TTK4250 Sensor Fusion Assignment 3

Hand in: *Friday 19. September 23:59* on Blackboard.

Read Python setup guide on BB, it can be found under *Course work*.
This assignment should be handed in on Blackboard, as a PDF pluss **the zip created by running *create_handin.py***, before the deadline.
You are supposed to show how you got to each answer unless told otherwise.
If you struggle, we encourage you to ask for help from a classmate or come to the exercise class on Tuesday.

Task 1: *Bayesian estimation of an existence variable*

We are back at tracking the number of boats in a region. However, you now know that there is at most one boat in the region, and that it is there with probability $r_k \in (0,1)$ at time step k. The boat will stay in the region with probability P_S and leave with probability $1 - P_S$ if it is there. Otherwise, it may enter with probability P_E and not enter with probability $1 - P_E$ if it is not in the region. You have an unreliable measurement coming from a radar that detects a present boat with probability P_D, and may not detect a present boat with probability $1 - P_D$. If it did not detect a present boat, it may declare that there is a boat present anyway, termed a false alarm, with probability P_{FA}. The task is to apply a Bayesian filter to this problem.

Hint:: It might ease proper book keeping if you start by denoting the events with variables and use their (possibly conditional) pmfs before you insert the given probabilities.

(a) Apply the Bayes prediction step to get the predicted probability $r_{k+1|k}$ in terms of r_k, P_S and P_E.

Hint: Introducing the events at time step k as "in region" R_k,"staying" S_k and "entering" E_k, where $S_{k+1} = R_{k+1}|R_k$ and $E_{k+1} = R_{k+1}|\neg R_k$ with $\neg$ denoting negation, can be helpful.

(b) Apply the Bayes update step to get posterior probability for the boat being in the region, r_{k+1}, in terms of $r_{k+1|k}$, P_D and P_{FA}. That is, condition the probability on the measurement. There are two cases that needs to be considered; receiving a detection and not receiving a detection.

Hint: Introduce the events $M_{k+1} = D_{k+1} \cup F_{k+1}$ to denote the sensor declaring a present boat, with D_{k+1} denoting the event of detecting a present boat and F_{k+1} denoting the event of a false alarm. We then for instance have $\Pr(D_{k+1}|\neg R_{k+1}) = 0$ and $\Pr(F_{k+1}|D_{k+1}) = 0$ from the problem setup.

Task 2: *KF initialization of CV model without a prior*

The KF typically uses a prior for initializing the filter. However, in target tracking we often have no specific prior and would like to infer the initialization of the filter from the data. For the CV model (see chapter 4) with positional measurements, the position is observable with a single measurement, while the velocity needs two measurements to be observable (observable is here used in a statistical sense to mean that there is information about the state from the measurements).

With $x_k = \begin{bmatrix} p_k^T & u_k^T \end{bmatrix}^T$, where p_k is the position and u_k is the velocity at time step k, you should recognize the CV model as

$$x_{k+1} = \begin{bmatrix} p_{k+1} \\ u_{k+1} \end{bmatrix} = Fx_k + v_k,$$

with $v_k \sim \mathcal{N}(0, Q)$ and F and Q as defined in (4.64) in the book. The measurement model is given by $z_k = \begin{bmatrix} I_2 & 0_2 \end{bmatrix} x_k + w_k = p_k + w_k$ and $w_k \sim \mathcal{N}(0, R) = \mathcal{N}(0, \sigma_r^2 I_2)$.

Since the KF is linear, we would like to use a linear initialization scheme that uses two measurements and the model parameters. That is

$$\hat{x}_1 = \begin{bmatrix} \hat{p}_1 \\ \hat{u}_1 \end{bmatrix} = \begin{bmatrix} K_{p_1} & K_{p_0} \\ K_{u_1} & K_{u_0} \end{bmatrix} \begin{bmatrix} z_1 \\ z_0 \end{bmatrix}. \tag{1}$$

(a) Write z_1 and z_0 as a function of the noises, true position and speed, p_1 and u_1, using the CV model with positional measurements. Use v_k to denote the process disturbance and w_k to denote the measurement noise at time step k, and T for the sampling time between $k-1$ and k.

Hint: A discrete time transition matrix is always invertible, and it is easy to find for the CV model: remove the process noise and write out the transition as a system of linear equations, solve the system for the inverse and rewrite it as a matrix equation again.

(b) Show that to get an unbiased initial estimate, the initialization gain matrix must satisfy $K_{p_1} = I_2$, $K_{p_0} = 0_2$, $K_{u_1} = \frac{1}{T} I_2$ and $K_{u_0} = -\frac{1}{T} I_2$, where T is the sampling time. That is, find the $K_\times$ so that $E[\hat{x}_1] = x_1 = \begin{bmatrix} p_1 & u_1 \end{bmatrix}^T$

Note: To find estimator biases, one fixes the values to be estimated and do not treat them as random variables.

(c) What is the covariance of this estimate?

(d) You have used this initialization scheme for your estimator and found a mean and covariance. What distribution does the true state have after this initialization? What are its parameters.

Hint: From equations you have already used, you can write x_1 in terms of $\hat{x}$ and disturbances and noises. You should be able to see the result as a linear transformation of some random variables. Note that $\hat{x}$ is given since the measurements are given and thus can be treated as a constant. x_1 is now treated as a random variable as opposed to when finding the mean and variance of the estimator.

(e) In theory, would you say that this initialization scheme is optimal or suboptimal, given that the model and two measurements is all we have? What would you say about its optimality in practice?

(f) Use the results from c) to finish implementing `initialize.get_init_CV_state`.

Task 3: *Make CV dynamic model and position measurement model*

Finish the Python classes `WhitenoiseAcceleration2D` that implements the continous velocity (CV) model and `CartesianPosition2D` that implements positional measurements so that they can be used in an (E)KF. Even though these particular models are linear, we are going to implement them as if they were more general. The skeletons can be found in the files `dynamicmodels.py` and `measurementmodels.py` in the zip file on Blackboard.

You have the standard model for a KF

$$x_k = F_{k-1} x_{k-1} + v_{k-1}, \qquad v_{k-1} \sim \mathcal{N}(0; Q) \tag{4}$$
$$z_k = H_k x_k + w_k, \qquad w_k \sim \mathcal{N}(0, R) \tag{5}$$

(a) Finish the implementation of `WhitenoiseAcceleration2D.f`

(b) Finish the implementation of `WhitenoiseAcceleration2D.F`

(c) Finish the implementation of `WhitenoiseAcceleration2D.Q`

(d) Finish the implementation of `WhitenoiseAcceleration2D.predict_state`

(e) Finish the implementation of `CartesianPosition2D.h`

(f) Finish the implementation of `CartesianPosition2D.H`

(g) Finish the implementation of `CartesianPosition2D.R`

(h) Finish the implementation of `CartesianPosition2D.predict_measurement`

Task 4: *Implement EKF i Python*

In Python: Finish the functions marked with 'TODO' in `ekf.py` skeleton found in the zip file on blackboard. The EKF class will be initialized with a measurement model and a dynamic model of the form you made in the last task. You can therefore assume that the methods (or some with the same interface) are available in `self.dynamic_models` and `self.measurement_model` when implementing `EKF`.

You are free to change the prewritten code (for instance to optimize or make things clearer for your self), but the function definitions must be the same for evaluation purposes.

Task 5: *Analyze filter output*

(a) Finish the implementation of `get_nis`

(b) Finish the implementation of `get_nees`

(c) Try out different parameters for the KF and the trajectory simulator to get an intuition of how the different parameters affect the output trajectory and NIS and NEES values. All the tunable values are found in `tuning.py` . You don't have to answer anything here.

> *Hint:* You can also change the `DEBUG` parameter in `config.py` to speed up the simulation.

(d) For each of the five requirements under (4.6.1) in the book, give a brief explanation of why the requirement is important and how one can check if a filter satisfies it.