Target Tracking: exercise 1

Gustaf Hendeby
gustaf.hendeby@liu.se

Rickard Karlsson
rickard.g.karlsson@liu.se

Version: 2021-09-26

The purpose of this exercise is to get acquainted with the basic building blocks used in target tracking, as well as build basic infrastructure to facilitate the subsequent exercises. At the end of the exercise, you will have the infrastructure to simulate measurements from simple tracking scenarios, visualize and benchmark the result. Furthermore, you have gained experience of the most basic single target methods, as well as, implemented an *interacting multiple model* (IMM) filter to deal with maneuvering targets. All in presence of missed detections and clutter. Simulated data is used to gain experience, before dealing with a provided realistic dataset. In the last part of the exercise, different track logics are evaluated.

As is often the case in practice, you are allowed to use the tools that you are used to. The MATLAB Sensor Fusion and Tracking Toolbox provides a very good starting point, and you are encouraged to make use of such tools, except when state differently. However, you are free to use Python or similar if you so wish.

1 Simulation Infrastructure

Task: Implement a method to simulate measurements from a target, given a state trajectory, a measurement function, and noise properties. The function should handle, measurement noise, missed detections, and clutter.

In MATLAB, we suggest the following declaration:

```
function Y = generatedata(X, sensor, clutter)
```

where `X` is the state trajectory one time per column, `sensor` is a struct comprising `h` the measurement function, `R` the measurement covariance (assume white Gaussian noise), `PD` the probability of detection; and `clutter` a struct comprising `volume` that span the tracking volume, and `beta` clutter rate (assume the number of clutter to be Poisson distributed, and the clutter uniformly distributed in the volume). The output is a cell array for each time instance, with zero of more measurements.

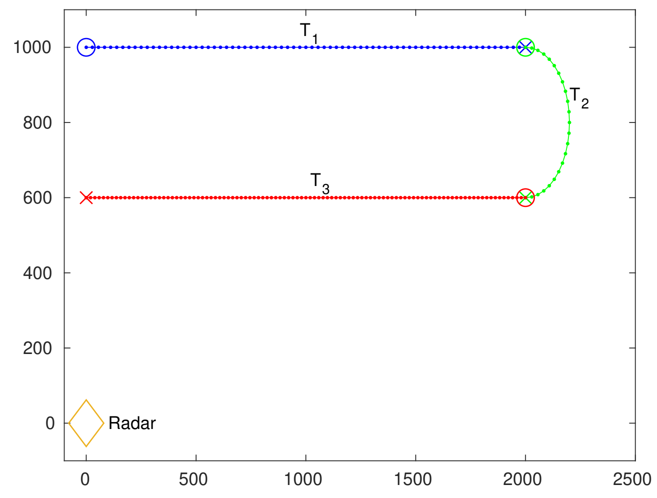

Figure 1: Illustration of the specified trajectories $\mathcal{T}_1$–$\mathcal{T}_3$. $\mathcal{T}_4$ is obtained by connecting the segments in order.

Hint: In MATLAB it is possible to use an anonomus function for the sensor model. The example below show how this can be used to construct a range measurement given position coordinates in the first and second column of X:

```
>> sensor = struct(...
     'h', @(X) sqrt(X(1,:).^2 + X(2,:).^2),...
     'R', 100^2, 'pD', 1);
```

Task: Create the following trajectories to be used in this exercise:

- $\mathcal{T}_1$ Target travels 2000 m due east, at 100 km/h.
- $\mathcal{T}_2$ Target traveling due east makes a $180°$ clock-wise turn following a circular path with radius 200 m, at 50 km/h, ending up heading west.
- $\mathcal{T}_3$ Target travels 200 m due west, at 70 km/h.
- $\mathcal{T}_4$ Connect the three segments ($\mathcal{T}_1$–$\mathcal{T}_2$–$\mathcal{T}_3$) above to get a trajectory of a maneuvering target.

See Figure 1 for an illustration.

These trajectories will be used in the rest of this exercise. Unless otherwise stated, assume a range-bearing radar ($\sigma_r = 10\,\text{m}$ and $\sigma_\phi = 0.001$) with probability of detection $P_D = 0.9$ to generate the measurements. Furthermore, assume Poisson clutter with intensity $\beta_{FA}V = 2$, uniformly distributed in the tracking volume. The sample time can be set to $T = 1\,\text{s}$.

2 Single Target Tracker

In this part of the exercise you will design three single target trackers, for different types of targets, and add functionality to handle both missing data and clutter.

2.1 EKF for a Low-Maneuvering Target

Task: Design and implement a *constant velocity* (CV) based *extended Kalman filter* (EKF) that provides good tracking when applied to a target traveling along trajectory segments $\mathcal{T}_1$ and $\mathcal{T}_3$.

To simplify, and not to focus on the track initialization, it is okay to initialize the filters in roughly the right place.

Develop the tracker in steps. Begin with the simple case with perfect detections P_D and no clutter. A good way to verify your implementation is to start out without measurement noise to see the filter behaves as expected. Then introduce missed detections $P_\text{D} < 1$, possibly in steps. Next, set $P_\text{D} = 1$ again, and add clutter. Use gating and *nearest neighbor* (NN) association. Finally, try out all at the same time.

Task: Evaluate the performance (*root mean square error*, RMSE) using Monte Carlo simulations on $\mathcal{T}_4$,

Task: For $\mathcal{T}_4$, plot the true trajectory, the measurements, and the estimated trajectory.

Note: In this part of the exercise, feel free to use available EKF implementations. If you end up implementing your own filter, it is advised to implement the filters in a class similar to the one available in the Sensor Fusion and Tracking toolbox. Using that design will simplify life further on.

2.2 EKF for a Maneuvering Target

Task: Repeat the tasks in Sec. 2.1, this time tune the filter to behave well on $\mathcal{T}_2$. Perform the RMSE evaluation on $\mathcal{T}_4$. Make the same plots as above.

2.3 Filter Bank for Maneuvering Targets

When doing the tasks above, you should have noticed that the best filter you can accomplish for the straight paths in $\mathcal{T}_1$ and $\mathcal{T}_3$ cannot follow the target in the turn in $\mathcal{T}_2$. At the same time, the filter that follows the target thought the turn gives a much noisier estimate on the straight paths. You will now implement an IMM comprising these two filters, which should give you the best of both worlds.

Task: Implement an IMM, based on the two filters above. Evaluate and illustrate the result the same way as above, this time performing the RMSE on $\mathcal{T}_4$. Furthermore, visualize the filter probabilities as a function of time.

You might have to experiment a bit with different values for the probabilities of jumping between the different modes before you get it right.

Note: You are expected to make your own IMM implementation, based on the above filters. It is strongly advised to try to mimic the interfaces used in the EKF as much as possible. This way it is easy to use the different filter interchangeably as soon as they are properly set up.

2.4 Mysterious Data

In the last part of the single target tracking exercise, you should estimate the trajectory of the target given by the measurements from a range-bearing radar in the file `ex1data.mat` available as `Y1`. The data is formatted the same way as the output of the simulator described above. The data is realistic, but is constrained to 2D, containing clutter and lost detections.

Task: Design a tracker that tracks the target from the measurements in `Y1` in `ex1data.mat`. The radar sensor has $\sigma_r = 135\,\mathrm{m}$ and $\sigma_\phi = 1.5°$. Visualize the measurements, and the estimated trajectory in the same plot.

3 Track Logic

In the next part of the exercise, you will implement two different types of track logic, N/M and track score based logic to help decide if target it present or not. The function to do this should have the same interface in both cases, *e.g.*, in the generic case:

```
function state = logic(y, filter, state, opts)
```

where `state` is a method dependent struct, containing at least one field `stage` indicating the status of the track (tentative, confirmed, or deleted). `y` is the current measurement (or empty if none is available), `filter` the filter estimating the target, and `opts` various options needed by the track logic.

3.1 N/M logic

Task: Implement an N/M logic, where N_1/M_1 and then N_2/M_2 is needed to confirm a track, and track is killed after N_3 consecutive missed observations.

The file `ex1data.mat` contains measurements in `Y2` from the following system

$$\begin{aligned} x_t &= \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} x_{t-1} + w_t \\ y_t &= \begin{pmatrix} 1 & 0 \end{pmatrix} x_t + e_t \end{aligned}$$

with $Q = 0.001\left(\begin{smallmatrix} \frac{1}{4} & \frac{1}{2} \\ \frac{1}{2} & 1 \end{smallmatrix}\right)$ and $R = 0.01$. The probability of detection is $P_{\mathrm{D}} = 0.9$ and the clutter rate is $\beta_{\mathrm{FA}} = 0.05$ in the tracking region $[-10, 10]$. Initialize the filters with

$$x_0 = \begin{pmatrix} y \\ 0 \end{pmatrix} \qquad P_0 = \begin{pmatrix} R & 0 \\ 0 & 0.1 \end{pmatrix}.$$

Task: Apply the track logic (2/2&2/3 for confirmation and 3/3 for termination) to the measurements `Y2` in `ex1data.mat`. Start a new tentative track for each observation not gated with any trajectory. Visualize the result as a horizontal line per (tentative) target, where the x-axis indicates time, and the y-axis the track identity. Indicate tentative tracks with a dashed line, and confirmed tracks with a solid line. Furthermore, indicate an associated measurement with a cross.

3.2 Track Score Logic

Task: Implement track score based logic, use a suitable solution to avoid “integrator windup”.

Task: Apply the score based logic to the measurements `Y2` in `ex1data.mat`, and visualize the result in the same way as above. Assume $P_{\mathrm{FC}} = 0.1\,\%$ and $P_{\mathrm{TM}} = 1\,\%$.