Car control using a fuzzy logic controller (FLC)

Christophoros Bekos (mpekchri@auth.gr)

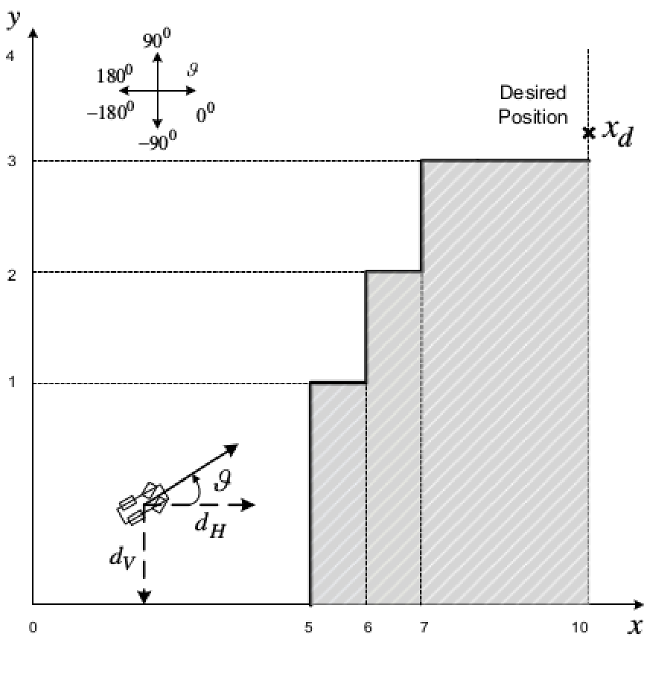
September 30, 2017

Abstract

A fuzzy control system is a control system based on fuzzy logic — a mathematical system that analyzes analog input values in terms of logical variables that take on continuous values between 0 and 1, in contrast to classical or digital logic, which operates on discrete values of either 1 or 0 (true or false, respectively).
Fuzzy logic is widely used in machine control. The term "fuzzy" refers to the fact that the logic involved can deal with concepts that cannot be expressed as the "true" or "false" but rather as "partially true". Although alternative approaches such as genetic algorithms and neural networks can perform just as well as fuzzy logic in many cases, fuzzy logic has the advantage that the solution to the problem can be cast in terms that human operators can understand, so that their experience can be used in the design of the controller. This makes it easier to mechanize tasks that are already successfully performed by humans.

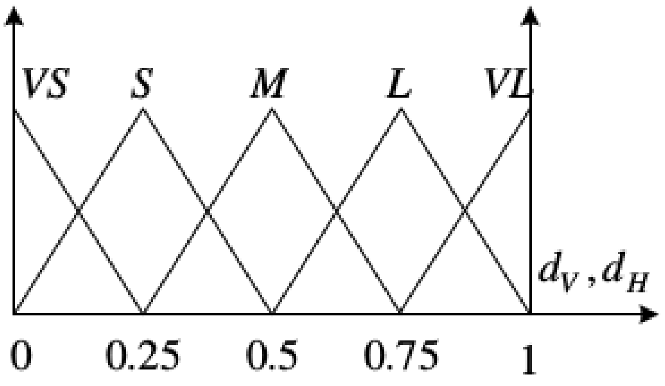
1 Project's task

In this project we will build a FLC ,in order to control a car's movement in a 2-D space . We consider that the car has negligible dimensions (a single point in the Cartesian coordinate plane). In addition , some obstacles in our 2-D space are pre-defined , as shown in figure below . Our task is to create the rule base of our FLC , in a way that, given the starting point x_0 = (4,0.4) and constant speed u = 0.05 m/sec , we achieve to move car in the desired position x_d = (10,3.2) .
Our FLC has 3 inputs , dV, dH ,and theta and 1 output dtheta . dV is the vertical distance from the nearest obstacle , while dH is the horizontal . Theta is our current angle (is symbolized with θ in the figure) , and dtheta is the amount of degrees that will be added to current angle in order to achieve moving to x_d .

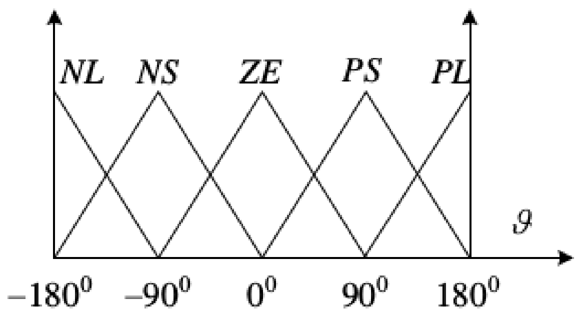


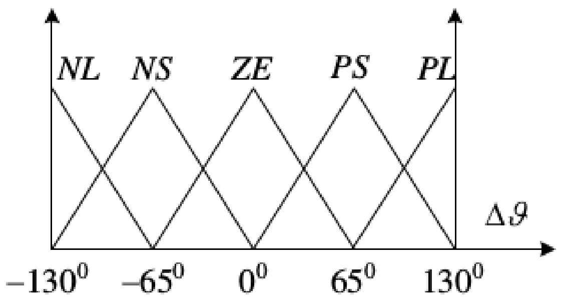
In order to solve this problem we are about to take advantage of fuzzy logic . Input's and output's of our FLC are described by a number of different fuzzy sets :

- dV and dH are being normalized in [0,1] and they are described by 5 fuzzy sets :



- theta belongs in $[-180^o, 180^o]$,while dtheta in $[-130^o, 130^o]$. They are described by the following fuzzy sets :





2 Creating the rule base

Since we know the initial and the desired position ,x_0 and x_d it is not difficult to form the rule base ,based on human experience . For example :

Supposing that theta (current angle) is 0^0 (moving only in x axis) , then :

- **IF** dV is VS **AND** dH is VS **THEN** dtheta is Ps

This means that if car is really close to an obstacle (both in x and y direction) , then you should add arround 65^0 to the current angle (which is 0^0) in order to avoid the obstacle in x direction . The same way we write :

- **IF** dV is S **AND** dH is VS **THEN** dtheta is Ps
- **IF** dV is M **AND** dH is VS **THEN** dtheta is Ps
- **IF** dV is L **AND** dH is VS **THEN** dtheta is Ps
- **IF** dV is VL **AND** dH is VS **THEN** dtheta is Ps

Using the same methodology we can create the whole rule base .
The script named 'create-fis.m' is is responsible for the creation of FLC's rule base , while script 'create-fis2.m' creates the final rule base, which will be used in order to reach the desired position in x-y grid .

3 Simulating car's movement in matlab

The script 'ergasia3.m' simulates and plots the car's movement among x-y grid . But before you read this, you better give a look at the next subsections :

3.1 Normalize angle (theta)

In order our FLC to behave as expected , we need to feed him with values normalized as we have already discussed . The script 'normalize-theta.m' is responsible to normalize the angle .

3.2 Normalize dV and dH

The values of dV and dH are normalized by the commands : dh = abs(dh/10) and dv = abs(dv/4) . However , because of the way that our fuzzy sets are defined , the desired dV and dH will always belong in VS or S , so the rest of the rules will be useless , leading to wrong results . **In order to avoid such a problem we may normalize them by dividing with a smaller number , or to evaluate distance from obstacles in a different way (will be described in the next subsection)**

3.3 Find distances from obstacles (dV and dH)

Script named 'distance-from-barrier' is responsible for finding the values dV and dH according the our current pos (position in x-y grid) . In the end the values of dV and dH that are return are changed a bit (by commands dv = dv+1.8; and dh = dh+1.8 ;) for the reason that we discussed earlier .

3.4 Update position

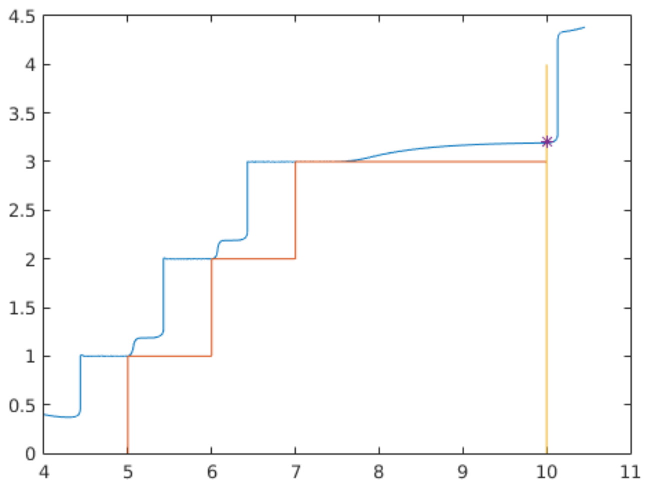
Using the script 'update-position.m' we estimate the next position of the car in the x-y grid . The argument dt represents the time derivative and it's equal to Ts = 0.1 sec (defined in the final script).

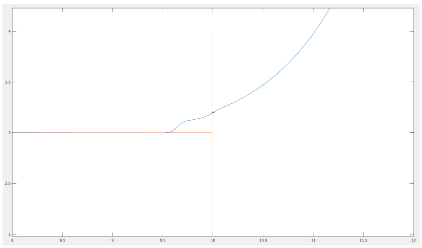
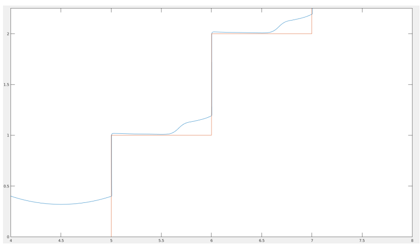
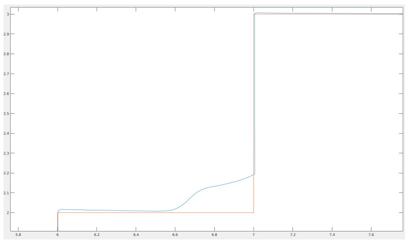
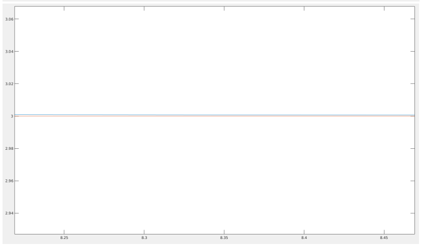
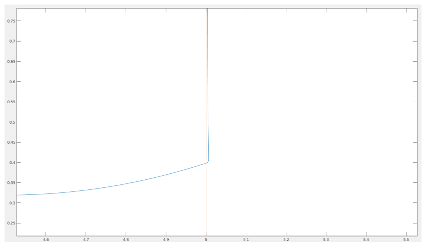
3.5 The final script

Finally , script 'ergasia3.m' simulates the car's movement and plot's it's final destination after a couple of iterations . In each iteration angle (theta) is updated according to equation : theta = theta + dtheta , where dtheta is the estimation of our FLC .

4 Results

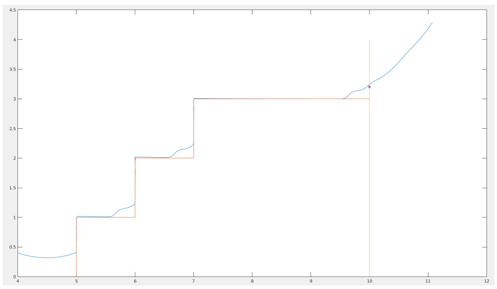
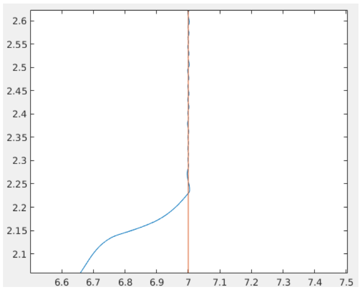
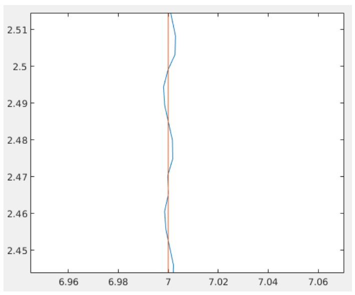
Using the 'create-fis.m' script we have created the fis structure 'fis.fis' . Simulating car's move using this fis ,gives us the following results :



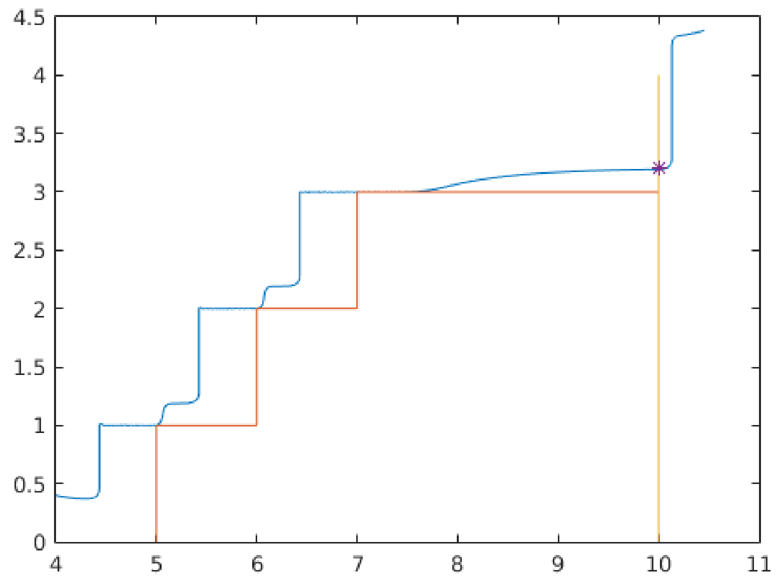






In all figures : red line symbolizes the obstacles , blue line the car's movement , while the marked point in yellow line is the desired position.

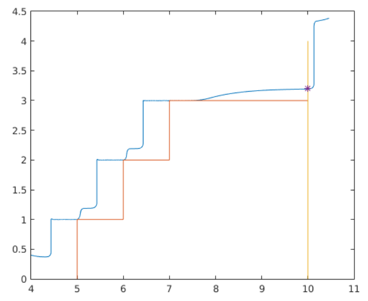
As you can see , we do achieve reaching the desired position , but in many cases we fail to completely avoid obstacles . This need that we must update our rule base :
The script 'create-fis-2.m' describes the way that the new rule base is created (new rule base is saved as 'fis-second-attempt.fis') .In this script i have commented any change that has been made to the initial approach for creating the rule base. The script was modified twice , in first time we got the following results :

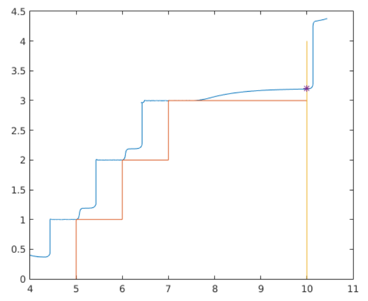




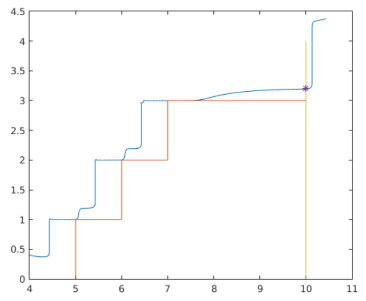
So after making a few more changes , and **defining that dtheta belongs in [-150,150]** we manage to get the desired results :



All those simulations where made when the initial value for angle was 0^0 . We also tried different angles ,like -90^0 , or -45^0 :





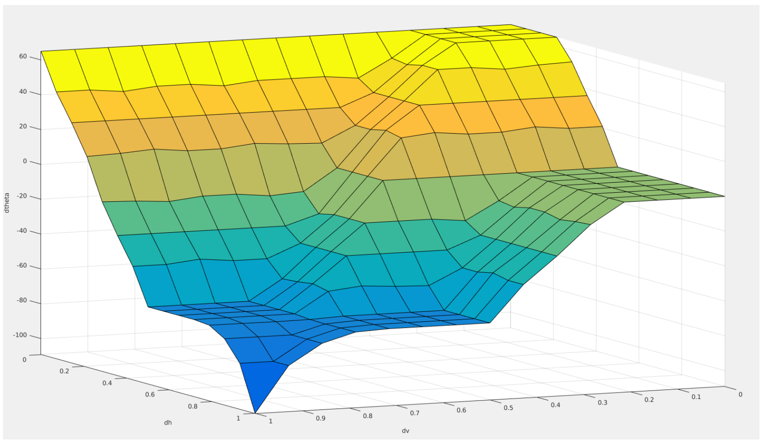


‘ INITIAL ANGLE 0^0 INITIAL ANGLE -45^0 INITIAL ANGLE -90^0

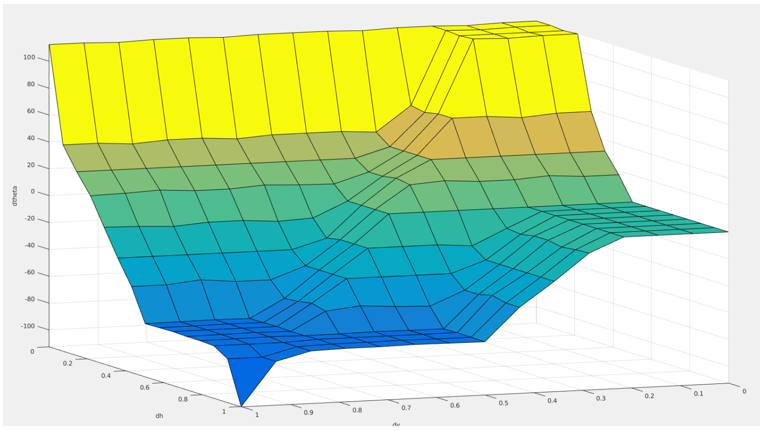
Of course , since we have made a correct definition of the rule base our system will behave the desired way , no matter the initial angle .

5 Rule Base form

Bellow , the figures of two rule bases (the one that failed at first and the final) are given . Because we are dealing with 3-D figures it may would be preferable to check 'rulebase1.fig' and 'rulebasefinal.fig' in project's folder .
First attempt for rule base creation :



Final attempt for rule base creation :


dtheta
dh
dv