CS231A: Computer Vision, From 3D Reconstruction to Recognition Homework #4

(Winter 2023) Due: **Monday, March 12**

Overview

This PSET will involve concepts from lectures 12, 13, and 14. It will involve optical and scene flow, Kalman Filters with monocular and stereo vision, and a learned observation model. Although there are 5 problems so this PSET may look daunting, most of the problems require comparatively little work.

Submitting

Please put together a PDF with your answers for each problem, and submit it to the appropriate assignment on Gradescope. We recommend you to add these answers to the latex template files on our website, but you can also create a PDF in any other way you prefer.

The assignment will require you to complete the provided python files and create a PDF for written answers. The instructions are **bolded** for parts of the problem set you should respond to with written answers. To create your PDF for the written answers, we recommend you add your answers to the latex template files on our website, but you can also create a PDF in any other way you prefer. To implement your code, make sure you modify the provided ".py" files in the code folder.

For the written report, in the case of problems that just involve implementing code, you will only need to include the final output (where it is requested) and in some cases a brief description if requested in the problem. There will be an additional coding assignment on Gradescope that has an autograder that is there to help you double check your code. Make sure you use the provided ".py" files to write your Python code. Submit to both the PDF and code assignment, as we will be grading the PDF submissions and using the coding assignment to check your code if needed.

For submitting to the autograder, just create a zip file containing all the files in the 'code' folder. Makes sure to add your finished code to the '.py' files after your code works in colab.

1 Extended Kalman Filter with a Nonlinear Observation Model (30 points)

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,

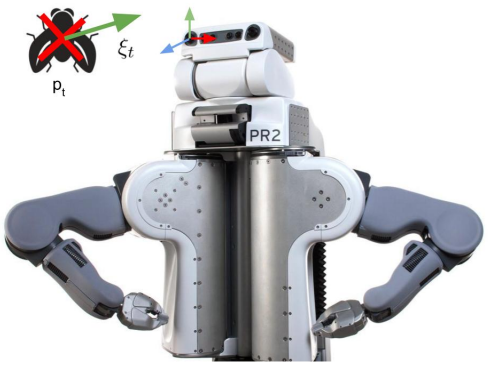

Figure 1

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 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, and implement the noise covariance functions (Implement your answer in the *system_matrix*, *process_noise_covariance*, and *observation_noise_covariance* functions in Q1.py). **[5 points]**

b. Define the observation model h in terms of the camera parameters (Implement your answer in the *observation* function in Q1.py). **[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 for 100 time steps. Attach a plot of the generated trajectories and the corresponding measurements. **[5 points]**

d. Find the Jacobian H of the observation model with respect to the fly's state $\mathbf{x}$. (Implement your answer of H in function *observation_state_jacobian* in Q1.py.)**[5 points]**

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` **[5 points]**

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

2 From Monocular to Stereo Vision (30 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 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). **[6 points]**

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) **[6 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. (Implement R in function *observation_noise_covariance* in Q2.py). **[6 points]**

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 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.**[6 points]**

e. In 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)! [6 points]**

3 Linear Kalman Filter with a Learned Inverse Observation Model (20 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 observes directly 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 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. **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 `LearnedObservationModel.ipynb`). Now you can work on using the model by completing `p4.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). textbfPlot 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. **[5 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]**

4 Tracking with Optical and Scene flow (20 points)

Lastly, a problem having to do with optical and scene flow. Consider the scenario depicted in Fig. 2, where an object O is being interacting by a person, while the robot observes the interaction with its RGB-D camera C. Files `rgb01.png`, `rgb02.png`, ..., `rgb10.png` in the `globe1` folder contain ten RGB frames of the interaction as observed by the robot. Files `depth01.txt`, `depth02.txt`, ..., `depth10.txt` contain the corresponding registered depth values for the same frames.

a. Detect and track the $N = 200$ best point features using Luca-Kanade point feature optical flow algorithm on the consecutive images, initialized with Tomasi-Shi corner features. Use the opencv implementation (look at this tutorial for more information: https://docs.opencv.org/3.4/d4/dee/tutorial_optical_flow.html). For the termination criteria use the number of iterations and the computed pixel residual from *Bouget*, 2001. Use only 2 pyramid

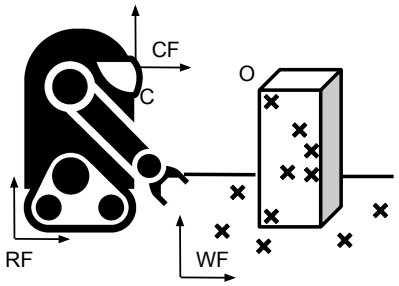

Figure 2

levels and use minimum eigen values as an error measure. **What is the lowest quality level of the features detected in the first frame? Provide a visualization of the flow for window sizes 15×15 and 75×75. Explain the differences observed. Is it always better a larger or a smaller value? What is the trade-off? [5 points]**

Note: You will need the opencv-python package. Please make sure that you have this installed.

b. Use the registered depth maps and the camera intrinsics provided in the starter code to compute the sparse scene flow (i.e. 3D positions of the tracked features between frames). (Note that depth maps contain NaN values. Features that have NaN depth value in any of the frames should be excluded in the result.). **[5 points]**

c. Now let's switch to another scenario of a person opening a book. We have the same format of data as in part(a); the ten RGB frames and the corresponding depth files are available in the `book` folder. Run the algorithm you implemented for part(a) in this new setting and provide a visualization of the flow for block sizes 75×75. **How does the result look qualititavely, compared to the visualization from part(a)? How many features are we tracking in this new scenario and how does it compare to part(a)? Give a brief explanation why it is the case.[5 points]**

d. Next, we will be comparing dense optical flow tracking results from two different methods: Gunnar Farneback's algorithm, and FlowNet 2.0. First, run the provided p4.py script to get the dense optical flow for two frames from the set used in part b, and two frames from the Flying Chairs dataset. The resulting dense optical flow images should be saved to a file titled farneback-(imagename).png; include this in your report.

Describe any artifacts you find between farneback-globe.png and farneback-globe2.png. Does one type of image appear have a better optical flow result? What differences between the image pairs might have caused one optical flow result to appear clearer than the other? [2 points]

e. Next, go to Google Colab and upload the included `FlowNet.ipynb` notebook. Once loading the notebook, follow the included instructions to start a GPU runtime, and upload the respective images from the `p4_data` folder used in the last step (`p4_data/globe2/rgb01.png`, `p4_data/globe2/rgb02.png`, `p4_data/globe1/rgb04.png`, `p4_data/globe1/rgb06.png`, `p4_data/chairs/frame_chairs_1.png`, `p4_data/chairs/frame_chairs_2.png`) to the notebook files tab. Run the cells in the notebook to produce the optical flow between the pairs of images, and include these flow images (and the reconstructed images) in your report. **Note any qualitative differences you see**

between the optical flow results for the different pairs of images. Did one of the optical flow results and reconstructions from a specific pair of the images look less noisier than the others? Describe why you think this might be the case, and if there are any ways in which the performance on the other pairs of images could be improved with the FlowNet model. [3 points]