MEAM 620 Project 2 Phase 3

Due: Thursday, April 18, 2019 at 11:59pm

1 Overview

In this phase you will complete a state estimation pipeline for the quadrotor by combining your pose and velocity estimators with a Kalman Filter. As you have demonstrated thus far, imagery captured by an onboard camera can be used to estimate the pose and velocity of the quadrotor. However, due to noisy sensor data the estimated state of the quadrotor was subject to a significant amount of noise. A Kalman Filter combats these variations by employing a model of the system and noise to reign in erratic sensor measurements.

You will implement two different Kalman Filters for state estimation. The first, `ekf1`, takes world frame linear velocity and body frame angular velocity from VICON as inputs and returns the position and orientation of the quadrotor. The second Kalman Filter, `ekf2`, takes body frame acceleration and angular velocity from the onboard IMU as inputs and returns the pose and linear velocity of the quadrotor. While the process models and inputs for each filter are different, the measurement for each will be your pose estimator from Phase 1. Note: you can optionally include your velocity estimator from Phase 2, if it runs fast enough.

2 Sensor Data

Throughout this project you have been using a dataset collected from a quadrotor equipped with a camera and IMU outputting synchronized data to a desktop computer. While your code is not designed to run onboard the quad, aspects of the dataset capture conditions common in online algorithms. For example, motion blur and limited field of view result in some sensor packets with few or no detected AprilTags. Further, delays in data transmission may result in invalid sensor packets at certain timesteps (i.e. `sensor.is_ready` is `false`). If a control loop is relying on regular state estimates in order to compute inputs it becomes critical to handle these edge cases appropriately. While before you were simply outputting empty vectors, a Kalman Filter has the added benefit of producing a predicted state when pose and velocity estimation fails (i.e. `estimate_pose` or `estimate_vel` return an empty array). As before, the sensor packet is a `struct` that contains following fields:

```
sensor.is_ready    % True if a sensor packet is available, false otherwise
sensor.t           % Time stamp for the sensor packet, different from the VICON time
sensor.omg         % Body frame angular velocity from the gyroscope
sensor.acc         % Body frame linear acceleration from the accelerometer
sensor.img         % Undistorted image.
sensor.K           % Calibration matrix of the undistorted image
sensor.id          % IDs of all AprilTags detected, empty if no tag is detected in the image
sensor.p0          % Corners of the detect AprilTags in the image,
sensor.p1          % the ordering of the corners, and the distribution of the tags,
sensor.p2          % are the same as Project 2 Phase 1
sensor.p3
sensor.p4
```

Note that although the image is provided, it is possible to complete the project without image processing depending on whether you use your velocity estimation code from Phase 2.

3 Extended Kalman Filter

In this project, you are encouraged to use the Extended Kalman Filter (EKF) to estimated the pose (`ekf1` and `ekf2`) and velocity (`ekf2` only) of the quadrotor. The process update and measurement update steps can be run at different rates. In `ekf1`, you should run the process update whenever you receive the velocity information from VICON, but only run the measurement when a tag-based pose estimation is available. You are also welcome to use other variants of the Kalman filter, such as the Unscented Kalman filter (UKF) or Error State Kalman filter (ESKF). A UKF may capture the non-linearity of the system better but with higher computational cost.

Note: in deriving the Kalman Filter process and measurement models it may be useful to use MATLAB's symbolic toolbox; however, **symbolic computations should never be performed in the loop**. Instead, important results should be cached or hardcoded into your functions for best performance.

4 Code Requirement

You will implement two functions `ekf1.m` and `ekf2.m`. The interface for `ekf1.m` is

```
[X,Z] = ekf1(sensor, vic, varargin)
```

where `X` and `Z` are state and measurement respectively. `vic` is a struct that holds VICON velocity and time:

```
vic.vel % world frame linear velocity and body frame angular velocity [vx; vy; vz; wx; wy; wz]
vic.t   % time
```

The interface for `ekf2.m` is

```
[X,Z] = ekf2(sensor, varargin)
```

Further details can be found in header comments of the function outlines.

5 Submission

We'll use the `turnin` system for submission of this part. The project name for this phase is called `proj2phase3`, so the command you'll use for `turnin` would be:

```
$ turnin -c meam620 -p proj2phase3 -v *
```

Your submission should include:

- A `README` file containing anything specific we should be aware of.
- Files `init_script.m`, `ekf1.m`, `ekf2.m` and any additional files your code requires to run.

Shortly after submitting using `turnin` you should receive an email from `meam620@seas.upenn.edu` stating that your submission has been received. After that, you can check the status of your submission on the monitor webpage at https://alliance.seas.upenn.edu/~meam620/monitor/.

Once the automated tests finish for your code, you should receive an email containing your test results. This email will contain plots of your estimates compared to the ground truth and also tell you the RMS error compared to ground truth and you how long each iteration of your code takes. Please make sure you don't have any `tic` or `toc` inside your functions. Your grade would depend on the time taken per iteration and the error between your estimate and the ground truth.