Joint-acceleration-model for association of multiple coordinates

Tomasz Dubiel-Teleszynski

2014

1. To start go to file **START.m**. There are several inputs this almost fully automatic data driven algorithm needs which are described there (default values are in line with movies' specifications). Preferences which are to be set as well are amenable within prefs structure (default values are set).

2. Algorithm starts with loading and preprocessing data in **fn_proc_filen_.m**. Three tracker outputs are taken into consideration:

   - cascade classifier (**Cascade**)

   - cross correlation (**X-correl**)

   - foreground detection (**Foreground**)

   Foreground often returns zeros at the beginning, thus all coordinates are truncated up to the point this tracker output is not zero anymore. Hence, time series coordinates are shorter in some cases but for initial interpretation of how efficient the algorithm is it plays minor role.

   Both raw and truncated coordinates can be plotted and in addition plots of truncated coordinates are saved in **.pdf** files.

   Optimal X&Y are coordinates labelled **.xy** in the original data structure and these are used for benchmarking.

3. Function **fn_proc_cord_.m** is responsible for processing coordinates. X and Y coordinates are processed separately what results from the logic of the underlying motion model. It assumes state vector $(x_1, v_1, x_2, v_2, ..., x_n, v_n, a)'$, where n is number of tracking algorithms, $x_i, i = 1, ..., n$ are X coordinates, $v_i, i = 1, ..., n$ are their respective velocities and a is acceleration common for all of them (in their direction). Wiener process model is used for dynamics of acceleration. Analogous holds for Y coordinates which are processed separately.

   Due to such a structure covariance matrix in the motion model (state transition equation) can not be inverted whence density does not exist. This is the main difficulty and it makes construction of particle filter nontrivial but the latter can be achieved using Kalman filter in the proposal distribution (only then particle weights can be updated).

4. Function **fn_sm_.m** is responsible for automatic construction of system matrices irrespective of how many coordinates we have. It makes use of MATLAB's symbolic expression toolbox.

5. Function **fn_sv_.m** is used to initialize and reinitialize state for different purposes.

6. Classical maximum likelihood estimation and reestimation (same apart from message shown) are done in **fn_mle_.m** and **fn_mle_re_.m**.

7. Outlier detection is based on Kalman disturbance smoother and resultant t-test and very low 0.001 significance level equivalent to 99.9% confidence is chosen to detect outliers. It all happens within **fn_outl_det_.m**.

8. Outlier adjustment is done either model based, that is using Kalman filter with automatically created dummies (**fn_dm_.m**) and maximum likelihood estimation (**fn_outl_adj_dums_.m**) or means based that is outliers are substituted with simple averages calculated from neighbouring points (**fn_outl_adj_mean_.m**). Although simple and rule of thumb the second method gives comparable results and is way faster!

9. After state reinitialization and reestimation classical estimates are then corrected using Bayesian adaptive Metropolis-Hastings algorithm (**fn_mcmce_.m**) and these are final.

10. Using these final estimates of observation noise variances for each coordinate and variance in the Wiener process model for common acceleration, the following is done. Kalman filtering (KF), Kalman state smoothing (SS), particle filtering (PF, using multinomial resampling, Kalman filter proposal and Metropolis-Hastings MCMC move step), particle smoothing (PS) and maximum a posteriori (MAP) estimation are performed on outlier adjusted coordinates using reinitialized state.

11. Estimates of final position which is compared to benchmark optimal coordinate are computed using weights calculated from inverse standard deviations of observation noise (**fn_wgts_.m** within **fn_fin_cord_.m**) and different state estimates obtained in the previous point. They are then plotted against benchmark and plots are saved for each X and Y coordinate in **.pdf** files. Numerical results are also saved in **.mat** file.

12. Finally there is a function **fn_compare_.m** which is commented out in **START.m**. It can be used to display these position estimates against benchmark (OPT) and mean raw estimates (MEAN) on used frames (some modifications to path naming have to be done and original **frames_used_****.mat** are needed for this function to work properly as it is written).