Jeff Schubbe
Dr. Mehdi Zargham
CPS 480
21 October 2013

Project 2: Fuzzy Robot Controller

The goal was to implement a fuzzy controller to simulate how a robot should move through space. From a starting position and direction within a set area, the robot will travel towards and northward along the line y=0 that runs vertically through the center of the area. This area is defined from -15 to 15 in the x-direction and extends infinitely in the y-direction.

The robot acts based on a simple set of rules each with two linguistic input values and one linguistic output value. Based on how much the robot is to the “left” of, is to the “right” of, or is “centered” with the center line as well as how much the robot is facing in each of the cardinal directions (“north”, “south”, “east”, and “west”), the robot will have a degree of output for “turning left”, “going straight”, and “turning right”. According to piecewise membership functions for the robot’s distance from the center line and the robot’s direction angle, membership degrees are obtained for each linguistic input value. In the program, an if-elseif block is used to set the membership of each linguistic input value based on which range of the piecewise function that the corresponding linguistic input variable is in.

Based on the set of linguistic rules for the robot’s movement, a linguistic output value is found for each combination of linguistic inputs when the robot is at a certain position and angle. The AND operation is used to form each rule with the two linguistic input variables, which means that the degree of each linguistic output value for each rule would be the minimum of the degrees of the two linguistic inputs for that rule. This is easily implemented in the program with the MATLAB min() function. Finally, a centroid defuzzification method is used to find the weighted average of all linguistic output values. In this scenario, the following is used to calculate the final angle that the robot must turn before moving a small increment, a positive value meaning a right turn and a negative value meaning a left turn. Here, the maximum degrees of each linguistic output value (“turn left”, “go straight”, “turn right”) is used:

$$\beta = \frac{(-15.0*MAX(f_{TL}(\cdot)))+(0.0*MAX(f_{ST}(\cdot)))+(15.0*MAX(f_{TR}(\cdot)))}{MAX(f_{TL}(\cdot))+MAX(f_{ST}(\cdot))+MAX(f_{TR}(\cdot))}$$

The maximum degree of each linguistic output value function is easily determined with the MATLAB max() function with its argument being an array that holds all of the degrees from each rule for one of the linguistic output values. Because a turn right is clockwise, our change of angle β is subtracted from the current direction angle α to find the new α. The robot’s x and y coordinates are updated and stored in an array which will be plotted once all iterations of the robot’s movement have completed.

The final output of the program plots three curves together, each representing the path that the robot would take if it began at different distances from the center and at different direction angles. The sets of inputs tested are as follows:

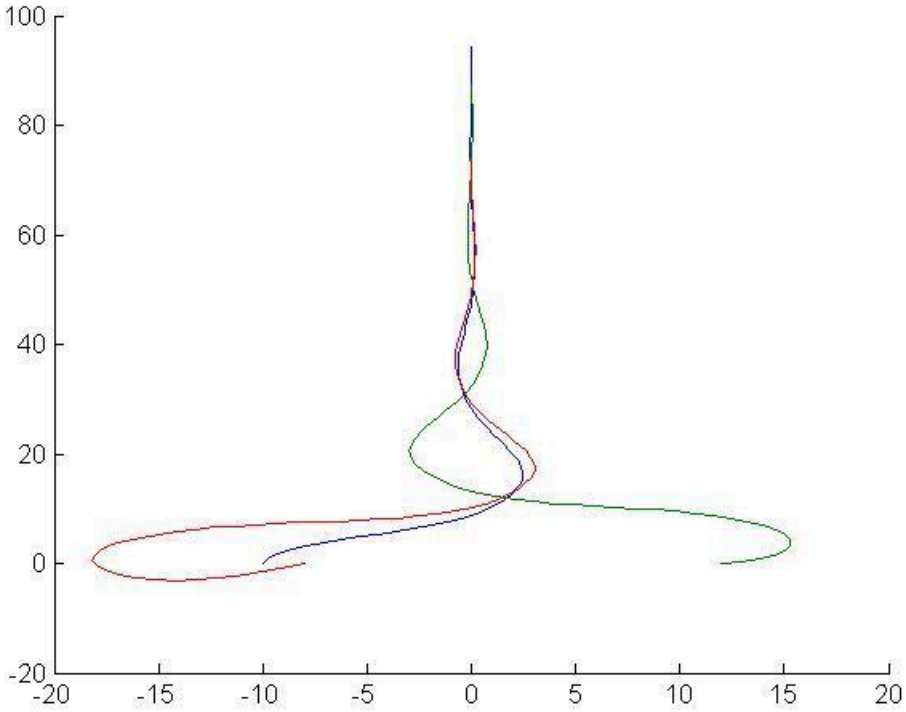
Distance (x)	Direction angle (α) in degrees
-10	89
12	0
-8	220

In order to reach the robot’s desired line of motion, different lengths of distance traveled for each iteration as well as different numbers of iterations of the robot’s movement were used. The first plot below shows each of the three trials tested with 100 iterations with a travel distance of 1 unit for each iteration. This plot shows a stark contrast when compared to the next plot, which was obtained with 300 iterations with a travel distance of 0.1 units for each iteration.

The latter plot shows a much smoother approach to the center line, while the former is stuck traveling further at less optimal angles, often moving away from the center line, simply because the angle is not updated as often. One advantage to the former scenario is that once the robot has reached the center line and is moving mostly straight, it can travel much farther in less iterations. However, more importantly, the number of iterations needed to get to the center line is also reduced, saving time on calculations of the next β value. The former approach may actually be preferred in other more complex systems where calculations are much more involved, but in this case, the latter plot provides a shorter, more direct approach to the desired line of motion, which is especially preferred if the robot can make its calculations faster than it travels in each iteration.

With smaller values of travel distance for each iteration, the plots become even smoother and more direct. However, a larger number of iterations is required to compensate for this and ultimately travel far enough to reach the center line.

1 Unit Traveled Per Iteration (100 Iterations)



0.1 Unit Traveled Per Iteration (300 Iterations)