UNIVERSITY OF VIRGINIA

CS 6501-012: LEARNING IN ROBOTICS

INTRUCTOR - PROF. MADHUR BEHL

HOMEWORK 2

DUE: 03/19/25 WED BY 11.59 PM

Instructions

Read the following instructions carefully before beginning to work on the homework.

- You will submit solutions typeset in LaTeX on Gradescope. You can use hw_template.tex from the course website (under assignments).
- Please start a new problem on a fresh page and mark all the pages corresponding to each problem. Failure to do so may result in your work not graded completely.
- Clearly indicate the name and UVA email ID of all your collaborators on your submitted solutions.
- **For each problem in the homework, you should mention the total amount of time you spent on it. This helps us gauge the perceived difficulty of the problems.**
- You can be informal while typesetting the solutions, e.g., if you want to draw a picture or a figure feel free to draw it on paper clearly, click a picture and include it in your solution. But we cannot accept handwritten solutions. At the same time do not spend undue time just on typesetting solutions.
- On Gradescope you will see an entry of the form "HW 1 PDF" where you will upload the PDF of your solutions. You will also see entries like "HW 1 Problem X Code" where you will upload your solution for the respective problems. **For each programming problem, you should create a fresh Python file**. This file should contain all the code to reproduce the results of the problem and you will upload the .py file to Gradescope. If we have installed Autograder for a particular problem, you will use the Autograder. Name your file to be the same filename as stated in the respective problem statement.
- **You should include all the relevant plots in the PDF if the problem requests them, without doing so you will not get full credit.** You can, for instance, export your Jupyter notebook as a PDF (you can also use text

cells to write your solutions) and export the same notebook as a Python file to upload your code.

- **Your PDF solutions should be completely self-contained. We will run the Python file to check if your solution reproduces the results in the PDF.**

Credit The points for the problems add up to 150. You only need to solve for 130 points to get full credit, i.e., your final score will be min(your total points, 130).

Problem 1 (Extended Kalman Filter, 25 points). In this problem, we will see how we can use filtering to estimate an unknown system parameter. Consider a dynamical system given by

$$\begin{aligned} x_{k+1} &= a x_k + \epsilon_k \\ y_k &= \sqrt{x_k^2 + 1} + \nu_k \end{aligned} \tag{1}$$

where $x_k, y_k \in \mathbb{R}$ are scalars, $\epsilon_k \sim N(0,1)$ and $\nu_k \sim N(0,1/2)$ are zero-mean scalar Gaussian noise uncorrelated across time k. The constant a is unknown and we would like to estimate its value. If we know that our initial state has mean 1 and variance 2

$$x_0 \sim N(1,2),$$

develop the equations for an Extended Kalman Filter (EKF) to estimate the unknown constant a.

(a) **(5 points)** You should first simulate (1) with $a = -1$. This means we plug in the true value of $a = -1$, and generate some observations y_k by simulating the system. This is the ground-truth value of a that we would like to estimate in the next step using filtering. For this part, provide details of how you simulated the system, in particular *how did you sample the noise* ϵ_k, ν_k. The observations $D = \{y_k : k = 1, \dots, \}$ are the "dataset" that we thus collect from the system. Run the simulation for about 100 observations and plot the system i.e. Plot y_k; and x_k at the very least.

(b) **(15 points)** Now you should now develop the EKF equations that will use the collected dataset D to estimate the constant a. Yes, we already know that the true value of $a = -1$, but we want you to estimate a from the data, and treat it like an unknown parameter. If you estimate converges to the true value of a, you can be sure that the EKF is implemented correctly. Show your approach in detail, including any linearization, and Kalman equation steps. Your goal is to compute two quantities

$$\begin{aligned} \mu_k &= \mathrm{E}\left[a_k \mid y_1, \dots, y_k\right] \\ \sigma_k^2 &= \mathrm{var}\left(a_k \mid y_1, \dots, y_k\right). \end{aligned}$$

for all times k.

(c) **(5 points)** Plot the true value $a = -1$, and the estimated values $\mu_k \pm \sigma_k$ as a function of time k. Discuss your result. In particular, do your estimated values $\mu_k \pm \sigma_k$ match the ground-truth value $a = -1$? Does the error reduce as your incorporate more and more observations?

FAQ 1: We advise you to pick the intial guess for a something far from -1 (including picking positive values) and show that as more observations are incorporated you converge towards the true underlying (hidden) value. Showing that regardless of where you picked the initial guess, the filter is doing its job as expected will earn full credit.

Problem 2 (Unscented Kalman Filter, 125 points). In this problem, you will implement an Unscented Kalman Filter (UKF) to track the orientation of a robot in three-dimensions. We have given you observations from an inertial measurement unit (IMU) that consists of gyroscopes and accelerometers and corresponding data from a motion-capture system (called "Vicon", see https://www.youtube.com/watch?v=qgS1pwsHQIA for example). We will develop the UKF for the IMU data and use the vicon data for calibration and tuning of the filter, this is typical of real applications where the robot uses an IMU but the filter running on the robot will be calibrating before test-time in the lab using an expensive and accurate sensor like a Vicon.

(a) **Loading and understanding the data**: First, load the data in the starter set - given on Canvas » Files » Homeworks » hw2 (file "hw2_p2_data.zip") using code of the form.

```
from scipy import io

data_num = 1
imu = io.loadmat('imu/imuRaw'+str(data_num)+'.mat')
accel = imu['vals'][0:3,:]
gyro = imu['vals'][3:6,:]
T = np.shape(imu['ts'])[1]
```

Ignore other fields inside the .mat file, we will not use them.

You can use the following code to load the vicon data

```
vicon = io.loadmat('vicon/viconRot'+str(data_num)+'.mat')
```

while calibrating and debugging.

⚠ **DO NOT load the Vicon data in your Autograder submission because we do not store or provide the vicon data on the server.** This data is provided to you only to assist with calibration and debugging your filter.

(b) **(15 points) Calibrating the sensors**. Check the arrays `accel` and `gyro`. The former gives the observations received by the accelerometer inside the IMU and the latter gives observations from gyroscope. The variable T denotes the total number of time-steps in our dataset. You will have to read the IMU reference manual (file "imu_reference.pdf" inthe hw2 folder on Canvas) to understand the quantities stored in these arrays. Pay careful attention to the following things. First, the accel/gyro readings are integers and not metric quantities, this is because there is usually an analog-to-digital conversion (ADC) that happens in these sensors and one reads off the ADC value as the actual observation. Because of the way these MEMS sensors are constructed, they will have biases and sensitivity with respect to the working voltage. In order to convert from raw values to physical units, the equation for both accel and gyro is typically

$$\text{value} = (\text{raw} - \beta)\,\frac{3300 \text{ mV}}{1023\ \alpha}$$

where β called the bias, mV stands for milli-volt (most onboard electronics operators at 3300 mV) and α is the sensitivity of the sensor. For the accelerometer, α has units of mV/g where g refers to the gravitational acceleration 9.81 m/s^2. In other words, if $\alpha = 100$ mV/g and bias β is zero, and if the raw accelerometer reading is 10, the actual value of the acceleration along that axis is

$$\text{value} = 10 \times \frac{3300}{1023 \times 100} \times 9.81 = 3.16 \text{ m/s}^2$$

Similarly, the sensitivity of a gyroscope has units mV/degrees/sec. You will have to convert the sensitivity into mV/radians/sec to make everything into consistent units.

Typically, in a real application, we do not know the bias and sensitivity of either sensor. Your goal is to use the rotation matrices in the Vicon data as the ground-truth orientation (see section on quaternions below) to estimate the bias and sensitivity of *both* the accelerometer and the gyroscope. You should be careful on two counts.

(1) The orientation of the IMU need not be the same as the orientation of the Vicon coordinate frame. Plot all quantities in the arrays accel, gyro and vicon rotation matrices to make sure you get this right. Do not proceed to implementing the filter if you are not convinced your solution for this part is correct.

(2) The acceleration a_x and a_y is flipped in sign due to device design. A positive acceleration in body-frame will result in a negative number reported by the IMU. See the IMU manual for more insight.

For this problem you should detail how you selected the two constants α, β for both the accelerometer and the gyroscope in your solution PDF. Simply reporting numbers will get zero credit. Also if you just copy paste your code as the "explanation", you will lose points. You need to describe in detail, the process and steps, the logic you used, accompanied with plots and empirical analysis that you conducted. Be as thorough as possible. For e.g. if you plot the Roll, Pitch, Yaw for Accelerometer and Gyro data - clearly label your axes, and include clear legends in the plots. If we cannot understand what you are plotting, or the plots are tiny and not legible, you lose points.

ℹ *Hint:* To find the sensitivity for the accelerometer, we can assume the only force acting is the gravitational force. Then the magnitude of your 3-dimensional accelerometer readings should be as close to 9.81 as possible. Plot the roll, pitch, and yaw values from the Vicon data; you can extract these from the Vicon rotation matrix. You should compare the Vicon plots with some simple plots obtained only from the accelerometer, and separately only the gyroscope values, to predict orientation. From the accelerometer, you can directly compute roll and pitch for each timestep and compare these with ground truth (Vicon) data to ensure your sensitivity and axes are correct. For the gyroscope, you can use the initial orientation from the accelerometer and then integrate the angular velocity values from the gyroscope for the rest of the time series. Note that the orientation estimates you get from this method will have significant drift, but you should be able to get a sense of the scale and check your sensitivity values. The purpose here is to ensure you are converting the raw digital values in the dataset into meaningful physical units before we begin the filtering.

ℹ *Hint:* The time steps are provided in the data (in UNIX format). If they do not line up exactly, you can use closest readings in time as the corresponding readings, or interpolate between successive readings if you want. Either is fine.

(c) **(0 points) Quaternions for orientation** We have given you a file named quaternion.py that implements a Python class for handling quaternions. Read this code carefully. In particular, you should study the function `euler_angle` which returns the Euler angles corresponding to a quaternion, `from_rotm` which takes in a 3×3 rotation matrix and assigns the quaternion and the function `__mul__` which multiplies two quaternions together. Try a few test cases for converting back and forth from a rotation matrix/Euler angles to a quaternion to solidify your understanding here.

(d) **(6 points) Implementing the UKF** Given this setup, you should now read the PDF on Canvas titled "hw2_p2_ukf_writeup" to implement an Unscented Kalman Filter for tracking the orientation using these observations. The state of your filter will be

$$x = \begin{bmatrix} q \\ \omega \end{bmatrix} \in \mathbb{R}^7$$

where q is the quaternion that indicates the orientation and ω is the angular velocity. The observations are of course the readings of the accelerometer and the gyroscope that we discussed above; recall that gyroscopes measure the angular velocity ω itself which simplifies their observation model. The paper also has a magnetometer as a sensor (which measures the orientation with respect to the magnetic north pole) but we will not use it here. You should implement quaternion averaging as described in Section 3.4 of the paper; this is essential for the UKF to work. **You will have to choose yourself the values of the initial covariance of the state, dynamics noise and measurement noise.**

In the PDF explain the steps and choices for how you chose the covariance matrices. What did you try, why did you try what you tried, and how did you settle at the values you finally chose.

(e) **(20 points) Analysis and debugging** Plot the quaternion q (mean and diagonal of covariance), the angular velocity ω (mean and diagonal of covariance), the gyroscope readings in rad/sec and the quaternion corresponding to the vicon orientation as a function of time in your solution PDF.

⚠ **Do not plot on the autograder server, it may crash out.**

You should show the results for the provided datasets and discuss whether your filter is working well. Use these plots to debug your performance; Plotting everything carefully is the fastest way to debug any filter, especially the UKF. Once again, label your plots, and make them big and clear. We would like to see plots of the Roll, Pitch, and Yaw UKF estimates vs the Vicon ground truth, and angular velocities vs the ground truth. We also want to see plots of the effect of the initial covariance noise on the UKF performance. It is likely that the filter will not be equally accurate on all datasets, this is typical of real world performance. The goal is to use the calibration ground truth to tune the covariances of the filter to our best ability.

(f) **(84 points) Evaluation** We have setup an Autograder for this problem. We will test the performance of your filter on the datasets provided to you as well as some of our held-out datasets. Make sure you submit **estimate_rot.py** as well as all its dependencies (if any). This function should return three Numpy arrays of length T, one each for **Euler angles** (roll, pitch, yaw) for the orientation.

FAQ 1: Every axis for both the accelerometer and gyro can have its own distinct bias and sensitivity value.

FAQ 2: The IMU measurements are expressed in the body frame, which rotates with the drone. The Vicon coordinate system is fixed.