average over all particles. Finally, compute the root-mean-square error over all particles with respect to truth to have a general gauge of how your implementation is performing. RMSE is the typical error characterization of a particle filter, however it is only useful when you have a ground truth estimate.

With the error and navigation solution metrics as defined above established, investigate the performance of the particle filter for a 250, 500, 750, 1000, 2000, 3000, 4000, and 5000 particles. Are there any trends to these results?

## Comparison

While both of these methods are nonlinear, they represent two very disparate approaches. Please discuss in 2-3 paragraphs the results from this assignment as well as the previous Nonlinear Kalman Filter assignment. Some specific points to discuss:

- Ease of implementation
  - Which method was easier to write the code for? Why?
  - Which method was easier to tune parameters for?
- Speed of code
  - Which method runs faster?
  - Why might this be important?
- Accuracy of results: which method yielded more accurate tracking results?

# Grading Rubric

| Category | Unacceptable | Marginal | Acceptable | Excellent |
|---|---|---|---|---|
| **Nonlinear Kalman Filter Implementation** | The student did not implement a the filter<br><br>***0 points*** | The student attempts an implementation, but the filter does not accurately track the true state.<br>***7 points*** | The student successfully implements the filter, and it tracks the state.<br>***8 points*** | The student successfully implements a filter that accurately tracks the true state.<br>***10 points*** |
| **Nonlinear Kalman Filter Analysis** | The student provides no analysis or discussion on their nonlinear Kalman filter implementation<br>***0 points*** | The student provides a cursory analysis of their filter.<br>***10 points*** | The student analyzes the process noise developed and their results.<br>***12 points*** | The student provides a detailed analysis of their filters design and results.<br>***15 points*** |
| **Particle Filter Implementation** | The student did not implement a particle filter<br>***0 points*** | The student attempts an implementation, but the filter does not accurately track the true state.<br>***7 points*** | The student successfully implements the filter, and it tracks the state.<br>***8 points*** | The student successfully implements a filter that accurately tracks the true state.<br>***10 points*** |
| **Particle Filter Analysis** | The student provides no analysis or discussion on | The student provides a cursory analysis of their filter. | The student analyzes the process noise | The student provides a detailed analysis of their filters design and results.<br>***15 points*** |

| | | | | |
|---|---|---|---|---|
| | their particle filter implementation<br>***0 points*** | ***10 points*** | developed and their results.<br>***12 points*** | |
| **Comparison Discussion** | Little to no discussion of the way each method compares to the other is provided.<br>***0 Points*** | The student provides simplistic basic analysis providing straightforward direct answers to the comparison questions.<br>***14 points*** | The student provides answers to the discussion questions and provides evidence and/or discusses their rational and findings based on their results.<br>***16 points*** | The student provides detailed responses to the discussion questions with reference and evidence based on their simulation findings.<br>***20 points*** |
| **Code** | The student does not submit their code, or their code does not run.<br>***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.<br>***7 Points*** | The code is detailed with a clear structure. Code is broken out into logical and intuitive segments with clear functionality.<br>***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.<br>***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 poorly organized, unprofessional, or is otherwise unacceptable.<br>***0 points*** | The report is incomplete or poorly written. Lacks clear organization and sufficient detail.<br>***14 points*** | The report is detailed with a clear structure: introduction, methodology, results, and conclusion. Thorough documentation of algorithms, simulation setup, and performance evaluation. Includes figures to support the analysis.<br>***16 points*** | 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 with high-quality figures and visuals.<br>***20 points*** |