README for “A distributed local Kalman consensus filter for traffic estimation” JAVA source code

Ye Sun and Daniel B. Work

May 28, 2014

Abstract

This document describes the implementation of the distributed local Kalman consensus filtering algorithm for traffic density estimation, introduced in the article “A distributed local Kalman consensus filter for traffic estimation” by Sun and Work, in preparation for submission to the IEEE Transactions on Control of Network Systems. The source code is available to be downloaded at https://github.com/yesun/DLKCF.

1 License

This software is licensed under the *University of Illinois/NCSA Open Source License*:

2 Publishing results using this software

We kindly ask any future publications using this software include a reference to the following publication:

Y. Sun and D.B. Work, "A distributed local Kalman consensus filter for traffic estimation," *submitted to the IEEE Transactions on Control of Network Systems*, 2014.

3 General Instruction

3.1 General usage notes

1. This release of software is intended to complement a paper submission to the the IEEE Transactions on Control of Network Systems;

2. This software is intended to be run on Eclipse. It was developed using Eclipse 4.3.0, and the development environment is Java SE Development Kit 7u25.

3.2 How to use the software

1. Download the folders **DLKCF_SectionV_A**, **Luenberger_SectionV_A** (for Section V-A in the paper) and **DLKCF_SectionV_B**, and import them into Eclipse. The folder *MatlabPlot* includes matlab code to generate plots for the true densities and the density estimates;

2. To generate Fig. 1b in the paper, run *Test.java* in any of the three projects and plot the generated *trueState.csv* in the folder *results* using *Main.m* in the folder *MatlabPlot*. Note that *trueState.csv* should be put in the folder *Result_Plot*, same for the rest of the plots generated using *Main.m*;

3. To generate Fig. 2a in the paper, run *Test.java* in **DLKCF_SectionV_A** and plot the generated *writerAvr.csv* using *Main.m* in the folder *MatlabPlot*;

4. To generate Fig. 2b in the paper, run *Test.java* in **Luenberger_SectionV_A** and plot the generated *writerAvr.csv* using *Main.m* in the folder *MatlabPlot*;

5. To generate Fig. 3, run *Test.java* in **DLKCF_SectionV_A** and **Luenberger_SectionV_A**, and plot the sum of error for section 0 and section 14 using the data in the last column of *D_28_0error.csv* and *D_23_14error.csv* respectively. The submode can be plotted by the data in the last two columns of *D_28_0Mode.csv* and *D_23_14Mode.csv* (column F is the submode for the estimated state, and column G is the submode for the true state);

6. To generate Fig. 4a in the paper, run *Test.java* in **DLKCF_SectionV_B** and plot the generated *writerAvr2.csv* using *Main.m* in the folder *MatlabPlot*;

7. To generate Fig. 4b in the paper, run *Test.java* in **DLKCF_SectionV_B** and plot the generated *writerAvr.csv* using *Main.m* in the folder *MatlabPlot*;

8. To generate Fig. 5a in the paper run Test.java in **DLKCF_SectionV_B** and plot the disagreement of the DLKCF, the DLKF and individual local KFs from the data in the first column of *Disagreement.csv*, *Disagreement1.csv*, and *Disagreement2.csv*, respectively.

9. To generate Fig. 5b in the paper run Test.java in **DLKCF_SectionV_B** and plot the error of the DLKCF, the DLKF and individual local KFs from the data in the first column of *Error.csv*, *Error1.csv*, and *Error2.csv*, respectively.

10. The csv files in the folders *results* are generated by the supplementary code, which can be considered the same as the figures in the paper subject to some random Gaussian noise from the initial guess noise and the sensor noise.

11. The two csv files currently in *Result_Plot* can serve as demonstrations to generate Fig. 1b (*trueState.csv*) and Fig. 4b (*writerAvr.csv*) in the paper.

4 Package List

1. *DoubleMatrix*, some operations on matrix and *GaussianGenerator.java* for generating Gaussian (and other types of) noise;

2. *trueSolutionCTM* (only for the DLKCF), where true solution of the entire road network is computed by the Godunov scheme in equation (3)-(4) of the paper;

3. *trueSolution*, true solution in each section obtained by selecting the part of the true solution computed by *trueSolutionCTM* that is within the local section. Note that here is no *trueSolutionCTM* package for the Luenberger observer, and the true solution is commputed by the original SMM given in [1,2], with boundary condition assumed to be known from measurements.

4. *section*, used to specify the mode, and to compute the matrices A, B^{ρ}, and B^{q} in the SMM;

5. *model*, used to specify parameters used for estimation (e.g. model/measurement error covariance matrix, output matrix, initial guess of the estimation, and system dynamics used in estimation);

6. *filters*, the distributed Kalman filter and the consensus filter for each agent, the entire estimation process is in *Estimation.exportResult*, which includes all the steps in estimation and functions to export result.

5 Parameters Instruction

5.1 Network setup

5.1.1 The DLKCF and the Luenberger observer in Section V-A

1. *Number of cells in each section*: 10;

2. *Number of overlapping cells between adjacent sections*: 1;

3. *Number of sections in the entire freeway network*: 15;

4. *The index of the section with a status transition located initially*: 7;

5. *Length of the time steps*: 1;

6. *Length of each cell*: 1000d/((double)*cells*), where $cells = 136$ is the total number of cells in the entire freeway network.

Parameters changing instructions: The above listed parameters for the DLKCF can be changed in lines 20-26 in *Test.java* in **DLKCF_SectionV_A**, and for the Luenberger observer the listed parameters can be changed in lines 18-24 in *Test.java* in **Luenberger_SectionV_A**.

5.1.2 The DLKCF and the DLKF in Section V-B

1. *Number of cells in each section*: 28;
2. *Number of overlapping cells between adjacent sections*: 10;
3. *Number of sections in the entire freeway network*: 7;
4. *The index of the section with a status transition located initially*: 3;
5. The lengths of the time steps and the cells are the same as in Section 5.1.1.

Parameters changing instructions: The above listed parameters for the DLKCF can be changed in lines 18-39 in *Test.java* in **DLKCF_SectionV_B** (the DLKF and the DLKCF corresponds to variables indexed by 1 and variables with no indexes, respectively).

5.1.3 The individual local KFs in Section V-B

1. *Number of overlapping cells between adjacent sections*: 1;
2. *Number of sections in the entire freeway network*: 5;
3. *The index of the section with a status transition located initially*: 2;
4. Other parameters are the same as in Section 5.1.2.

Parameters changing instructions: The above listed parameters for the DLKCF can be changed in lines 18-39 in *Test.java* in **DLKCF_SectionV_B** (the individual local KFs corresponds to variables indexed by 2).

5.2 Initial condition of the true state

Initial density (normalized) of the 136 cells (indexed from 0 to 135) are detailed below:
cell0-cell4: 0.2; cell5-cell67: 0.8; cell68-cell129: 0.2; cell130-135: 0.35
Parameters changing instructions: For the DLKCF, DLKF, and individual local KFs, the initial condition can be changed in the method *TrueSolutionCTM.initial* in the package *trueSolutionCTM*. For the Luenberger observer, the initial condition can be set in the method *TrueSolution.initial* in the package *trueSolution*.

5.3 Boundary condition of the true state

5.3.1 The upstream boundary condition

The inflow from the upstream boundary is given by the minimum of a sinusoidal flow and the receiving capacity of the upstream cell:

$$\text{inflow} = \min\left\{0.1125 + 0.1125 \times \sin(\frac{k\pi}{4000} + \pi), R(\rho_k^0)\right\},$$

where k is the time index, $R(\rho_k^0)$ is the receiving capacity of the upstream cell at the current time step.

5.3.2 The downstream boundary condition

Assume the density of the downstream cell is the same as its downstream neighboring cell, and the outflow is computed by the minimum of the sending capacity of the downstream cell and the receiving capacity of its downstream neighboring cell.
Parameters changing instructions:
The boundary condition can be changed in *Estimation.exportResult* (lines 380-408 for the DLKCF in Section V-A, lines 274-286 for the Luenberger observer and lines 564-597 for the DLKCF in Section V-B).

5.4 Parameters and perturbed parameters in the traffic model

1. *Maximal density*: 1 (true), 0.9 (perturbed for sections indexed by even numbers) and 1.1 (perturbed for sections indexed by odd numbers);
2. *Critical density*: 0.225 (true), 0.2 (perturbed for sections indexed by even numbers) and 0.3 (perturbed for sections indexed by odd numbers);
3. *Maximal speed*: 1 (true), 1.2 (perturbed for sections indexed by even numbers) and 0.9 (perturbed for sections indexed by odd numbers).

Parameters changing instructions:

1. Changing parameters in the traffic model used to generate true solution can be done in the constructors *TriangularTrueCTM* and/or *TriangularTrue*;
2. Changing parameters used in the estimators can be done by setting the default parameters (line 1) in the classes *FF*, *CC*, *CF*, and *FC* that extend the class *Section*;
3. The perturbed parameters in **DLKCF_SectionV_B** can be set in the method *setSections()* (lines 158-165) of the class *TrueSolution.java*.

5.5 Important settings for estimation

1. *Model error covariance matrix*: $0.01I$ (DLKCF in Section V-A) and $0.0025I$ (DLKCF, DLKF and individual local KFs in Section V-B);
   Parameters changing instructions: The model error covariance matrix can be changed in the method *defineVarAndMean()* in *RoadModel.java*, and can be updated at each time step by *updateModelVar()* in *RoadModel.java*;

2. *Sensors are located* at the upstream and downstream cells in each local section (for Section V-A), and are located at cells (locally) indexed by 0, 9, 18, 27 in each local section (for Section V-B);
   Parameters changing instructions: The sensor locations can be changed in the method *getMeasurementsAll()* in *TrueSolution.java* by selecting the cells whose densities are queried;

3. *True measurement error covariance matrix* (used to generate sensor data): $0.0081I$ in Section V-A, $0.0009I$ in Section V-B (for the low quality sensors, the measurement error statdard deviation is changed to 0.09)
   Parameters changing instructions: In the case of Section V-A, the true measurement error covariance matrix can be changed in the method *initial()* in *TrueSolution.java*. For filters in Section V-B, the true measurement error covariance matrix can be changed in *initial()* in *TrueSolutionCTM.java* (lines 69-74);

4. *Measurement error covariance matrix* (used in the filter for estimation): Assumed to be the same as the true measurement error covariance matrix (i.e. can provide correct guess of the sensor accuracy) except for the case in Section V-B when low quality agents exist (in that case the measurement error covariance matrix is set to be $0.0009I$ before inter-agent communication);
   Parameters changing instructions: The measurement error covariance matrix used in estimation, when different from the true measurement error covariance matrix (applicable in Section V-B) can be changed in in the method *initialThis()* in *D.java*;

5. *Initial estimates* is defined as a linear interpolation added by some randomly generated Gaussian noise (see *defineVarAndMean()* in *RoadModel.java* for details). When the initial mode of a local section is in FC, the initial estimates of the first three cells in that section are set to be zero, and the initial estimates of the other cells in that section are set to be one.
   Parameters changing instructions: The initial estimates can be changed in *defineVarAndMean()* in *RoadModel.java*.

5.6 Important settings in constructors

This subsection is only applicable to **DLKCF_SectionV_B**, where low quality sensors and agents may exist, and parameters in the traffic model may be perturbed.

1. The last parameter in the constructor *TriangularTrueCTM* is 0 if there is no low quality sensors, and is set to be 1 if the reverse is true;

2. The last three parameters in the constructor *TriangularTrue* stand for whether parameters in the traffic model is perturbed (0 for false and 1 for true), whether low quality agents exits in the network (0 for false and 1 for true) and whether inter-agent communication is conducted in the estimation.

References

[1] L. Munoz, X. Sun, R. Horowitz, and L. Alvarez, "Piecewise-linearized cell transmission model and parameter calibration methodology," *Transportation Research Record*, no. 1965, pp. 183–191, 2006.

[2] I. Morarescu and C. Canudas de Wit, "Highway traffic model-based density estimation," in *Proceedings of the American Control Conference*, vol. 3, 2011, pp. 2012–2017.