TTK4250 - Sensor Fusion Assignment 2

Martin Eek Gerhardsen

11. September 2019

Department of Engineering Cybernetics

Contents

1 Task 1: Bayesian estimation of an existence variable

1.1 a)

Defining $x_k \rightarrow$ measure the boat, and $z_k \rightarrow$ decide the boat was measured. Also worth noting that the there are two possible choices for the state, measured or not measured. Then:

$$\begin{aligned}
r_{k+1|k} &= p(x_{k+1}|z_{1:k}) = \int p(x_{k+1}, x_k|z_{1:k})dx_k \\
&= \int p(x_{k+1}|x_k)p(x_k|z_{1:k})dx_k \\
&= \sum_{x_k\{boat,no-boat\}} p(x_{k+1}|x_k)p(x_k|z_{1:k}) \\
&= P_S r_k + P_E(1-r_k) = P_E + (P_S - P_E)r_k
\end{aligned}$$

1.2 b)

Then, using the update step:

$$\begin{aligned}
r_{k+1} &= \frac{p(z_{k+1}|x_{k+1})p(x_{k+1}|z_{1:k})}{p(z_{k+1}|z_{1:k})} \\
&= (P_D + (1-P_D)P_{FA})r_{k+1|k}
\end{aligned}$$

2 Task 2: KF initialization of CV model without prior knowledge

Defining:

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad\qquad O = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

2.1 a)

Defining the CV-model:

$$\begin{aligned} \dot{x} &= Ax + Gn \\ n &\sim \mathcal{N}(0, D\delta(t-\tau)) \end{aligned}$$

With:

$$\begin{aligned} A &= \begin{bmatrix} O & I \\ O & O \end{bmatrix} \qquad\qquad G = \begin{bmatrix} O \\ I \end{bmatrix} \\ D &= \sigma_a^2 I \end{aligned}$$

Then we discretize (I'll also simplify things like $t_k - t_{k-1} = T$):

$$\begin{aligned} x_k &= Fx_{k-1} + v_k \\ F &= e^{AT} = I_{4x4} + AT + 0 \\ &= \begin{bmatrix} I & TI \\ O & I \end{bmatrix} \\ v_k &= \int_{t_{k-1}}^{t_k} e^{AT} Gn(\tau) d\tau \end{aligned}$$

For future use, the inverse of F is simply:

$$F^{-1} = \begin{bmatrix} I & -TI \\ O & I \end{bmatrix}$$

Further defining:

$$\begin{aligned} \hat{x}_1 &= \begin{bmatrix} \hat{p}_1 \\ \hat{u}_1 \end{bmatrix} = \begin{bmatrix} K_{p_1} & K_{p_0} \\ K_{u_1} & K_{u_0} \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \\ z_k &= \begin{bmatrix} I & O \end{bmatrix} x_k + w_k = p_k + w_k \\ x_k &= \begin{bmatrix} p_k^\top & u_k^\top \end{bmatrix}^\top \\ w_k &\sim \mathcal{N}(0, R) \end{aligned}$$

Inserting:

$$\begin{aligned}
z_k &= \begin{bmatrix} I & O \end{bmatrix} x_k + w_k \\
x_1 &= F x_0 + v_0, \qquad\qquad x_0 = F^{-1}(x_1 - v_0) \\
z_0 &= \begin{bmatrix} I & O \end{bmatrix} F^{-1}(x_1 - v_0) + w_0 \\
&= \begin{bmatrix} I & -TI \end{bmatrix} (x_1 - v_0) + w_0 \\
&= p_1 - Tu_1 - \begin{bmatrix} I & -TI \end{bmatrix} v_0 + w_0 \\
z_1 &= \begin{bmatrix} I & O \end{bmatrix} x_1 + w_1 \\
&= p_1 + w_1
\end{aligned}$$

Note that z_0 is the position at $k = 1$ minus the timestep multiplied with the velocity, or the distance travelled in that timestep.

2.2 b)

Simplifying the estimate as:

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

$$= \begin{bmatrix} K_{p_1} & K_{p_0} \\ K_{u_1} & K_{u_0} \end{bmatrix} \begin{bmatrix} p_1 + w_1 \\ p_1 - Tu_1 - \begin{bmatrix} I & -TI \end{bmatrix} v_0 + w_0 \end{bmatrix} \tag{2}$$

Using the fact that finding the expected value can be applied linearly, we simplify away all noise (as their expected value is assumed zero):

$$\begin{aligned}
E[\hat{x}_1] &= \begin{bmatrix} E[K_{p_1} p_1 + K_{p_0}(p_1 - Tu_1)] \\ E[K_{u_1} p_1 + K_{u_0}(p_1 - Tu_1)] \end{bmatrix} \\
&= \begin{bmatrix} (K_{p_1} + K_{p_0}) p_1 - TK_{p_0} u_1 \\ (K_{u_1} + K_{u_0}) p_1 - TK_{u_0} u_1 \end{bmatrix} = \begin{bmatrix} p_1 \\ u_1 \end{bmatrix}
\end{aligned}$$

Therefore, we may conclude that:

$$\begin{aligned}
K_{p_1} &= I_2 & K_{p_0} &= O_2 \\
K_{u_1} &= \frac{1}{T} I_2 & K_{u_0} &= -\frac{1}{T} I_2
\end{aligned}$$

or

$$K = \begin{bmatrix} I_2 & O_2 \\ \frac{1}{T} I_2 & -\frac{1}{T} I_2 \end{bmatrix} \tag{3}$$

will give an unbiased estimate.

2.3 c)

Finding the Q matrix as defined in Theorem 4.5.1 in the textbook, [1, page 60]:

$$
\begin{aligned}
Q = E[v_k v_k^\top] &= \int_0^T e^{(T-\tau)A} G D G^\top e^{(T-\tau)A^\top} d\tau \\
&= \int_0^T \begin{bmatrix} I & (T-\tau)I \\ 0 & I \end{bmatrix} \begin{bmatrix} 0 \\ I \end{bmatrix} \sigma_a^2 I \begin{bmatrix} 0 & I \end{bmatrix} \begin{bmatrix} I & 0 \\ (T-\tau)I & I \end{bmatrix} d\tau \\
&= \int_0^T \begin{bmatrix} (T-\tau)I \\ I \end{bmatrix} \sigma_a^2 I \begin{bmatrix} (T-\tau)I & I \end{bmatrix} d\tau \\
&= \int_0^T \sigma_a^2 \begin{bmatrix} (\tau^2 - 2T\tau + T^2)I & (T-\tau)I \\ (T-\tau)I & I \end{bmatrix} d\tau \\
&= \sigma_a^a \begin{bmatrix} \frac{T^3}{3}I & \frac{T^2}{2}I \\ \frac{T^2}{2}I & TI \end{bmatrix}
\end{aligned}
$$

Then simplifying eq. (2) by removing the constants and inserting eq. (3), and using the rules for linear combinations of covariance matrices we find:

$$
\begin{aligned}
Var[\hat{x}_1] &= \begin{bmatrix} Var[w_1] & Cov(z_0, z_1) \\ Cov(z_0, z_1) & Var[\frac{1}{T}w_1 - \frac{1}{T}w_0 + u_1 + \frac{1}{T}\begin{bmatrix} I & -TI \end{bmatrix} v_0] \end{bmatrix} \\
&= \begin{bmatrix} R & Cov(z_0, z_1) \\ Cov(z_0, z_1) & \frac{2}{T^2}R + \frac{1}{T^2}\begin{bmatrix} I & -TI \end{bmatrix} Q \begin{bmatrix} I \\ -TI \end{bmatrix} \end{bmatrix}
\end{aligned}
$$

Calculating the result from indroducing the Q matrix, as calculated above:

$$
\begin{aligned}
\frac{1}{T^2}\begin{bmatrix} I & -TI \end{bmatrix} Q \begin{bmatrix} I \\ -TI \end{bmatrix} &= \frac{\sigma_a^2}{T^2}\begin{bmatrix} I & -TI \end{bmatrix} \begin{bmatrix} \frac{T^3}{3}I & \frac{T^2}{2}I \\ \frac{T^2}{2}I & TI \end{bmatrix} \begin{bmatrix} I \\ -TI \end{bmatrix} \\
&= \frac{\sigma_a^2}{T^2}\begin{bmatrix} I & -TI \end{bmatrix} \begin{bmatrix} \frac{T^3}{3}I - \frac{T^3}{2}I \\ \frac{T^2}{2}I - T^2 I \end{bmatrix} \\
&= \frac{\sigma_a^2}{T^2}(-\frac{T^3}{6} + \frac{T^3}{2}I) = \frac{\sigma_a^2 T}{3}I
\end{aligned}
$$

Noting that $Cov(a, b) = 0$ for $a, b \in \{w_k, v_k\}$, we calculate $Con(z_0, z_1)$:

$$
\begin{aligned}
Con(z_0, z_1) &= Con(z_1, z_0) = E[(z_0 - E[z_0])(z_1 - E[z_1])] \\
&= E[(p_1 + w_1 - E[p_1 + w_1]) \\
&(\frac{1}{T}w_1 - \frac{1}{T}w_0 + u_1 + \frac{1}{T}\begin{bmatrix} I & -TI \end{bmatrix} v_0 \\
&- E[\frac{1}{T}w_1 - \frac{1}{T}w_0 + u_1 + \frac{1}{T}\begin{bmatrix} I & -TI \end{bmatrix} v_0])] \\
&= E[w_1(\frac{1}{T}w_1 - \frac{1}{T}w_0 + \frac{1}{T}\begin{bmatrix} I & -TI \end{bmatrix} v_0)] \\
&= \frac{1}{T}E[w_1^2] - \frac{1}{T}[w_0 w_1] + \frac{1}{T}\begin{bmatrix} I & -TI \end{bmatrix} E[v_0 w_1] \\
&= \frac{1}{T}(Var[w_1] - E[w_1]^2) - \frac{1}{T}(Cov(w_0, w_1) + E[w_0]E[w_1]) \\
&+ \frac{1}{T}\begin{bmatrix} I & -TI \end{bmatrix} (Cov(v_0, w_1) + E[v_0]E[w_1]) \\
&= \frac{1}{T}R
\end{aligned}
$$

Then, we may find the covariance matrix of the estimate as:

$$
\begin{bmatrix} R & \frac{1}{T}R \\ \frac{1}{T}R & \frac{2}{T^2}R + \frac{\sigma_a^2 T}{3}I \end{bmatrix}
$$

2.4 d)

Solving for x_1

$$
\begin{aligned}
\hat{x}_1 &= \begin{bmatrix} I_2 & 0_2 \\ \frac{1}{T}I_2 & -\frac{1}{T}I_2 \end{bmatrix} \begin{bmatrix} z_1 \\ z_0 \end{bmatrix} = \begin{bmatrix} p_1 + w_1 \\ p_1 - Tu_1 - \begin{bmatrix} I_2 & -TI_2 \end{bmatrix} v_0 + w_0 \end{bmatrix} \\
&= \begin{bmatrix} I_2 & O_2 \\ I_2 & -TI_2 \end{bmatrix} x_1 + \begin{bmatrix} I_2 \\ O_2 \end{bmatrix} w_1 + \begin{bmatrix} O_2 \\ I_2 \end{bmatrix} w_0 - \begin{bmatrix} I_2 & O_2 \\ I_2 & -TI_2 \end{bmatrix} v_0 \\
&\begin{bmatrix} I_2 & O_2 \\ I_2 & -TI_2 \end{bmatrix}^{-1} = \begin{bmatrix} I_2 & O_2 \\ \frac{1}{T}I_2 & -\frac{1}{T}I_2 \end{bmatrix} \\
x_1 &= \begin{bmatrix} I_2 & O_2 \\ \frac{1}{T}I_2 & -\frac{1}{T}I_2 \end{bmatrix} \hat{x}_1 - \begin{bmatrix} I_2 & O_2 \\ \frac{1}{T}I_2 & -\frac{1}{T}I_2 \end{bmatrix} \begin{bmatrix} I_2 \\ O_2 \end{bmatrix} w_1 - \begin{bmatrix} I_2 & O_2 \\ \frac{1}{T}I_2 & -\frac{1}{T}I_2 \end{bmatrix} \begin{bmatrix} O_2 \\ I_2 \end{bmatrix} w_0 \\
&+ \begin{bmatrix} I_2 & O_2 \\ \frac{1}{T}I_2 & -\frac{1}{T}I_2 \end{bmatrix} \begin{bmatrix} I_2 & O_2 \\ I_2 & -TI_2 \end{bmatrix} v_0 \\
&= \begin{bmatrix} I_2 & O_2 \\ \frac{1}{T}I_2 & -\frac{1}{T}I_2 \end{bmatrix} \hat{x}_1 - \begin{bmatrix} I_2 \\ \frac{1}{T}I_2 \end{bmatrix} w_1 + \begin{bmatrix} O_2 \\ \frac{1}{T}I_2 \end{bmatrix} w_0 + \begin{bmatrix} O_2 & O_2 \\ -\frac{1}{T}I_2 & I_2 \end{bmatrix} v_0
\end{aligned}
$$

Knowing that a sum of gaussians is also gaussian, we may conclude that x_1 also is gaussian. Then, we can find:

$$E[x_1] = \begin{bmatrix} I_2 & O_2 \\ \frac{1}{T}I_2 & -\frac{1}{T}I_2 \end{bmatrix} \hat{x}_1$$

$$Var[x_1] = \begin{bmatrix} I_2 \\ \frac{1}{T}I_2 \end{bmatrix} R \begin{bmatrix} I_2 & \frac{1}{T} \end{bmatrix} + \begin{bmatrix} O_2 \\ \frac{1}{T} \end{bmatrix} R \begin{bmatrix} O_2 & \frac{1}{T} \end{bmatrix} + \begin{bmatrix} O_2 & O_2 \\ -\frac{1}{T}I_2 & I_2 \end{bmatrix} Q \begin{bmatrix} O_2 & -\frac{1}{T}I_2 \\ O_2 & I_2 \end{bmatrix}$$

Then, calculating the products:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ \frac{1}{T} & 0 \\ 0 & \frac{1}{T} \end{bmatrix} \begin{bmatrix} r_1 & r_2 \\ r_3 & r_4 \end{bmatrix} \begin{bmatrix} 1 & 0 & \frac{1}{T} & 0 \\ 0 & 1 & 0 & \frac{1}{T} \end{bmatrix} = \begin{bmatrix} r_1 & r_2 \\ r_3 & r_4 \\ \frac{1}{T}r_1 & \frac{1}{T}r_2 \\ \frac{1}{T}r_3 & \frac{1}{T}r_4 \end{bmatrix} \begin{bmatrix} 1 & 0 & \frac{1}{T} & 0 \\ 0 & 1 & 0 & \frac{1}{T} \end{bmatrix}$$

$$= \begin{bmatrix} r_1 & r_2 & \frac{1}{T}r_1 & \frac{1}{T}r_2 \\ r_3 & r_4 & \frac{1}{T}r_3 & \frac{1}{T}r_4 \\ \frac{1}{T}r_1 & \frac{1}{T}r_2 & \frac{1}{T^2}r_1 & \frac{1}{T^2}r_2 \\ \frac{1}{T}r_3 & \frac{1}{T}r_4 & \frac{1}{T^2}r_3 & \frac{1}{T^2}r_4 \end{bmatrix} = \begin{bmatrix} R & \frac{1}{T}R \\ \frac{1}{T}R & \frac{1}{T^2}R \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \frac{1}{T} & 0 \\ 0 & \frac{1}{T} \end{bmatrix} \begin{bmatrix} r_1 & r_2 \\ r_3 & r_4 \end{bmatrix} \begin{bmatrix} 0 & 0 & \frac{1}{T} & 0 \\ 0 & 0 & 0 & \frac{1}{T} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \frac{1}{T}r_1 & \frac{1}{T}r_2 \\ \frac{1}{T}r_3 & \frac{1}{T}r_4 \end{bmatrix} \begin{bmatrix} 0 & 0 & \frac{1}{T} & 0 \\ 0 & 0 & 0 & \frac{1}{T} \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{T^2}r_1 & \frac{1}{T^2}r_2 \\ 0 & 0 & \frac{1}{T^2}r_3 & \frac{1}{T^2}r_4 \end{bmatrix} = \begin{bmatrix} O_2 & O_2 \\ O_2 & \frac{1}{T^2}R \end{bmatrix}$$

$$\sigma_a^2 \begin{bmatrix} O_2 & O_2 \\ -\frac{1}{T}I_2 & I_2 \end{bmatrix} \begin{bmatrix} \frac{T^3}{3}I_2 & \frac{T^2}{2}I_2 \\ \frac{T^2}{2}I_2 & TI_2 \end{bmatrix} \begin{bmatrix} O_2 & -\frac{1}{T}I_2 \\ O_2 & I_2 \end{bmatrix} = \sigma_a^2 \begin{bmatrix} O_2 & O_2 \\ -\frac{1}{T}I_2 & I_2 \end{bmatrix} \begin{bmatrix} O_2 & T^2(\frac{1}{2} - \frac{1}{3})I_2 \\ O_2 & T(1 - \frac{1}{2})I_2 \end{bmatrix}$$

$$= \begin{bmatrix} O_2 & O_2 \\ O_2 & T(\frac{1}{2} - \frac{1}{6})I_2 \end{bmatrix} = \begin{bmatrix} O_2 & O_2 \\ O_2 & \frac{T}{3}I_2 \end{bmatrix}$$

Such that:

$$\begin{aligned} Var[x_1] &= \begin{bmatrix} R & \frac{1}{T}R \\ \frac{1}{T}R & \frac{1}{T^2}R \end{bmatrix} + \begin{bmatrix} O_2 & O_2 \\ O_2 & \frac{1}{T^2}R \end{bmatrix} + \begin{bmatrix} O_2 & O_2 \\ O_2 & \frac{T}{3}I_2 \end{bmatrix} \\ &= \begin{bmatrix} R & \frac{1}{T}R \\ \frac{1}{T}R & \frac{2}{T^2}R + \frac{T}{3}I_2 \end{bmatrix} \end{aligned}$$

And to sum up, the true state is distributed as a gaussian, with expected value $E[x_1]$ and covariance matrix $Var[x_1]$ as stated above.

2.5 e)

In theory, this initialization scheme is optimal, as we know the exact system model. Then, as we would generally not know the exact model in practice, it may no longer be optimal. Still, as long as our model is reasonable, this would be a fairly decent starting point, but other models and methods would then be the optimal choice.

3 Task 3: Implement EKF in MATLAB

This task was implemented according to **Algorithm 2** The extended Kalman filter, as stated in the textbook, [1, page 73]. The code is added with the report, see appendix A.1.

4 Task 4: Make CV model to use with the EKF class

The model was implemented in MATLAB, see appendix A.2. Note that the model implemented was already linear, so in theory there was no need for an extended Kalman filter, though as we can see in the next task, the EKF does still work.

5 Task 5: Tuning of KF

See appendix A.3 for the code implemented. Note that I was unable to complete the subtasks b, c and d, as I didn't really understand how to do some of the plotting and calculations.

6 Task 6: Implement a SIR particle filter for a pendulum

6.1 a)

The filter is not really preforming too well, but the primary reason here is that the filter is missing which side of $\theta = 0$ the pendulum is at. This is obviously due to the fact that the measurement device is placed directly bellow this point, meaninig that there is no difference in the measurement wether the pendulum is on one side or the other.

The code is added to the report, see appendix A.4.

6.2 b)

Placing the measurement device further to the left meant that the filter predicted the values much better. This makes sense, as there is now a measurable difference between $\theta < 0$ and $\theta > 0$.

For fun I also calculated and recorded the degeneracy of the filter over time, and it can be noted that degeneracy quickly became a problem several times.

6.3 c)

One problem is that nonlinear transformations often lead to pdfs that no longer are gaussian. This means that we no longer can use an EKF. Therefore, in certain cases where the posterior can't be expressed as a gaussian (or decently approximated as one), it would be better to use a particle filter, as we could implement it for more arbitrary pdfs.

A MATLAB Code

The MATLAB code generated for this assignment. Note that most of this is based on a skeleton handed out with the assignment.

A.1 Task 3

```
classdef EKF
    % FILL IN THE DOTS
    properties
        model

        f % discrete prediction function
        F % jacobian of prediction function
        Q % additive discrete noise covariance

        h % measurement function
        H % measurement function jacobian
        R % additive measurement noise covariance
    end
    methods
        function obj = EKF(model)
            obj = obj.setModel(model);
        end

        function obj = setModel(obj, model)
           % sets the internal functions from model
           obj.model = model;

           obj.f = model.f;
           obj.F = model.F;
           obj.Q = model.Q;

           obj.h = model.h;
           obj.H = model.H;
           obj.R = model.R;
        end

        function [xp, Pp] = predict(obj, x, P, Ts)
            % returns the predicted mean and covariance for a time step
            Fk = obj.F(x, Ts);

            xp = obj.f(x, Ts);
            Pp = Fk * P * (Fk') + obj.Q(x, Ts);
```

```
        end

        function [vk, Sk] = innovation(obj, z, x, P)
            % returns the innovation and innovation covariance
            Hk = obj.H(x);

            % Assuming z = z_{k}
            % Assuming x = x_{k|k-1}
            vk = z - obj.h(x);
            % Assuming P = P_{k|k-1}
            % Assuming R is implemented as such
            Sk = Hk * P * (Hk') + obj.R;
        end

        function [xupd, Pupd] = update(obj, z, x, P)
            % returns the mean and covariance after conditioning on the
            % measurement

            % Same assumptions as above
            [vk, Sk] = obj.innovation(z, x, P);
            Hk = obj.H(x);
            I = eye(size(P));

            Wk = P * (Hk') / Sk;

            xupd = x + Wk * vk;
            Pupd = (I - Wk * Hk) * P;
        end

        function NIS = NIS(obj, z, x, P)
            % returns the normalized innovation squared
            [vk, Sk] = obj.innovation(z, x, P);

            NIS = vk / Sk * vk;
        end

        function ll = loglikelihood(obj, z, x, P)
            % returns the logarithm of the marginal mesurement distribu
            [vk, Sk] = obj.innovation(z, x, P);
            NIS = obj.NIS(z, x, P);

            ll = -0.5 * (NIS + log(det(2 * pi * Sk)));
        end

```

```
        end
end
```

A.2 Task 4

Note that I assume that q is a $4x4$ matrix, and r is a $2x2$ matrix. This is handled in the code of the next task.

```
function model = discreteCVmodel(q, r)
    % returns a structure that implements a discrete time CV model with
    % continuous time accelleration covariance q and positional
    % measurement with noise with covariance r, both in two dimensions.

    model.f = @(x, Ts) [1, 0, Ts, 0;
                         0, 1, 0, Ts;
                         0, 0, 1, 0;
                         0, 0, 0, 1] * x;
    model.F = @(x, Ts) [1, 0, Ts, 0;
                         0, 1, 0, Ts;
                         0, 0, 1, 0;
                         0, 0, 0, 1];
    % in the CV model, assuming q = sigma^2 eye(2)
    model.Q = @(x, Ts) q * [Ts^3 / 3,   0,          Ts^2 / 2,   0;
                             0,          Ts^3 / 3,   0,          Ts^2 / 2;
                             Ts^2 / 2,   0,          Ts,         0;
                             0,          Ts^2 / 2,   0,          Ts];

    model.h = @(x) [1, 0, 0, 0;
                    0, 1, 0, 0] * x;
    model.H = @(x) [1, 0, 0, 0;
                    0, 1, 0, 0];
    model.R = r;
end
```

A.3 Task 5

As I used $z_{0,1}$ to calculate $\hat{x}_1$, then these measurements are no longer useful for estimation. Therefore these values are omitted from all the executions of the EKF.

```
% get and plot the data
usePregen = true % choose between own generated data and pregenerated
if usePregen
    load task5data.mat
    fprintf('K = %i time steps with sampling intervall Ts = %f sec', K,
    figure(1); clf; grid on; hold on;
    % show ground truth and measurements
    plot(Xgt(1,:), Xgt(2,:));
    scatter(Z(1, :), Z(2, :));
    title('Data')
    % show turnrate
    figure(2); clf; grid on;
    plot(Xgt(5, :));
    xlabel('time step')
    ylabel('turn rate')
else
    % rng(...) % random seed can be set for repeatability
    % inital state distribution
    x0 = [0, 0, 1, 1, 0]';
    P0 = diag([50, 50, 10, 10, pi/4].^2);
    % model parameters
%   % commented out to be able to run pregen without filling this in
%     qtrue = [...; ...];
%     rtrue = ...;
%     % sampling interval a lenght
%     K = ...;
%     Ts = ...;
    % get data
    [Xgt, Z] = sampleCTtraj(K, Ts, x0, P0, qtrue, rtrue);
    % show ground truth and measurements
    figure(1); clf; grid on; hold on;
    plot(Xgt(1,:), Xgt(2,:));
    scatter(Z(1, :), Z(2, :));
    title('Data')
    % show turnrate
    figure(2); clf; grid on;
    plot(Xgt(5, :));
    xlabel('time step')
    ylabel('turn rate')
```

```
end
%%
% 5 a: tune by hand and comment -- FILL IN THE DOTS

% allocate
xbar = zeros(4, K);
xhat = zeros(4, K);
Pbar = zeros(4, 4, K);
Phat = zeros(4, 4, K);

% set parameters
q = 5 * eye(4);
r = 2 * eye(2);

% create the model and estimator object
model = discreteCVmodel(q, r);
ekf = EKF(model);

% initialize
K_gain = [eye(2),           zeros(2, 2);
     (1/Ts) * eye(2), - (1/Ts) * eye(2)];
xhat(:, 1) = K_gain * [Z(:, 1); Z(:, 2)];
Phat(:, :, 1) = [r,          1/Ts * r;
                 1/Ts * r,  2/(Ts^2) * r + Ts/3 * eye(2)];

for k = 3:(K-1)
    % estimate
    [xp, Pp] = ekf.predict(xhat(:, k), Phat(:, :, k), Ts);
    xbar(:, k) = xp;
    Pbar(:, :, k) = Pp;
    % innovate
    [vk, Sk] = ekf.innovation(Z(:, k + 1), xp, Pp);
    % update
    [xupd, Pupd] = ekf.update(Z(:, k + 1), xp, Pp);
    xhat(:, k + 1) = xupd;
    Phat(:, :, k + 1) = Pupd;
end

% calculate a performance metric
RMSE = @(x, x_hat) (sqrt(mean((x' - x_hat').^2)));
posRMSE = RMSE(Xgt(1:2, :), xhat(1:2, :)); % position RMSE
velRMSE = RMSE(Xgt(3:4, :), xhat(3:4, :)); % velocity RMSE

% show results
```

```
figure(3); clf; grid on; hold on;
plot(Xgt(1,:), Xgt(2,:));
plot(xhat(1,:), xhat(2, :));
title(sprintf('q = %f, r = %f, posRMSE = %f, velRMSE= %f',q, r, posRMSE
%%
% Task 5 b and c -- FILL IN THE DOTS

% parameters for the parameter grid
Nvals = 100;
qlow = 0.1;
qhigh = 100;
rlow = 0.1;
rhigh = 100;

% set the grid on logscale (not mandatory)
qs = logspace(log10(qlow), log10(qhigh), Nvals);
rs = logspace(log10(rlow), log10(rhigh), Nvals);

% allocate estimates
xbar = zeros(4, K);
Pbar = zeros(4, 4, K);
xhat = zeros(4, K);
Phat = zeros(4, 4, K);

% allocate for metrics over the grid
NIS = zeros(Nvals, Nvals, K);
NEES = zeros(Nvals, Nvals, K);

% other values of interest that can be stored
% only if you want to investigate something, like bias etc.

% initialize (the same for all parameters can be used)
xbar(:, 1) = K_gain * [Z(:, 1); Z(:, 2)];
Pbar(:, : , 1) = [r,          1/Ts * r;
                  1/Ts * r,  2/(Ts^2) * r + Ts/3 * eye(2)];


% loop through the grid and estimate for each pair
for i = 1:Nvals % q = qs
    for j = 1:Nvals % r = rs
        % create the model and estimator object
        model = discreteCVmodel(qs(i) * eye(4), rs(j) * eye(2));
        ekf = EKF(model);
        for k = 3:(K-1)
```

```
            % estimate
            [xp, Pp] = ekf.predict(xhat(:, k), Phat(:, :, k), Ts);
            xbar(:, k) = xp;
            Pbar(:, :, k) = Pp;
            % innovate
            % [vk, Sk] = ekf.innovation(Z(:, k + 1), xp, Pp);
            % update
            [xupd, Pupd] = ekf.update(Z(:, k + 1), xp, Pp);
            xhat(:, k + 1) = xupd;
            Phat(:, :, k + 1) = Pupd;
            NIS(i, j, k) = ekf.NIS(Z(:, k + 1), xhat(:, k + 1), Phat(:,
        end

    end
end

% calculate averages
ANEES = ...
ANIS = sum(NIS);
%%
% Task 5 b: ANIS plot -- FILL IN THE DOTS

% specify the  probabilities for confidence regions and calculate
alphas = ...
CINIS = ...; % the confidence bounds, Hint: inverse CDF.
disp(CINIS);

% plot
[qq ,rr] = meshgrid(qs, rs); % creates the needed grid for plotting
figure(4); clf; grid on;
surf(...); hold on;
caxis([0, 10])
[C, H] = contour(...);
i = 1;
while i <= size(C, 2)
    istart = i + 1;
    iend = i + C(2, i);
    % plots the countours on the surface
    plot3(C(1,istart:iend), C(2,istart:iend),ones(1,C(2, i))*C(1,i), 'r
    i = i + C(2, i) + 1;
end
xlabel('q')
ylabel('r')
zlabel('ANIS')
```

```
zlim([0, 10])
%%
% Task 5 c: ANEES plot

% specify the  probabilities for confidence regions and calculate
alphas = ...
CINEES = ...; % the confidence bounds, Hint inverse CDF
disp(CINEES);

% plot
[qq ,rr] = meshgrid(qs, rs); % creates the needed grid for plotting
figure(8); clf; grid on;
surf(...); hold on;
caxis([0, 50])
[C, H] = contour(...);
i = 1;
while i <= size(C, 2)
    istart = i + 1;
    iend = i + C(2, i);
    % plots the countours on the surface
    plot3(C(1,istart:iend), C(2,istart:iend),ones(1,C(2, i))*C(1,i), 'r
    i = i + C(2, i) + 1;
end
xlabel('q')
ylabel('r')
zlabel('ANEES')
zlim([0, 50])
%%
% anything extra:
```

A.4 Task 6

```
% trajectory generation

% scenario parameters
x0 = [pi/2, -pi/100];
Ts = 0.05;
K = round(40/Ts);

% constants
g = 9.81;
l = 1;
a = g/l;
d = 0.1;
S = 5;

% disturbance PDF
fpdf = makedist('uniform', 'lower', -S, 'upper', S); % disturbance PDF

% dynamic function
modulo2pi = @(x) [mod(x(1) + pi, 2 * pi) - pi; x(2)]; % loop theta to [
contPendulum = @(x) [x(2); -d*x(2) - a*sin(x(1))]; % continuous dynamic
discPendulum = @(x, v, Ts) modulo2pi(x + Ts * contPendulum(x) + Ts*[0;

% sample a trajectory
x = zeros(2, K);
x(:, 1) = x0;
for k = 1:(K-1)
    v = random(fpdf);
    x(:, k + 1) = discPendulum(x(:,k), v, Ts);
end

% vizualize
figure(1);clf;
subplot(2,1,1);
plot(x(1,:))
xlabel('Time step')
ylabel('\theta')
subplot(2,1,2)
plot(x(2,:))
xlabel('Time step')
ylabel('d/dt \theta')
%%
% measurement generation
```

```

% constants
Ld = 4;
Ll = 3;
r = 0.25;

% noise pdf
hpdf = makedist('Triangular','a',-r,'b',0,'c',r); % measurement PDF

% measurement function
h = @(x) sqrt((Ld - l * cos(x(1)))^2 + (l * sin(x(1)) - Ll)^2 ); % meas

Z = zeros(1, K);
for k  = 1:K
    w = random(hpdf);
    Z(k) = h(x(:,k)) + w;
end

% vizualize
figure(2); clf;
plot(Z)
xlabel('Time step')
ylabel('z')
%%
% Task: Estimate the pendulum state using a particle filter
% -- FILL IN THE DOTS

% number of particles to use (tuning)
% dots
N = 1000;

% initialize particles, pretend you do
% not know where the pendulum starts
px = [2 * pi * (rand(1, N) - 1/2); randn(1, N) * pi/4];

% initial weights
w = ones(1, N) / N;

% allocate a variable for resampling particles
pxn = zeros(size(px));

% PF transition PDF: SIR proposal, or something you would like to test
% (tuning)
PFfpdf = makedist('uniform', 'lower', -S, 'upper', S);
```

```
% initialize a figure for particle animation.
figure(4); clf; grid on; hold on;
set(gcf,'Visible','on')
plotpause = 0;

N_eff = zeros(1, K);

% estimate
for k = 1:K
    % weight update
    for n = 1:N
        w(n) = pdf(hpdf, Z(k) - h(px(:, n))); % write help pdf
    end
    w = w / sum(w); % normalize

    % resample
    cumweigths = cumsum(w);
    noise = rand(1, 1) / N;
    i = 1;
    for n = 1:N
        u_n = (n - i) / N;
        while (u_n > cumweigths(i))
            i = i + 1;
        end
        % find a particle i to pick
        % algorithm in the book, but there are other options as well
        pxn(:, n) = px(:, i);
    end

    % trajecory sample prediction
    for n = 1:N
        px(:, n) = discPendulum(pxn(:, n), random(PFfpdf), Ts);
    end

    % degeneracy
    N_eff(1, k) = 1 / (sum(w.^2));

    %plot
    clf; grid on; hold on;
    scatter(l * sin(pxn(1,:)), -l * cos(pxn(1,:)),'b.');
    sh = scatter(l * sin(x(1, k)), -l * cos(x(1,k)), 'rx');
    axis([-l,l,-l,l]*1.5)
    xlabel('x')
```

```
    ylabel('y')
    title('theta mapped to x-y')
    legend('particl','\theta true')
    drawnow;
    pause(plotpause);
end
plot(1:K, N_eff);
```

References

[1] Edmund Brekke. *Fundamentals of Sensor Fusion*. 2019.