Albert-Ludwigs-Universität Freiburg
Lecture: Mobile robotics II
Winter term 2011/2012

Institut für Informatik
Exercise: Prof. Dr. Maren Bennewitz
Lab instructor: Dipl.-Phys. Jonas Rist

Exercise: Gps-based location tracking and prediction

Introduction

In this exercise you will implement a simplified version of the hierarchical dynamic bayes net for activity recognition, which was presented in the lecture. We will leave out the higher levels of the model and concentrate on the location tracking based on the GPS input data stream and a street map.

Data import and visualization

1. The **GPS data** is stored in `*.mat` files in the directory `data/gps_log`. Each `*.mat` file contains a daily GPS log in a struct named `track_*`. The fields `'x'` and `'y'` contain sequences of GPS coordinates, and `'date'` holds the date. The units are meters and seconds. The waypoints in a track are always in chronological order. You can plot different temporal sections of the track by calling `plot(track_0.x(i:j),track_0.y(i:j),'+')` with some indices `i,j`.

2. The **street map data** is stored in the `complete_map.mat` file in the directory `data/`. It is a struct containg information about nodes and edges. `'x'` and `'y'` are node coordinates, `'id'` is a node ID and `'n'` contains a cell array with vectors of neighbouring node IDs. Edges mean streets, and nodes are usually points where more than two edges meet. Call `plot_street_map(map)` to get a plot of the nodes. The edges can also be plotted, but this should be used only for small subsets of the map (it takes too long otherwise). The function `map = filter_map(map,R1,R2)` keeps all nodes within the rectangle defined by the 2D vectors R_1 and R_2. Call

   ```
   map_f = filter_map(map,[388e3 742e2],[3895e2 75e3]);
   ```

   and plot the filtered map including its edges: `plot_street_map(map_f,1)`. Also check out the useful `filter_track` function (documented in the code).

3. Calling

   ```
   [t,m]=plot_gps_track(track_0, map, [388e3 742e2],[3895e2 75e3],10e3,15e3);
   ```

   plots the track from date index 10000 to 15000 and the street map in the selected region. Implement a function

   ```
   v = get_track_speed(track)
   ```

   that returns a vector of speed values along the track and plots it. You should use this later to choose meaningful track data for the location tracking.

Hidden Markov Model (HMM) for location estimation
Implement the Hidden Markov Model from slide 11 of the lecture (a HMM is a special case of a Dynamic Baysian Network). The hidden variable is the state vector

$$\vec{x} = (x, y, v_x, v_y, e), \tag{1}$$

where (x, y) is the position on the street map, (v_x, v_y) the velocity and e the current edge. If we leave the edge variable aside for now, this model is a pure Kalman Filter which you know from the tracking exercises. Calling

```
hmm = build_hmm(track)
```

gives you a struct array `hmm`, which represents the HMM. Each struct in the array stores the information about one slice of the HMM, including the GPS measurements from the track. The variable names for the Kalman Filter variables are the same as in the tracking exercises. Implement the Kalman filtering in a function

```
hmm_filtered = filter_hmm(hmm)
```

that returns the model with the values after filtering. Use the update and prediction functions which you have implemented for the tracking exercise. You can visualize your result using the function

```
plot_hmm(hmm_filtered, map)
```

that plots the data and the filter values. (Note: If the functions hangs, the area is probably too big and too many edges have to be plotted). Adjust the initial covariance, process and measurement noise and the gating until your filter is able to follow the GPS tracks without diverging.

Tracking on the street map
Implement a function

```
[N, ids] = get_nearest_edge(map,p)
```

that returns the coordinates of the nearest point on any edge and the node IDs of the edge. Change the Kalman filter so that it snaps its position estimate to the nearest edge (slides 14,15). Also, modify the prediction step so that it can follow edge transitions.

Optional: Particle filter
For the particle filter, we have to extend the `filter_hmm(hmm)` function to include the sampling step from slide 24. To keep things simple, leave out the velocity sampling and sample the edge transition randomly. The particles can be seen as instances of filtered HMMs, their importance weights are given by the likelihoods of the measurements, which are already used for the gating. Complete the particle filter by implementing functions for dynamic deletion and creation of such HMM instances according to their importance weights (resampling).