Kalman Filter

Project #2

In this assignment you will use a Kalman filter to track the motion of a drone flying through an open space. A Qualisys motion capture system, consisting of a set of infrared cameras positioned around the flight space, tracks the drone by measuring the spherical retroreflective markers placed on the drone, as shown below in Figure 1. The motion capture system can track the drone at a rate of over 100 Hz.

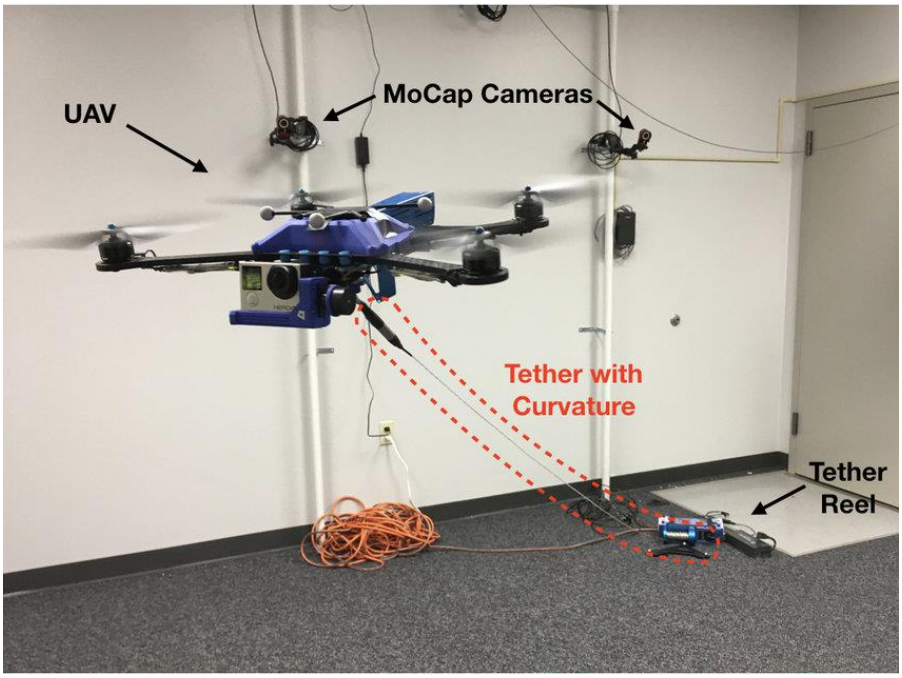

Figure 1: Example of a quadcopter drone with a motion capture system.

Input

The data you will use is a mixture of real and hybrid (real/simulated) data. The real data comes from the Qualisys motion capture system. The hybrid data comes from taking the real data and artificially adding extra noise to it. There are four separate data files, each based on the same motion of the robot:

1. kalman_filter_data_mocap.txt
2. kalman_filter_data_low_noise.txt
3. kalman_filter_data_high_noise.txt
4. kalman_filter_data_velocity.txt

As the names imply, the first file uses the real motion capture data. The second and third use the real data plus noise of varying levels. Finally, the last file provides measurements of the velocity of the robot (as opposed to its position, like in all of the other files). All data are in SI units, so distances are in meters, time in seconds, etc.

Each file is a CSV (comma separated value) text file with data in the following format:

$$t, u_1, u_2, u_3, z_1, z_2, z_3$$

where t is the time, u_i are the components of the input, and z_i are the components of the measurement. The input in each of the data files is the net force applied on the drone from the propellers. Note that the net force is the applied force minus gravity, so a net force of 0 means the robot is hovering in place. The measurements are either position or velocity, depending on the data file.

System Modeling

Since the input to the system is of second order (a force is proportional to acceleration, the second order derivative of position), the system state must include both the position and the velocity to satisfy the Markov assumption. In other words, the system state should be of the form:

$$x = \begin{bmatrix} p \\ \dot{p} \end{bmatrix}$$

where $\boldsymbol{x}$ is the 6x1 state vector consisting of $\boldsymbol{p}$, the 3x1 position vector, and $\dot{\boldsymbol{p}}$, the 3x1 velocity vector (time derivative of position).

The input vector to the system is $\boldsymbol{u}=m\ddot{\boldsymbol{p}}$, where $\boldsymbol{u}$ is the input vector, m is the mass of the drone (in this case, 27g), and $\ddot{\boldsymbol{p}}$, the 3x1 acceleration vector (second time derivative of position).

As was mentioned above, the observations $\boldsymbol{z}$ are either the position or the velocity of the robot, depending on the data file.

You may assume that all noise is additive Gaussian noise with 0 mean and a diagonal covariance matrix. The simulated noise added to the measurements are of the form $\Sigma=\sigma^2 I$, where I is the identity matrix and σ^2 is the covariance ($\sigma=0.05$ m for low noise, $\sigma=0.20$ m for high noise, and $\sigma=0.05$ m/s for the velocity). It is up to you to determine the noise value for the motion capture data and for the process model noise. When doing this, be sure to think through physically reasonable values. A standard deviation of $\sigma=10^{-6}$ m is probably too low since that implies it is accurate to the micron scale while $\sigma=10^3$ m is too large since that implies there is a kilometer of uncertainty. Pick a reasonable starting value and adjust it until you get the best-looking results. This is what is often done in practice when using the Kalman filter.

Task 1: Kalman Filter Equations

Using this information, determine the matrices used in the process and measurement models. Note, you do **not** need to include the noise matrices. For the process model, be sure to start from a continuous time model, extract the time step from the data, and use that to create a discrete time model. Note that in doing this you may treat the drone as a point mass and the input has no constraints on it. This is not realistic but should still yield good results. Future assignments will look at making things more realistic but constraining the direction of the force from the propellers, etc.

Task 2: Kalman Filter Implementation

Your next task is to implement the Kalman filter to track the motion of the drone. You should use the matrices derived in the previous task in your code. For each of the .txt files your code should be capable of producing the state vector at each time step as well as a three-dimensional plot of the three-dimensional position at a minimum.

To initialize your Kalman filter, you may assume that the robot starts at the first measured position in the input file. The initial covariance is a design parameter for you to tune. You should test different values and see how this affects the output of the filter. As was mentioned above, you also need to determine the noise in the process model. A useful method that leverages known system dynamics and quantities can be found here: https://web.archive.org/web/20220202010928/https://www.kalmanfilter.net/covextrap.html.

You must submit your code and any other files necessary to run your code. Note, you do not need to submit the original data files.

Again, please submit either a Jupyter notebook containing both your code and written report responses to the tasks and questions or a separate report and your raw code file.

Grading Rubric

Task 1	**Excellent – 5 pts** The student has correctly answered the question and provided an explanation detailing how that answer was reached.	**Acceptable – 4 pts** The student has correctly answered the question.	**Marginal – 2 pts** The student has attempted to answer the question, but the answer is incorrect due to minor errors.	**Unacceptable – 0 pts** No answer was given, the response was lacking sufficient detail, or the answer was completely incorrect.
Task 2	**Excellent – 5 pts** The student has correctly implemented a Kalman Filter and accurately tracks position.	**Acceptable – 4 pts** The student has correctly implemented a Kalman filter that tracks position.	**Marginal – 2 pts** The student has implemented a Kalman Filter with some errors.	**Unacceptable – 0 pts** No answer was given, the response was lacking sufficient detail, or the answer was completely incorrect.
Code Quality	**Excellent – 5 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.	**Acceptable – 4 pts** The student's code makes sense given the context of the problem. Some comments or documentation is provided.	**Marginal – 2 pts** The student's code is poorly structured and not intuitive. Little to no comments or documentation is provided.	**Unacceptable – 0 pts** The student's code fails to run, runs incorrectly, or otherwise fails to address the problem.