CS231A: Computer Vision, From 3D Perception to 3D Reconstruction and beyond Homework #4

(Spring 2025) Due: **Wednesday, June 4**

Submitting

Prepare a PDF with your answers and submit it to the corresponding Gradescope assignment. We recommend using the provided LaTeX templates from our website, but you may generate the PDF in any preferred manner.

For code implementations, modify the provided `.py` files located in the `code` folder. Problems involving only code implementation are marked with the 🖥 symbol. Written answers are required only where instructions are **bolded** and usually ask you to include the final output of your code and/or a brief description of your implementation. (These questions are also marked with the 📄).

Additionally, submit your completed Python files to the separate Gradescope coding assignment, which includes an autograder for verification. We grade primarily from the PDF and use the autograder submission as needed for additional checks.

Before submission, transfer your code from the notebook into the `.py` files. Compress these Python files into a `.zip` file and submit it to the Gradescope code assignment.

Honor Code

- **Collaboration**: You may collaborate with other students on the homework. However, each student should independently write up their own solutions and **clearly list the names of their collaborators** in their write-up.
- **Committing code to GitHub**: Please do not push your homework code to public GitHub repositories (for example, a fork of our repository). If you wish to commit your solutions to GitHub, please create a **private repository** by making a new copy of our repository, rather than forking it.

1 Extended Kalman Filter with a Nonlinear Observation Model (40 points)

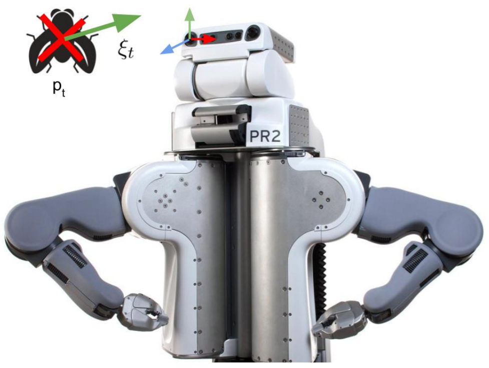

Figure 1

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 $\mathbf{p}_t \in \mathbb{R}^3$ 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 of the fly's 3D location to pixels as the observation model h:

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

where $\mathbf{x} = (\mathbf{p}, \xi)^T$ 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 and of value, $R = \text{diag}(5, 5)$.

We assume a known 3×3 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, and define the observation model h in terms of the camera parameters. **Provide the result for A and h in the report. [5 points]**

(b) Implement the noise covariance functions and the observation model in the relevant sections of p1.py. (Look at `system_matrix`, `process_noise_covariance`, `observation_noise_covariance` and `observation` functions). [**5 points**]

(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 the trajectory for 100 time steps. [**5 points**]

(d) **Attach a plot of the generated trajectories and the corresponding measurements. [5 points]**

(e) Find the Jacobian H of the observation model with respect to the fly's state $\mathbf{x}$ and implement your answer of H in function `observation_state_jacobian` in p1.py. [**5 points**]

(f) **Report the Jacobian H. [3 points]**

(g) 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`. [**5 points**]

(h) **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` **[2 points]**

(i) **Discuss the difference in magnitude of uncertainty in the different dimensions of the state. [5 points]**

2 From Monocular to Stereo Vision (35 points)

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 2.

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 and the Jacobian H of the observation model with respect to the fly's state x and **provide the results in the report**. [**5 points**]

(b) Implement the functions `observation` and `observation_state_jacobian` in p2.py. [**5 points**]

(c) What is the new observation noise covariance matrix R? Assume the noise on (u^L, v^L), and (u^R, v^R) to be independent and to have the same distribution as the observation noise given in Question 1, respectively. [**5 points**]

(d) Implement R in function `observation_noise_covariance` [**5 points**]

(e) 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 question. [**5 points**]

(f) **Plot the means 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`. [**2 points**]

(g) **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`. [**5 points**]

(h) For this question, we are defining $\mathbf{z} = (u^L, v^L, d^L)^T$. Alternatively, we could reconstruct the 3D position $\mathbf{p}$ of the fly from its left and right projection (u^L, v^L, u^R, v^R) through triangulation and use $\mathbf{z} = (x, y, z)^T$ directly. **Discuss the pros and cons of using** (u^L, v^L, d^L) **over** (x, y, z) [**3 points**]

3 Linear Kalman Filter with a Learned Inverse Observation Model (25 points)

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 directly observes noisy 3D locations of the ball, we end up with a linear model whose state can be estimated with a Kalman filter. **Note that since you are modifying code from previous parts and are implementing your own outlier detection for part (c), there is no autograder for this problem - we will be grading based on your plots.**

(a) In the folder `data/Q3A_data` you will find a training set of 1000 images in the subfolder `training_set` and the file `Q3A_positions_train.npy` that contains the ground truth 3D position of the red ball in the image. We have provided you with the notebook LearnedObservationModel.ipynb that can be used to train a noisy observation model. As in PSET 3, use this notebook with Google Colab to do this – note that you'll need to upload the data directory onto a location of your choosing in Drive first. Alternatively, if you have an M1+ -chip Mac, you can use the commented line `device = torch.device('mps')` to run locally by uncommenting it (optional). **Report the training and test set mean squared error in your write-up.** [**5 points**]

(b) In the folder `data/Q3B_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 (the logic for this is provided in `PSET4.ipynb`). Now you can work on using the model by completing `p3.py`. 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 (the logic for this is provided in the starter code). [**5 points**]

(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/Q3D_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 in the previous question, run your linear Kalman Filter using the sequence of images as input that are corrupted by occlusions (this is also provided in the notebook `LearnedObservationModel.ipynb`). **Plot the resulting estimated trajectory of the ball over 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.** [**10 points**]

(d) Design an outlier detector and use the data from `data/Q3D_data`. Provide the same plots as in part (c) with `filter_outliers=True`. **Explain how you implemented your outlier detector and add your code to the report. Hint: Your observation model predicts where your measurement is expected to occur and its uncertainty.** [**5 points**]