Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Lecture: Mobile robotics II
Exercise: Prof. Dr. Kai Arras
Winter term 2011/2012
Lab instructor: Prof. Dr. Kai Arras

Exercise: People Detection and Tracking

In this exercise we will address one of the most fundamental problems of social robots deployed in populated environments: the task of recognizing people and tracking their position and motion state. We will use 2D range data for this purpose as many robots are equipped with range finders that acquire such information on their surrounding.

In the first part of this exercise, we will segment the scans to obtain a meaningful subdivision of the data into groups of neighboring points. We will then create a people detector by classifying the segments with a simple classifier.

In the second part we will implement a Kalman filter for tracking a single target based on a constant velocity motion model.

In the third part, we will deal with multiple targets and implement a nearest neighbor strategy to perform data association. Finally, we will combine the detector with the multi-target tracker and track people in real-world data collected at the Freiburg main station.

Part I: Segmentation and Detection

Exercise 1: Getting Started, Read From Data Set

1. Get the data sets and the Octave frame. The log file with the scans is `datalog.mat`, the frame is called `PeopleTrackingFrame1.m`. Find and understand where and how the log file is imported, get familiar with the resulting variable `log` using e.g. `whos` and `size`. How many steps does the log file have?

2. Plot the first step using the Octave frame. Note that the laser points in the scans are available in polar coordinates through `phi = log.phi; rho = log.step(i).rho` for step `i`. Cartesian coordinates are obtained by `[x y] = pol2cart(phi,rho);`. **Hint** for Octave users: for a good visualization use Marker 'd' and set the 'MarkerSize' value to 8.

3. Animate the log file. Iterate over the steps, plot the scan of each step and store the graphical handle returned by the plot command `h = plot(...)`. Call `drawnow` to update the current graphics and then delete the handle calling `delete(h);`. This is a simple and convenient way to animate data in

Octave/Matlab. Use the figure title to display the step numbers by concatenating the title text with the string `int2str(i)`.

4. To skip the animation for the remainder of the exercise, put an if-then-else statement around the previous code.

Exercise 2: Segmentation

We assume that people are always well separated from the background and each other. This enables us to implement a simple but effective segmentation strategy based on the detection of jump distances. The method initializes a new segment each time a range discontinuity is larger than a threshold.

1. Segment the scan into groups of neighboring points using jump distance clustering and an appropriate threshold. Do not use for-loops over the scan for this exercise, elegant solutions using `diff` or `find` exist. Interpret the first and last beams in the scan as pseudo-break points. **Hint:** you might want to first compute a vector of all begin and end indices of all segments based on the result of `find`.

2. Store the segments in an array of struct. For each entry, store relevant information such as a segment identifier, the begin and end indices in the scan, the actual points in x,y-coordinates, or the center of mass.

3. Visualize your segmentation result in a new figure by looping over all segments, picking a different color for each segment (e.g. randomly), and plotting its points. You can also plot the center of mass and, using the `text` command, the segment identifier at the latter position.

Exercise 3: People Detection by Classification of Segments

Once the scan is subdivided into meaningful clusters, we can now characterize the segments using sets of geometric or statistical features. Based on such features we will create a simple decision tree-like classifier with hand-tuned thresholds.

1. Compute a couple of features for each segment that appear informative to you given our classification task. Examples include segment width, number of points, compactness, convexity, etc.

2. Create a decision tree-like classifier from these features and find appropriate thresholds. Classify each segment to be people or non-people.

3. Visualize the segments again in a new figure, now by plotting the negative samples using a background color (e.g. black).

4. Animate the detection results by iterating over the log file and visualizing the detection result of each step. To delete a vector of graphical handles (possible in Matlab, not implemented in Octave), store the handle for each segment in the struct and iterate over all struct entries for deletion.

Exercise 4 (Optional): Feature Analysis

1. Load the ground truth annotations from the file `datalabels.mat` as demonstrated in the Octave frame. The labels denote the ground truth segmentation, detection and data association. Get the annotations for each frame by `labels(i,:)` and segment each scan using this information into foreground segments (corresponding to people) and background segments (corresponding to other objects). For each segment compute a couple of features such as width, standard deviation, convexity, curvature, etc. Then, for each feature, plot all values along the x-axis but use different colors for the two classes. A good separation between the fore- and background class is indication that the feature is well suited for this classification task. Improve your people detector accordingly.

Part II: Tracking and Data Association

Exercise 5: Getting Started, Read From Data Set

We will first consider a single track in isolation and come back to our original problem towards the end of the exercise.

1. Start by loading the file `datatracks.mat` as demonstrated in the Octave frame `PeopleTrackingFrame2.m` and check the variables that you have read in. The file contains three $2 \times n$-matrices `z1`, `z2`, `z3` with a sequence of (x, y)-observations of the position of a moving person through a simulated sensor and three Boolean $1 \times n$-vectors `zvalid1`, `zvalid2`, `zvalid3` whose entries indicate if the corresponding measurement is valid. We assume the target has been observed at a constant frequency f and let $\Delta t = 1/f$. What is the meaning of matrices `R` ?

2. Plot the two observation sequences into a new figure. Use the `zvalid` variable to only plot the valid observations. Note that some observation sequences have gaps and outliers. Explain possible causes of such events.

3. Set up a for-loop over the observation sequence and extract the respective z, R for each step.

We will now implement a Kalman filter to track people and start with the observation sequence `z1`.

Exercise 6: State Representation and Initialization

We define the state representation of the filter to be $\mathbf{x} = (x, y, \dot{x}, \dot{y})^T$. It contains the estimated position of the person and its velocities in x and y. The associated covariance matrix is thus a 4×4-matrix whose entries describe the uncertainties of the state components and covariances between them.

1. Initialize the state at time 0 from the first measurement. As our sensor can only observe the position of the target and not its velocities, we set the velocity component to zero and define a large initial covariance C_0 that reflects our lack of knowledge with suitably large number on its diagonal (see `C0` in the frame).

Exercise 7: Motion Model

The role of the motion model is to project the state into the future within Δt. Here we will implement the constant velocity motion model that assumes constant velocities perturbed by a normally distributed zero-mean acceleration.

1. The prediction expressions for the mean $\mathbf{x}$ and the covariance C are

$$\mathbf{x}(k+1|k) = F\,\mathbf{x}(k|k) + G\,\mathbf{a} \quad (1)$$
$$C(k+1|k) = F\,C(k|k)\,F^T + Q \quad (2)$$

   where $\mathbf{a}$ is the 2×1 acceleration noise term with constant 2×2 covariance A. Determine the matrices F and G formally. **Hint:** consider it a one-dimensional problem. The matrix Q, called process noise covariance, can be derived from A and G (see course). For simplicity we will assume it to be `diag([0.001 0.001 0.1 0.1])`. What is the meaning of the matrices F, G and Q ?

2. Create a new m-file for the function `predictstate` and implement the motion model. The function takes Δt, Q, the last posterior state estimates $\mathbf{x}(k|k), C(k|k)$ as input and returns the predicted state and predicted state covariance $\mathbf{x}(k+1|k), C(k+1|k)$.

Exercise 8: Measurement Model

1. In general, observations of the state cannot be done directly but remotely over a (linear or non-linear) function of the state. This relationship is the measurement or observation model, $\mathbf{z}(k+1|k) = H\,\mathbf{x}(k+1|k)$. The model

serves to predict the next measurement based on the predicted state. The term $\mathbf{z}(k+1|k)$ is the position where we expect the next measurement to occur. Since in our case we can directly observe two state variables, H acts as a row selection matrix. Determine H and assign the predicted measurement to a variable, e.g. `zp`.

Exercise 9: Kalman Filter

Given the predicted state and the observation, we are now able to close the loop and compute the posterior state and state covariance estimates. This is done by the Kalman filter, the optimal minimum mean-square error estimator under linear Gaussian conditions. In a nutshell, the filter makes a weighted average between the uncertain state prediction and the uncertain target observation. If the motion model describes the system's dynamics very well (small Q) and the sensor is unreliable and noisy (large R), the filter will give more weight to the state prediction and vice versa.

1. The Kalman update expressions for the mean $\mathbf{x}$ and the covariance C are

$$\begin{aligned} S(k+1) &= H\, C(k+1|k)\, H^T + R \\ \nu(k+1) &= \mathbf{z}(k+1) - \mathbf{z}(k+1|k) \\ K(k+1) &= C(k+1|k)\, H^T\, S(k+1)^{-1} \\ \mathbf{x}(k+1|k+1) &= \mathbf{x}(k+1|k) + K(k+1)\, \nu(k+1) \\ C(k+1|k+1) &= C(k+1|k) - K(k+1)\, H\, C(k+1|k) \end{aligned}$$

   Implement the update equations in a new function m-file with the interface `[x C] = updatestate(xp,Cp,z,zp,R,H)` where `xp,Cp` denote the predicted state estimates $\mathbf{x}(k+1|k), C(k+1|k)$.

2. We want to store the histories of relevant variables over the tracking sequence. This is done by concatenating the estimates into a matrix at the end of the cycle using the `cat` command. Concatenate the two matrices along the third dimension.

3. Track the person by running the filter over the observation sequence. Plot the estimation histories of $\mathbf{x}(k+1|k+1), C(k+1|k+1)$ and $\mathbf{x}(k+1|k), C(k+1|k)$. For the covariance matrices, use the `librobotics`-command `drawprobellipse` (use 0.95 for α). Explain the tracking behavior of the filter in particular during maneuvers.

4. Familiarize yourself with the `subplot` command. Plot the state component x, y and $\dot{x}, \dot{y}$ in different subfigures. Plot the diagonal elements of the $C(k+1|k+1)$

into a third subfigure. **Hint:** to obtain a vector of values across a multi-dimensional matrix, use the `squeeze` command.

Exercise 10: Handling Occlusions, Misdetections and Outliers

We now turn to the second track in our log file. As you can see from the plot of the observation sequence, the sensor was not able to detect this person reliably: there are gaps that correspond to occlusions and detector failures. We now implement a mechanism to maintain the track over such gaps.

1. Make sure that the algorithm receives the proper `z`, `zvalid`, `R`.

2. Introduce a simple logic that allows the tracker to deal with general observation sequences that have gaps and do not start at the beginning of the experiment. There are two conditions to be tested: if the filter is initialized and if the observation is valid. **Hint:** Predict the state regardless the validity of the current measurement. In case z is invalid, the posterior state estimates become the predicted state estimates.

3. Track the second person by running the filter over the observation sequence. Plot the estimation histories. The estimated state $\mathbf{x}$ should nicely interpolate gaps. Explain the behavior of the associated covariance matrix.

4. Consider now the third observation sequence `z3`, `zvalid3`, `R3`. Rerun the filter and describe what happens.

5. To extend the tracker with the capability to deal with outliers we will introduce gating. Gating is a test if $\mathbf{z}(k+1)$ is statistically compatible with the prediction $\mathbf{z}(k+1|k)$. If yes, the observation is said to lie in the validation gate of the prediction. Implement the following expression using the matrix $S(k+1)$ and vector $\nu(k+1)$:

$$d^2 \;=\; \nu(k+1)^T \, S(k+1)^{-1} \, \nu(k+1)$$

   The distance d^2 is the squared Mahalanobis distance, a generalized form of the Euclidian distance. An observation is an outlier if the condition $d^2 < \theta$ is not satisfied with θ being a threshold drawn from a Chi-square distribution. To obtain θ, use `chi2invtable(0.99,2)` (more details in the course). In case of a non-match, close the loop by copying the priors into the posteriors.

6. Rerun the tracker and describe what happens. Use different values of Q and R and examine their effect.

Part III: Data Association and Multi-Target Tracking

We will now consider a multi-target tracking problem and implement the Nearest Neighbor Data association Filter (NNSF). Finally, we will come back to our initial detection problem and put all pieces together to obtain a full people detection and tracking system for 2D range data.

Exercise 11: Getting Started, Read From Data Set

1. Start by loading the file `dataobs3.mat` as demonstrated in the Octave frame `PeopleTrackingFrame3.m` and check the variables that you have read in. We again assume the target has been observed at a constant frequency f and let $\Delta t = 1/f$.

2. Plot all observation sequences into a new figure. Use the `zvalid`-field to only plot the valid observations. Explain possible causes of the gaps in some of the sequences.

Exercise 12: Data Association: Nearest Neighbor Strategy

1. Set up the main tracking loop using two structs, `tracks` and `obs` that hold the track- and observation-specific information such as the z-values, the state estimates etc. Use a status-field for both, tracks and observations, to indicate the states `'non-matched'` and `'matched'`. Fill in the missing code in the frame.

2. Implement the Nearest Neighbor Data association Filter (NNSF) using the interface `[tracks,obs] = matchnnsf(tracks,obs,ALPHA);` where `ALPHA` is 0.99. First compute the Mahalanobis distance matrix D between all observations and state predictions. Then iteratively find the minimum distance in D, test if it is smaller than the gating threshold from the Chi-square distribution, if yes, mark the pairing as matched and remove the corresponding row and column in D. Terminate otherwise. (The NNSF has a very similar implementation than a hierarchical clustering algorithm). **Hint:** Instead of deleting the rows and columns, mark them with `Inf`.

3. Plot the estimation histories.

4. Rerun the tracking with the log file `dataobs4.mat`. What happens?

Exercise 12: Full People Detection and Tracking

1. Implement the full people detection and tracking system by combining the detector from the first part of this exercise with the multi-target tracking of the third part.