ReadMe

Yu Fu

July 25, 2017

1 Folder overview

This folder includes all codes for the platooning project. Different folders have different versions of the code. Every folder has 2 subfolders, which include the codes for the centralized model and decentralized model respectively. All codes are tested using Matlab R2016b and Python 2.7.

For every code, it has the same structure:

- First run the main function called 'main*.m', which calculates all decisions (acceleration, velocity, position) for all cars in 700 seconds. There are 4 scenarios where the platooning system has 3, 5, 7 and 9 cars (including the leading vehicle). The leading vehicle is driving based on the given real-life pattern (modified HWFET pattern). When the code is executed, results of the platooning system will be stored in folder 'result'. It has:

  1. output_a_n.mat (accelerations for vehicles in a n-car platooning model);
  2. output_cost_n.mat (cost function values for vehicles in a n-car platooning model);
  3. output_fuel_n.mat (instant fuel consumption for vehicles in a n-car platooning model);
  4. output_s_n.mat (positions for vehicles in a n-car platooning model);
  5. output_v_n.mat (velocity for vehicles in a n-car platooning model).

- Then you can run the drawing function called 'draw.m', which draws all results and stores them in the folder 'figure'. It has:

  1. cost_n_d.fig (figure for cost functions values in a n-car platooning system with decentralized control);
  2. result_n_d.fig (figure for acceleration, velocity, position, inter-car distance and fuel consumption in a n-car platooning system with decentralized control).

Note that every time you run 'main.m', everything in the folder 'result' will be deleted before executing 'main.m' in order to avoid conflicts. Every time you tun 'draw.m', everything in the folder 'figure' will be deleted before executing 'draw.m'. So you need to save the 'result' and 'figure' properly beforehand.

When you get the results for both centralized and decentralized models, you can compare the results using the folder 'compare'. The function to run is 'compare.m'. The decentralized results are in variable table_d, and the centralized results are in variable table_c. The first row shows the number of cars in the platooning system. The second row shows the mean and the third row shows the variation. The 4th row shows the mpg increase for the platooning system.

2 Project goal

The steps that are needed to finish the projects include:

1. Develop a working platooning model with centralized and decentralized control. The meaning of 'working' is that:

- The platooning performance makes sense with 2 control strategies;
- There is a same set of parameter for cost functions under 2 control strategies.

2. Introduce noise and kalman filter to:
   - Compare the platooning performance under 3 scenarios (platooning without noise, platooning with noise, platooning with noise and kalman filter);
   - See how sensor and plant noise affect the platooning performance, and how to tune parameters in kalman filter to eliminate the noise.
3. Develop the attacker model to attack the platooning system;
4. Use game theory to analyze and design better controllers for the platooning system.

3 Version explanation

3.1 V1

This is the original version using the cost function directly from Dr. Pisu's paper. The cost function is:

$$L_i(t) = \underbrace{\omega_1 \int_{\Delta_t} \frac{Fuel}{v_i(t)}}_{\text{Fuel consumption}} + \underbrace{\omega_2 R_{error}^2 + \omega_6 R_{error'}^2}_{\text{Distance to the front car}} + \underbrace{\omega_3 (v_i(t+1) - v_{i-1}(t+1))^2}_{\text{velocity difference between the (i+1)th car and the ith car}} + \underbrace{\omega_4 a_i(t)^2}_{\text{minimize the acceleration}} + \underbrace{\omega_5 (v_i(t+1) - v_{i+1}(t+1))^2}_{\text{velocity difference between the ith car and the (i+1)th car}} \tag{1}$$

The issue with this version is that the cost function cannot produce good platooning results, where all cars accelerate and decelerate a lot to save fuel consumption. The reason is that there is a bigger advantage of decelerating & accelerating over smooth driving), which is not the case in real life. Also, the cost function gives a tight restriction on the platooning system so that cars cannot achieve smooth driving. Several ways to solve the issue include:

- change the cost function (check V2);
- use PSO or genetic algorithm to find a better set of parameters in the cost function so that the desired performance can be achieved. This will take a longer time.

3.2 V2

Since V1 doesn't produce a good platooning pattern, we engineered the cost function so that the desire performance is achieved. We change the cost function equation (1) in design.pdf to the following:

$$\begin{aligned} L_i(t) = & \underbrace{\omega_1 \int_{\Delta_t} \frac{Fuel}{v_i(t)}}_{\text{Fuel consumption}} + \underbrace{\omega_2 R_{error}^2 + 200\sqrt{d_i(t+1) + 700} + 5(d_i(t+1) + 700)^2}_{\text{Distance punishment to the front car}} \\ & + \underbrace{\omega_3 (v_i(t+1) - v_{i-1}(t+1))^2 + 1000|v_i(t+1) - v_{i-1}(t+1)|}_{\text{velocity difference between the (i+1)th car and the ith car}} + \underbrace{\omega_4 a_i(t)^2}_{\text{minimize the acceleration}} + \underbrace{\omega_5 (v_i(t+1) - v_{i+1}(t+1))^2}_{\text{velocity difference between the ith car and the (i+1)th car}} \end{aligned} \tag{2}$$

This achieves the very good performance on the decentralized control model. When we use the same parameter on the centralized control, the results are not good (the platooning system falls apart). Also, in the future paper, it might be hard to explain why we choose those numbers in the design. We did this mainly to prove that it is possible to achieve a very good performance. Dr. Brooks prefers V1 if we can find the 'right' weights for Dr. Pisu's cost function.

3.3 V3

This version introduces sensor and plant noise to the system. The idea is that there is noise in the reading of the sensors and plants, so that the information shared by different vehicles is not accurate.

Note that, this version is usable when we decide on the cost function between V1 and V2 and change the cost function correspondingly.

3.4 V4

The version develops Kalman Filter to eliminate the sensor and plant noise in V3. Similar to V3, this version is usable when we decide on the cost function between V1 and V2 and change the cost function correspondingly. Check the design.pdf for more details about the Kalman Filter.

3.5 python

This version is Python-based. The reason to choose Python is that it is normally faster than Matlab. It is the same code as V2 using decentralized control. The issue with the python code is that the exponential function in the cost functions caused an overflow error. We tried both math.exp and numpy.exp, but they all have this error and stop the code in the middle of the iterations (in less than 100 seconds). Python library bigfloat with precision 100 has a better performance (stop in around 500 seconds), but still has an overflow issue. Also the bigfloat library makes the code very slow, which has a similar speed as the Matlab version. So there is no point of using Python, so we stop increasing the precision. But I suspect there is a precision (larger than 100) might make the code work.