Robot Perception and Decision-Making (CS336) Homework #2

(Fall 2019) Due: **Wednesday, October 23**

Name _____ **SUNet ID** _____

1. **Extended Kalman Filter with a Nonlinear Observation Model**

Consider the scenario depicted in Figure 1 where a robot tries to catch a fly that it tracks visually with its cameras. To catch the fly, the robot needs to estimate the 3D position $\mathbf{p}_t \in \mathbb{R}^3$

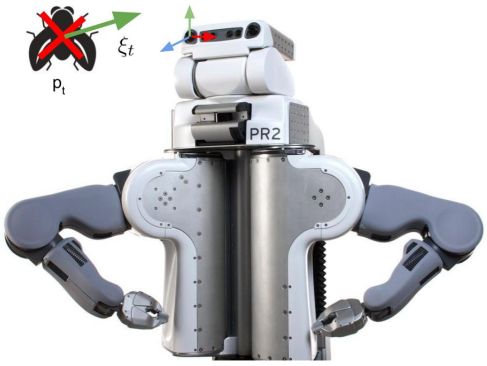

Figure 1

and linear velocity $\xi_t \in \mathbb{R}^3$ of the fly with respect to its camera coordinate system. The fly is moving randomly in a way that can be modelled by a discrete time double integrator:

$$\mathbf{p}_{t+1} = \mathbf{p}_t + \Delta t \xi_t \tag{1a}$$

$$\xi_{t+1} = 0.8\xi_t + \Delta t \mathbf{a}_t \tag{1b}$$

where the constant velocity value describes the average velocity value over Δt and is just an approximation of the true process. Variations in the fly's linear velocity are caused by random, immeasurable accelerations $\mathbf{a}_t$. As the accelerations are not measurable, we treat it as the process noise, $\mathbf{w} = \Delta t \mathbf{a}_t$, and we model it as a realization of a normally-distributed white-noise random vector with zero mean and covariance Q: $\mathbf{w} \sim N(0, Q)$. The covariance is given by $Q = \text{diag}(0.05, 0.05, 0.05)$

The vision system of the robot consists of (unfortunately) only one camera. With the camera, the robot can observe the fly and receive noisy measurements $\mathbf{z} \in \mathbb{R}^2$ which are the pixel coordinates (u, v) of the projection of the fly onto the image. We model this projection mapping fly's 3D location to pixels as the observation model h:

$$\mathbf{z}_t = h(\mathbf{x}_t) + \mathbf{v}_t \tag{1c}$$

where $\mathbf{x} = (\mathbf{p}, \mathbf{s})^T$ and $\mathbf{v}$ is a realization of the normally-distributed, white-noise observation noise vector: $\mathbf{v} \sim N(0, R)$. The covariance of the measurement noise is assumed constant of value, $R = \text{diag}(5, 5)$.

We assume a known 3x3 camera intrinsic matrix:

$$K = \begin{bmatrix} 500 & 0 & 320 & 0 \\ 0 & 500 & 240 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{1d}$$

(a) Let $\Delta t = 0.1s$. Find the system matrix A for the process model (Implement your answer in the *system_matrix* function in Q1.py).

(b) Define the observation model h in terms of the camera parameters (Implement your answer in the *observation* function in Q1.py).

(c) Initially, the fly is sitting on the fingertip of the robot when it is noticing it for the first time. Therefore, the robot knows the fly's initial position from forward kinematics to be at $\mathbf{p}_0 = (0.5, 0, 5.0)^T$ (resting velocity). Simulate in Python the 3D trajectory that the fly takes as well as the measurement process. This requires generating random acceleration noise and observation noise. Simulate for 100 time steps. Attach a plot of the generated trajectories and the corresponding measurements. Please add your code to the report.

(d) Find the Jacobian H of the observation model with respect to the fly's state x. (Implement your answer of H in function *observation_state_jacobian* in Q1.py.)

(e) Now let us run an Extended Kalman Filter to estimate the position and velocity of the fly relative to the camera. You can assume the aforementioned initial position and the following initial error covariance matrix: $P_0 = \text{diag}(0.1, 0.1, 0.1)$. The measurements can be found in `data/Q1E_measurement.npy`. Plot the mean and error ellipse of the predicted measurements over the true measurements. Plot the means and error ellipsoids of the estimated positions over the true trajectory of the fly. The true states are in `data/Q1E_state.npy`

(f) Discuss the difference in magnitude of uncertainty in the different dimensions of the state.

2. **From Monocular to Stereo Vision**

Now let us assume that our robot got an upgrade: Someone installed a stereo camera and calibrated it. Let us assume that this stereo camera is perfectly manufactured, i.e., the two cameras are perfectly parallel with a baseline of $b = 0.2$. The camera intrinsics are the same as before in Question 1.

Now the robot receives as measurement $\mathbf{z}$ a pair of pixel coordinates in the left image (u^L, v^L) and right image (u^R, v^R) of the camera. Since our camera system is perfectly parallel, we will assume a measurement vector $\mathbf{z} = (u^L, v^L, d^L)$ where d^L is the disparity between the projection of the fly on the left and right image. We define the disparity to be positive. The fly's states are represented in the left camera's coordinate system.

(a) Find the observation model h in terms of the camera parameters (Implement your answer in function *observation* in Q2.py).

(b) Find the Jacobian H of the observation model with respect to the fly's state x. (Implement H in function *observation_state_jacobian* in Q2.py)

(c) What is the new observation noise covariance matrix R? Assume the noise for (u^L, v^L), and (u^R, v^R) are independent and same distribution with Q1, respectively. (Implement R in function *observation_noise_covariance* in Q2.py).

(d) Now let us run an Extended Kalman Filter to estimate the position and velocity of the fly relative to the left camera. You can assume the same initial position and the initial error covariance matrix as in the previous questions. Plot the means and and error ellipses of the predicted measurements over the true measurement trajectory in both the left and right images. The measurements can be found in `data/Q2D_measurement.npy`. Plot the means and error ellipsoids of the estimated positions over the true trajectory of the fly. The true states are in `data/Q2D_state.npy` Include these plots here.

(e) Discuss the advantages of using (u^L, v^L, d^L) over (u^L, v^L, u^R, v^R)

3. **Particle Filter with Nonlinear Process Model and Linear Observation Model**
Now assume that the robot moves on to visually tracking an object that it's being manipulated with its own end-effector. Therefore, it does not only need to estimate the 3D position and velocity but also its orientation and angular velocity relative to the camera, and the predictions will be strongly affected by robot's actions.

We assume that we can model the object's motion again by a discrete time double integrator:

$$\mathbf{T}_{t+1} = e^{[\xi_t]\Delta t}\mathbf{T}_t \tag{3a}$$
$$\xi_{t+1} = \xi_t + \Delta t\mathbf{u}_t + \Delta t\epsilon_t \tag{3b}$$

where the constant velocity value describes the average velocity value over Δt and is just an approximation of the true process. Variations in the objects twist velocity are caused by robot actions that we directly obtain as accelerations on the object, $\mathbf{u}_t$, and additional randomness in the motion is modeled as additional accelerations $\mathbf{w} = \Delta t\epsilon_t$. We model $\mathbf{w}$ as a realization of a normally-distributed white-noise random vector with zero mean and covariance Q: $\mathbf{w} \sim N(0, Q)$. Q is a diagonal matrix, $Q = \text{diag}(0.01, 0.001, 0.001, 0.0, 0.005, 0.0025)$.

The robot receives noisy pose estimates of the object relative to the camera frame. We assume the measurements is the true pose $\mathbf{T}$ perturbed with noise in the tangential Lie algebra space. I.e., the noise, represented in exponential coordinates, is a realization of a normally-distributed white-noise random vector with zero mean and covariance $R = \text{diag}(0.1, 0.1, 0.1, 0.05, 0.05, 0.05)$. The likelihood of a pose given an observation is thus defined by the Gaussian likelihood in 6D of the difference between the pose and the observation (in exponential coordinates).

(a) Assume the initial pose of the object wrt. the camera is $(q_0 = (w = 1, 0, 0, 0), p_0 = (1.0, 2.0, 5.0))$. The first 100 actions of the robot (i.e. the acceleration $\mathbf{u}$) are given in file `data/Q3_data/q3_commands.npy`. Simulate the 6D trajectory that the object takes in Python. Generate plot of this trajectories where you represent the object as a 3D coordinate frame in space. In your report, please paste your code for the *simulate* function in Q3.py.

(b) Now let us run a Particle Filter to estimate the pose and twist of the object relative to the camera. You can assume the aforementioned initial position and zero initial pose uncertainty. The measurements from the camera can be found in `data/Q3_data/q3_measurements.npy`. Plot the most likely pose at each time step as a 3D coordinate frame. Include these plots here. In the report, please paste your code for the *particle_filter* function in Q3.py.

4. **Linear Kalman Filter with a Learned Inverse Observation Model**
Now the robot is trying to catch a ball. So far, we assumed that there was some vision module that would detect the object in the image and thereby provide a noisy observation. In this part of the assignment, let us learn such a detector from annotated training data and treat the resulting detector as a sensor.

If we assume the same process as in the first task, but we have a measurement model that observes directly noisy 3D locations of the ball, we end up with a linear model whose state can be estimated with a Kalman filter. (For this problem, you should re-use code from Questions 1 and 2, so we will not be supplying starter code. You might find `Q4_helper.py` helpful for loading the data).

(a) In the folder `data/Q4A_data` you will find a training set of 1000 images in the folder `training_set` and the file `Q4A_positions_train.npy` that contains the ground truth 3D position of the red ball in the image. Design a detector that performs regression from the images to the 3D position. Use standard Deep Learning methodology for training this detector with the training set data. The test set has 100 images in the folder `testing_set`. The file `Q4A_positions_test.npy` contains the corresponding ground truth. Describe the architecture here and the rationale behind it. In the write-up, post your code, your hyperparemeters (learning rate, batch-size, etc), and how many epochs you trained for. Report the training and test set mean squared error in your write-up.

(b) In the folder `data/Q4B_data` you will find a set of 1000 images that show a new trajectory of the red ball. Run your linear Kalman Filter using this sequence of images as input, where your learned model provides the noisy measurements. Tune a constant measurement noise covariance appropriately, assuming it is a zero mean Gaussian and the covariance matrix is a diagonal matrix. Plot the resulting estimated trajectory from the images, along with the detections and the ground truth trajectory (which can be found in the file `Q4B_positions_gt.npy`). Please paste your code in the report.

(c) Because the images are quite noisy and the red ball may be partially or completely occluded, your detector is likely to produce some false detections. In the folder `data/Q4D_data` you will find a set of 1000 images that show a trajectory of the red ball where some images are blank (as if the ball is occluded by a white object). Discuss what happens if you do not reject these outliers but instead use them to update the state estimate. Like the last question, run your linear Kalman Filter using this sequence of images with occlusion as input. Plot the resulting estimated trajectory from images and the ground truth trajectory. Also plot the 3-D trajectory in 2-D (x vs. z) and (y vs. z) to better visualize what happens to your filter.

(d) Design an outlier detector and use the data from `data/Q4D_data`. Plot the 2D projection of the observation trajectory, assuming the camera of Question 1, and label the outliers you have detected in the plot (with a different color). Also plot the ground truth projection of the trajectory (found in file `Q4D_positions_gt.npy`). Explain how you implemented your outlier detector and post your code in your write-up. Hint: Your observation model predicts where your measurement is expected to occur and its uncertainty.