MEAM 620 Project 2 Phase 3

Due: Monday, April 20, 2015, 11:56pm

1 Overview

It is time to put together everything that you've learned in this course! In this phase, you'll use vision and estimation in addition to the controller you developed in Project 1.

You will implement two different Extended-Kalman Filters for state estimation. You will start with an easy one, which takes world frame linear velocity and body frame angular velocity from Vicon as your inputs. We will refer to this one as **ekf1**. Then you will move on to use body frame acceleration and angular velocity from the onboard IMU as your inputs. We will refer to this one as **ekf2**. Thus, you will have two different process models and inputs. However, the measurement remains the same in both two, which will be your pose estimator from Phase 1. (You can optionally include your velocity estimator from Phase 2, if it runs fast enough).

2 Sensor Data

The onboard processor of the robot collects synchronized camera and IMU data and sends them to the desktop computer. The sensor data is decoded into standard MATLAB format. Note that since the sensor data is transmitted via wireless network, there may or may not be a sensor packet available during a specific iteration of the control loop. A 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
```

This struct is accessible in the controller (`student_control_hover`, etc). The sensor time stamp is different than the control time stamp, however, this should not matter since you only need the incremental time for your EKF. Also note that although the image is provided, it is possible to complete the project without image processing.

3 Extended Kalman Filter

In this project, you will need to use the Extended Kalman Filter (EKF) to estimated the pose (**ekf1** and **ekf2**) and velocity (**ekf2** only) of the vehicle. We suggest that you use the `jacobian` function to calculate the required Jacobian matrices and then use the `simplify` function to shorten the symbolic representation. Note that you should precalculate the Jacobians and use `matlabFunction` to generate a function for them.

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 it might require more runtime.

4 Special Instructions

4.1 Runtime Requirement

In this project, you will need to run your vision-based pose estimator (from Phase 1) and the EKF in real-time. It is very likely that this will require significantly more computation than Project 1 Phase 4. As such, you should carefully monitor the computation time of all your algorithms (tag-based pose estimator, EKF, controller, trajectory generator). A **5ms** (**15ms** with the optional Phase 2 velocity estimator) total runtime bound can be considered as safe.

4.2 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
```

While developing **ekf1**, you should generate such structure yourself from the given dataset, as it will be used by our testing function. The interface for `ekf2.m` is

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

Details can be found in the released student code.

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.