UNIVERSITY OF PENNSYLVANIA

ESE 650: LEARNING IN ROBOTICS

SPRING 2023

[02/07] HOMEWORK 2

DUE: 02/27 MON 11.59 PM ET

Changelog:

- **2/13 7pm: pg 6 lines 5-7 on using the auto-calibration method on training to get hard-coded values for the test set.**
- **2/19 6pm: UKF paper (A Quaternion-based Unscented Kalman Filter for Orientation Tracking) - Sec 2.3 Eq 27 and 28:**
  These are the corrected equations for the measurement model:

$$g' = q_k^{-1} g q_k$$
$$b' = q_k^{-1} b q_k$$

- **2/19 6pm: Pg 6 Line 10: Gyroscope: sensitivity $\alpha \sim$ 200 mV/(rad/sec)**
- **2/27 4pm: Page 9 Line 26: The final $\bar{q}$ is the mean of the Gaussian represented by the transformed sigma points and the covariance Cov$(e_i) = (2n)^{-1} \sum_i e_i e_i^\top$.**

Instructions

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

- You will submit solutions typeset in LaTeX on Gradescope (strongly encouraged). You can use hw_template.tex on Canvas in the "Homeworks" folder to do so. If your handwriting is *unambiguously legible*, you can submit PDF scans/tablet-created PDFs.
- 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 Penn 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 feel free to draw it on paper clearly, click a picture and include it in your solution. Do not spend undue time on typesetting solutions.
- You will see an entry of the form "HW 2 PDF" where you will upload the PDF of your solutions. You will also see entries like "HW 2 Problem 1 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 "pennkey_hw2_problem1.py", e.g., I will name my code for Problem 1 as "pratikac_hw2_problem1.py".
- **You should include all the relevant plots in the PDF, 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 125. You only need to solve for 100 points to get full credit, i.e., your final score will be $\min(\text{your total points}, 100)$.

Problem 1 (Extended Kalman Filter, 25 points). In this problem, we will see how to 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 is the ground-truth value of a that we would like to estimate. Provide details of how you simulated the system, in particular how you sampled the noise ϵ_k, ν_k. The observations $D = \{y_k : k = 1, \ldots, \}$ are the "dataset" that we thus collect from the system. Run the simulation for about 100 observations.

(b) **(15 points)** You should now develop the EKF equations that will use the collected dataset D to estimate the constant a. Discuss your approach in detail. Your goal is to compute two quantities

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

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? Argue why/why not.

Problem 2 (Unscented Kalman Filter, 100 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 a gyroscope (to measure angular velocity) and an accelerometer (to measure accelerations in body frame) as well as data from a motion-capture system called "Vicon", see https://www.youtube.com/watch?v=qgS1pwsHQIA which is a set of cameras that track an object in an indoor environment. While the estimates of the position or rotation obtained from the IMU will be noisy, a Vicon is extremely accurate (with errors of the order of a few millimeters for the position and fractions of a degree for the orientation). We can therefore treat the Vicon data as the "ground-truth". We will develop the UKF for the IMU data and use the Vicon data for calibration and tuning of this 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) **Understanding the data**: First, load the data given on Canvas (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. But do not include this line in the autograder submission because we do not store the Vicon data in the autograder.

(b) **(15 points) Calibrating the sensors**. Check the arrays named “accel” and “gyro”. The former gives the observations received by the accelerometer inside the IMU and the latter gives observations from the 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” on Canvas) to understand the quantities stored in these arrays. Pay careful attention to the following thing.

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. The 1023 in the denominator comes because in our sensor there is a 10-bit ADC that was being used.

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

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

The equation for the gyroscope is similar. The sensitivity of a gyroscope has units mV/(degrees/sec). Remember to convert the output into radians/sec before you use the gyroscope readings to update the filter. You can also calculate the sensitivity in mV/(rad/sec) then your output for the angular velocity will be in radians/sec. **There is a bias and sensitivity for each of the three axes for both the accelerometer and the gyroscope; these quantities will typically be similar for all the three axes but you will get more accurate estimates if you tune them individually a bit.**

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. While doing so, 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.

How to calibrate the accelerometer? 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. Next 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 (you will calibrate the gyroscope separately as detailed below) to predict the orientation. From the accelerometer, you can directly compute roll and pitch for each timestep by looking at the angle with respect to gravity (which always points downwards); compare these with the Vicon roll and pitch to ensure your sensitivity (the hard part here is to make sure your axes are correct).

How to calibrate the gyroscope? 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. 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. A better strategy is to differentiate the orientation obtained from the Vicon data to obtain the true angular velocity and estimate the bias and sensitivity of the gyroscope by comparing its readings to this true angular velocity.

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. Calibrating the sensors is a non-trivial step and even if your filtering code is completely correct, you will not get accurate estimates if your calibration is off. The Autograder will execute your filter on held-out datasets that we have created. To do well on these test dataset, use you automatic calibration method to extract good values from the "training set" and hard-code those values for the test set. As a hint, we have given you the rough range of the calibration constants.

- Accelerometer: bias $\beta \sim 500$, sensitivity $\alpha \sim 25$–50 mV/(m/s^{-2})
- Gyroscope: bias $\beta \sim 350$, sensitivity $\alpha \sim 250$ mV/(rad/sec)

(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 to-and-fro from a rotation matrix/Euler angles to a quaternion to solidify your understanding here.

(d) **(0 points) Implementing the UKF** Given this setup, you should next read the Appendix of this homework PDF before implementing the Unscented Kalman Filter for tracking the orientation of the quadrotor. 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 directly. 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 the values of the initial covariance of the state, dynamics noise and measurement noise yourself.** You should discuss your steps and choices in your solution PDF.

(e) **(10 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 server, it may crash out. You should show the results for one dataset and discuss whether your filter is working well. You should also use these plots to debug your performance on the other datasets; plotting everything carefully is the fastest way to debugging the UKF.

(f) **(75 points) Evaluation** We will use the 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. This function should return three Numpy arrays of length T, one each for **Euler angles** (roll, pitch, yaw) for the orientation.

Appendix A. Implementing the UKF

This Appendix will help you understand the differences in the notation between the course notes (denoted as PC) and Edgar Kraft's paper (denoted as EK) on implementing the UKF using quaternions posted on Canvas. The state at time k in the notes is x_k while the next state is x_{k+1}. EK defines the previous state x_{k-1} and the current state as x_k. We will use the convention in PC. The UKF is the same and its basic steps are:

(i) propagate the dynamics,
(ii) obtain an observation from the accelerometer and the gyroscope,
(iii) compute the Kalman gain, and
(iv) update the mean and covariance using the latest observation.

State. The estimate of the state is given by the mean and covariance (PC p.59, 64). Since the state $x = (q, \omega) \in \mathbb{R}^7$ consists of the quaternion and the angular velocities, the mean of our estimate of the state is $\mu_{k|k} \in \mathbb{R}^7$. In the code, you will create a tuple as a concatenation of an object of the Quaternion class and some initial values for the angular velocity; this way you can use the methods in the Quaternion class to perform operations on the first 4 elements. The quaternion always has a unit magnitude so the covariance is not a 7×7 matrix but instead

$$\Sigma_{k|k} \in \mathbb{R}^{6\times 6}$$

You will initialize the covariance to be positive definite, e.g., with positive entries on the diagonal and zeros otherwise. See EK sec 3.2 now to get a better idea of the dimensions.

The Unscented Transform (UT). As we have discussed in the lectures, the Unscented Transform (UT) uses sigma points to compute an approximation of the probability distribution of a random variable $y = f(x)$ given the distribution of the random variable x. In simple words, this amounts to using the Gaussian $N(\mu_{k|k}, \Sigma_{k|k})$ or $N(\mu_{k+1|k}, \Sigma_{k+1|k})$ to generate the sigma points, apply the function $f(\cdot)$ or $g(\cdot)$ (corresponding to the dynamics or measurement equations respectively) to each of these sigma points and then recomputing the mean and covariance of the transformed sigma points (PC p.59, sec 3.7.1).

Generating sigma points (PC Sec 3.7.1, EK Sec 3.1-3.2) Let us generate sigma points from a Gaussian $x \sim N(\mu, \Sigma)$. Remember that $\mu \equiv (\mu_q, \mu_\omega) \in \mathbb{R}^7$ consists of two parts, the quaternion part of the estimate and the estimate for angular velocity.

Using PC eqn 3.29, we will create $2n$ sigma points where n is the number of columns in Σ, e.g. $n = 6$ for our problem. We need to calculate the square root of the covariance $\sqrt{\Sigma}$. The notes describe a method using diagonalization while EK uses a Cholesky decomposition.

For the homework you can simply call scipy's linear algebra matrix square root; see https://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.sqrtm.html.

The n column vectors of $\sqrt{\Sigma}$ matrix are multiplied by $\pm\sqrt{n}$ and form the set $\left\{x^{(i)}\right\}_{i=1}^{2n}$ to get $2n$ elements (corresponding to positive and negative multipliers). As shown in the notes Eqn 3.29, we would now like to add these things to the mean μ to obtain the sigma points. However, the state in our problem is a concatenation of the quaternion and the angular velocities so it is slightly more complicated (EK sec 3.2). We cannot "add" two quaternions using the standard rules of summation. In any case, the vectors we created from $\sqrt{\Sigma}$ are 6-dimensional and the mean $\mu \in \mathbb{R}^7$ so we cannot even add them...

We will therefore transform the first three elements of each vector $x^{(i)}$ into quaternion space and obtain the representation in the axis-angle form (EK calls this the vector quaternion in Eqn. 26). The transformed version is now a legitimate quaternion and can be "added" to the first 4 elements of μ using the quaternion multiply operation; just like two translations are added to get the total translation, two quaternions which represent rotation one after the other are multiplied together to get the total rotation. Multiplication of quaternions is not commutative, but in this case it does not matter whether we do $\mu_q\, {x^{(i)}}_q$ or ${x^{(i)}}_q\, \mu_q$ because half of the vectors are negative and correspond to the opposite rotation. We do not need to do anything special for the angular velocity and can use the standard rules of summation to add μ_ω with the lower three elements of ${x^{(i)}}_\omega$.

Recompute the Gaussian from sigma points We need to use eqn. 3.31 in the notes to estimate the empirical mean and covariance of the transformed sigma points. Again in this problem, this is not straightforward because we are using quaternions. More details on this will be given in the process/dynamics and measurements sections below.

Propagating the dynamics. We will implement Eqn. 3.32 or EK Eq 35 in a slightly different way. We will add the covariance R to $\Sigma_{k|k}$ before calculating the sigma points. This does not change anything, it just expands the Gaussian and then transforms it. The unscented transform is not exact because it approximates a general probability distributed using a Gaussian (it just does a better job of this approximation...). In a filtering problem it is always better to err towards a larger covariance that will be shrunk by the observations when they come instead of using an incorrect but small covariance.

Another important point is to notice that the time interval between two readings from the accelerometer and gyroscope is not fixed in the dataset. It is variable in a real system because, e.g., sometimes the CPU can take a few extra cycles to read the data from the sensor. We will therefore think of the dynamics covariance corresponding to the continuous time dynamics as R and always multiply by $\mathrm{d}t$ of the specific time-step to get the discrete-time covariance. Altogether, first calculate

$$\Sigma_{k|k} + R\,\mathrm{d}t$$

and then use this to compute the sigma points and transform them (EK Eqn. 37). The transformation is deterministic because we have already incorporated the dynamics noise.

The next step is to compute Eqn. 3.32 in the notes to obtain $\mu_{k+1|k}$ and $\Sigma_{k+1|k}$ from the transformed sigma points. As Edgar notes in his paper (EK sec 3.4), the mean of the orientations of the sigma points is not equal to correct mean of the rotations represented by these quaternions (as we said above, we should not be doing an algebraic sum of two quaternions, summation for Euclidean vectors corresponds to multiplication for quaternions). EK therefore develops a procedure to calculate the correct mean of the quaternion part of the state using gradient descent. **Note that this is only for the quaternion portion of the state**.

Gradient descent to compute the mean and covariance of the quaternion part of the sigma points (EK Sec. 3.4–3.5) Initialize a quaternion $\bar{q}$ and a matrix $E \in \mathbb{R}^{3\times 2n}$ for the $2n$ sigma points. First initialize an estimate of mean $\in \mathbb{R}^7$ which we will iteratively improve through GD and an error vector matrix E $\in \mathbb{R}^{3,2n}$. You can initialize $\bar{q}$ to the quaternion part of the previous state $(\mu_{k|k})_q$ which will help gradient descent converge quickly. We will perform the following iterations.

(i) Compute the error vectors $e_i \in \mathbb{R}^3$ for $i = \{1, \dots, 2n\}$ for every sigma point. Fill in the matrix E using these error vectors. The error e_i is the relative rotation between the sigma point $x_q^{(i)}$ and the current estimate of the mean $\bar{q}$ from the previous estimate of gradient descent. This can be computed using quaternion math (Ref. EK Sec 3.4, Eq. 52-53) and then can be converted to axis-angle (3d vector) representation to store in our maintained matrix E.

(ii) Compute the standard mean of the error vectors $\bar{e} = (2n)^{-1} \sum_i e_i$ and convert it into the quaternion space; this is the new mean $\bar{q}$ (EK sec 3.4, eq 55).

(iii) Iterate upon the above two steps until the magnitude of $\bar{e}$ falls below a threshold.

(iv) The final $\bar{q}$ is the mean of the Gaussian represented by the transformed sigma points and the covariance $\text{Cov}(e_i) = (2n)^{-1} \sum_i e_i e_i^\top$.

The estimate of the mean and covariance for the angular velocity, which is a Euclidean vector, are obtained in the standard way.

Altogether, this step will compute $\mu_{k+1|k} \in \mathbb{R}^7$ and $\Sigma_{k+1|k} \in \mathbb{R}^{6\times 6}$.

Measurement update. First obtain the calibrated data from the accelerometer (for the orientation) and gyroscope (angular velocity) and then compute the following steps which are explained below.

(i) Generate sigma points (use the same function from the dynamics step)

(ii) Propagate sigma points using the measurement model (EK 3.7.7)

(iii) Compute a new mean, covariance (Σ_{yy}), and cross covariance (Σ_{xy}) from sigma points (3.7.8); make sure that you add the measurement noise Q to Σ_{yy})

(iv) Calculate innovation (EK 3.7.9)

(v) Compute the Kalman gain (EK 3.7.11) (PC Eqn 3.36, PC Step 2.2 in Sec 3.7.3) and update the estimate to obtain $\mu_{k+1|k+1}$ and $\Sigma_{k+1|k+1}$.

Propagate sigma points with the measurement model First obtain the calibrated accelerometer and gyroscope readings in metric units (m sec^{-2} and rad sec^{-1} respectively). We will use the procedure in EK Sec 2.3 using our previous estimate of the state $\mu_{k+1|k}$ and $\Sigma_{k+1|k}$. This converts the gravity vector into the frame of reference of the quadrotor using our current estimate of the orientation and thereby the reading of the accelerometer can be thought of just as a vector in $\mathbb{R}^3$ which is an observation for such a transformed vector, up to some noise. Note that this assumes that the quadrotor is not accelerating. The gyroscope measures the second part of the state (the angular velocities) directly, up to some noise of course. Use some reasonable values for the covariance of the noise on the accelerometer and gyroscope observation model. Since we are now working in standard Euclidean space of accelerations and angular velocities the sigma point transform is computed using exactly the equations in PC Sec 3.7.1 Step 2.1). In particular, the mean in PC Eqn 3.33 is just the Euclidean mean. Next compute the covariance Σ_{yy} and add the measurement noise as a diagonal matrix Q (this is a tunable parameter like dynamics noise R that you should pick). Compute the cross covariance Σ_{xy} using PC Sec 3.34.

Compute Innovation Use the observation y_{k+1} and the estimated measurement $\hat{y}$ (the mean of the sigma points from the previous step) to compute the innovation

$$\text{innovation} = y - \hat{y}.$$

Compute the Kalman gain and the updated estimate (PC Eqn 3.36) using

$$K = \Sigma_{xy}\Sigma_{yy}^{-1}$$

The mean of the updated estimate $\mu_{k+1|k+1}$ is computed as follows. First convert $K(y - \hat{y})$ back into quaternion space for the quaternion part; the angular velocity part remains unchanged. We now use the standard equation

$$\mu_{k+1|k+1} = \mu_{k+1|k} + K\text{innovation}.$$

The covariance is

$$\Sigma_{k+1|k+1} = \Sigma_{k+1|k} - K\Sigma_{yy}K^{\top}.$$

You should convert the orientation part of $\mu_{k+1|k+1}$ into Euler angles for comparison to the Vicon data. Now loop through all the observations and you're done.