Exercise Solution: A Fuzzy Controller for the Pole Balancing Problem

Advanced Control lecture
at Ecole Centrale Paris

Anne Auger and Dimo Brockhoff
firstname.lastname@inria.fr

Jan 18, 2013

Abstract

After implementing the pole balancing problem and getting an intuition about how difficult it is to design the linear controller for it, we will now apply fuzzy logic to create a controller that does not use all possible input measurements of the cart.

Solutions are written in blue.

1 Getting familiar with fuzzy implications

Let the temperature T be a fuzzy variable that can belong to the fuzzy set "hot" and to the fuzzy set "moderately hot" the membership functions of which are defined as follows:

$$\text{Hot}(T) = \begin{cases} 0 & \text{if } T < 25° \\ \frac{T-25}{10} & \text{if } 25° \leq T \leq 35° \\ 1 & \text{if } T > 35° \end{cases}$$

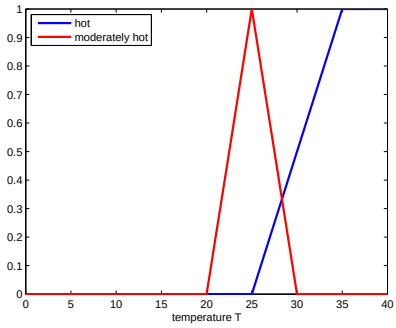
$$\text{ModeratelyHot}(T) = \begin{cases} \frac{T-20}{5} & \text{if } 20° \leq T \leq 25° \\ -\left(\frac{T-30}{5}\right) & \text{if } 25° \leq T \leq 30° \\ 0 & \text{else} \end{cases}$$

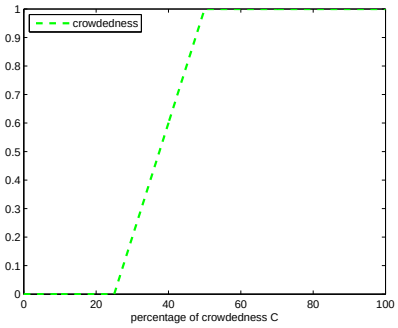
Further, let us define the fuzzy set "Crowdedness" of an ice-cream shop as

$$\text{Crowdedness}(C) = \begin{cases} 0 & \text{if } C < 25 \\ \frac{C-25}{25} & \text{if } 25 \leq C \leq 50 \\ 1 & \text{if } x > 50 \end{cases}$$

where C is the number of customers in the shop.

a) Draw/Plot the membership functions of the three above fuzzy sets. Interesting MATLAB functions to look at: `trimf`, `trapmf`, `plot`, `hold on/off`
With the `Fuzzy Logic Toolbox` of Matlab, the implementation of membership functions is quite easy although it is, in principle, also possible to program them "by hand" as somebody of you did in the exercise. The MATLAB script `crowdednessOfShop.m` in the zip file `icecream.zip` shows a possible way to implement the membership functions with the help of the Fuzzy Logic Toolbox:



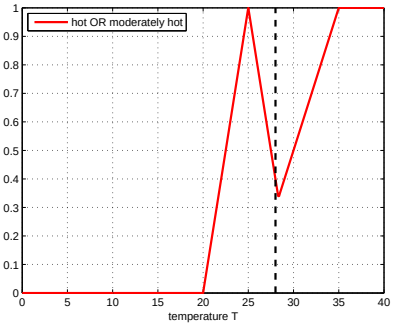


b) Find the implication of the rule

```
IF the temperature is hot OR the temperature is
moderately hot THEN the ice-cream shop is crowded.
```

with respect to Mamdani's rule and Larsen's product rule if the temperature is 28°. In other words, how crowded is the ice-cream shop at a temperature of $T = 28°$ according to the above rule? To this end

I) plot first the membership function of the left-hand side of the implication (see `max` and `min`),

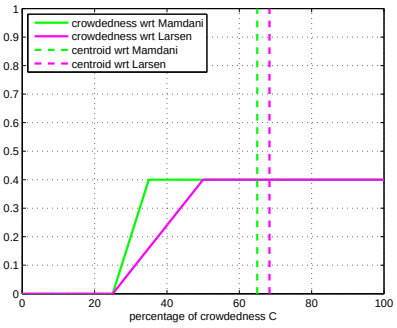


II) then write a function for each of the implication rules (Mamdani/Larsen), and third

III) evaluate the left-hand side membership function at the right temperature (see e.g. `evalmf`) and apply the two implication rules to find out how crowded the ice-cream shop is (plot the resulting membership functions).

IV) Finally, use the centroid defuzzification[1] to get a "crisp" answer whether the ice-cream shop is crowded or not. In which way does it matter which implication function you use?

The script `crowdednessOfShop.m` also shows how to implement this part of the exercise. The final answers according to the two implication rules are that the shop is 64.9655% crowded (Mamdani rule) and 68.3587% crowded (Larsen rule). The following plot shows the resulting membership functions as well as the defuzzified values for both rules.



[1]It might be helpful to look at the `defuzz` function from the Fuzzy Logic Toolbox in MATLAB.

2 A fuzzy controller for the pole balancing problem

Review the pole balancing problem from the previous exercise. All parameters and variables are essentially the same as before except that, for simplicity, we loose the constraints of the track length to $h = \pm 100\text{m}$ and of the pole failure angle to $r = \pm\pi/2$ (90°). In this exercise, the controller does not have access to the direct measurements of all four input values (value and derivative of the position x of the cart and of the angle θ of the pole). Instead, we allow only for the two inputs θ and $\dot{\theta}$ for the moment.

a) Think about how to fuzzify the inputs and outputs of a fuzzy controller for this restricted pole balancing problem. As a recommendation, use three linguistic variables or membership functions per input and five or seven for the output. The inputs to the system are, as mentioned above, the angle θ and its velocity $\dot{\theta}$. The former could be for example modeled with the three membership functions *left*, *middle*, and *right* while for the latter, the names could be *negative*, *okay*, and *positive*. The system's output is the applied force F which could be modeled with the membership functions for *strongly negative*, *negative*, *slightly negative*, *zero*, *slightly positive*, *positive*, and *strongly positive*.

b) Define the membership functions according to your fuzzification. For simplicity (and for the debugging), the sole use of triangular and trapezoidal membership functions is highly recommended. As suggested, only triangular and trapezoidal membership functions are used which, for each variable, overlap slightly between neighboring functions. Open the file `controller_2inputs.fis` within the archive `fuzzycontroller.zip` with the fuzzy logic toolbox for details.

c) Define the rule matrix for your fuzzy controller. Also for the rule matrix, we refer to the file `controller_2inputs.fis` for details.

d) Implement the fuzzy controller with the help of MATLAB's Fuzzy Control Toolkit. Use Mamdani's rule for the implications and the centroid defuzzification.

   I) Start the Fuzzy Control Toolkit by typing `fuzzy` in MATLAB. In the upper half of the new window, you see an abstract view

of your system with the inputs on the left, the implication rules in the middle, and the outputs on the right. Add a second input variable with choosing `Edit-->Add Variable...-->Input` from the menu. Advice: Save your fuzzy system regularly by choosing `File-->Export-->To File...` from the menu.

II) By double-clicking on the input or output variables, a new window opens where you can name and enter your membership functions. The menu allows to add or delete functions. Change the membership functions' shape according to your above design choices by editing the values in the `Params` field or using the mouse.

III) In the initial FIS editor window, double-click on the white middle block with "mamdani" written in the brackets to enter your rules of the rule matrix.

IV) If you have not done yet, save your fuzzy controller as a `.fis` file that you can later on use within your script from last week to get the response for a certain input.

The file `controller_2inputs.fis` contains all functionality of the controller.

e) Test your controller on the pole balancing benchmark from the previous exercise. To this end, use the functions `readfis` and `evalfis` to load your controller and evaluate it, e.g. like this:

```
myController = readfis('NAMEOFYOURCONTROLLER');
F = evalfis([angle_theta angle_velo], myController);
```

The provided files `simulate_fuzzy.m`, `simulate.m`, and `myFuzzyController_2inputs.m` provide the functionality to run the controller as well as plotting the variables of the system over time based on the solution of last week's exercise. Does your fuzzy controller allow to stabilize the pole?
Yes, looking at the angle values and the corresponding velocities over time (see Fig. 1), we can say that the pole is stabilized around $\theta = 0$ after starting with an angle of 0.1. Similar starting angles result in similarly stable behavior.

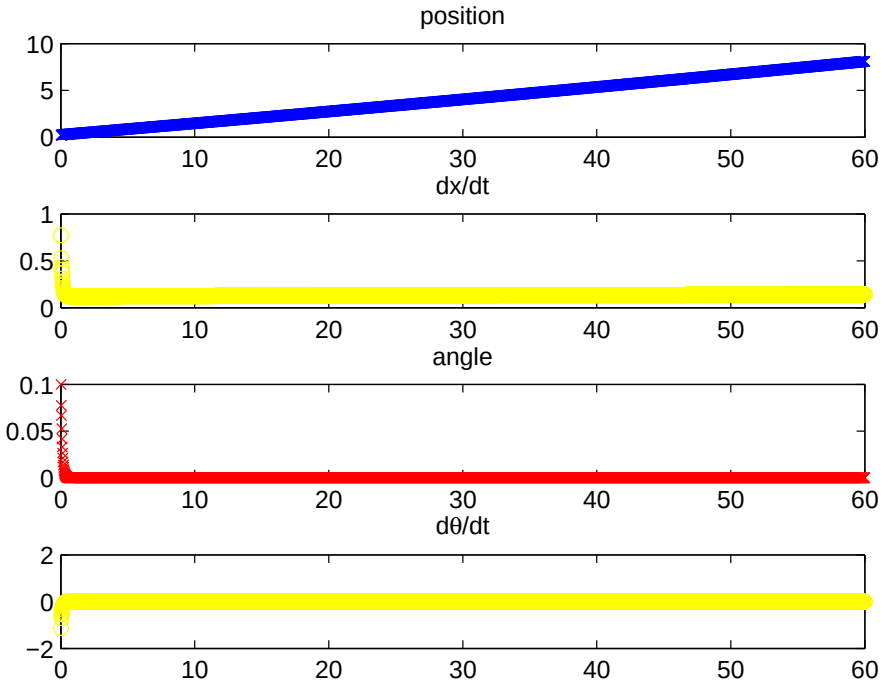


Figure 1: Position (blue), angle (red), and their velocities over time.

f) What is not possible with the current problem formulation and why? The values of the cart's position indicate that, although the pole angle is stable, the cart continues to move in one direction. Simulating longer would cause the simulation to fail due to the constraints on the variable x. The reason is that the measurement of x is not taken into account by the controller and the position of the cart is therefore also not conrolled.

3 Non-Mandatory Questions

If you have more time, you can further play around with the system. For answering those questions, the lecture was too short and no solution will be given here. However, it is indeed easily possible to answer the questions by playing around with the provided code.

a) Is the centroidal defuzzification giving better results than other methods?

b) What about the influence of other parameters of the sytem such as the implication rule, the simulation accuracy, or the initial cart and pole positions?

c) Can you come up with a nice illustration of the system behavior in which you can follow the trajectory of the pole and the cart during the simulation in one single plot? This plot is already contained in the provided code example.