Robot movement controller in the CiberRato simulation environment

Paulo Pereira[98430]

Perceção e Controlo, Universidade de Aveiro, 3810-193 Aveiro, Portugal

Abstract. This paper covers the approach for the assignment 2 in the course "Perceção e Controlo".

Keywords: Classic Control · Fuzzy Control

1 Introduction

The posed problem was to control the movement of a robot through an unknown closed circuit defined by a line as fast as possible and without leaving the line. To solve this problem we should develop two approaches firstly a robot controller using a more classical controller approach and a fuzzy logic based one. In this report we'll be explaining how we implemented both controllers, comparing results and future improvements.

2 Classical Approach

2.1 Strategy

We took the mainRob.py robot as a baseline, and just revamped the wander function so that using the lineSensor of the robot we could receive the readings that told where the robot was in relation to the line. These readings came in form of a char list (0s and 1s) of 7 elements where the first 3 chars were referring to the right, the center one to the center and the final 3 to the left, this meant if there were more ones in the first 3 than the last 3 the robot should turn left, and the other way around, if the number of 1s were the same in both sides the robot would move forward, with this in mind we used a simple conditional approach using if else statements that when passed would send to the motors the correct wheel speed as well as the direction of that spin.

```
operationline=(operationline[::-1][:3]-operationline[:3])

input=operationline.sum()
print(operationline)
print(input)
if input< 0:
    line[-3:] = [0,0,0]
elif input>0:
    line[0:3] = [0,0,0]
```


```
operationline=(operationline[::-1][:3]-operationline[:3])

input=operationline.sum()
print(operationline)
print(input)
if input< 0:
    line[-3:] = [0,0,0]
elif input>0:
    line[0:3] = [0,0,0]
```


Fig. 1. Relevant code for list splicing and movement conditions

2.2 Results

Using this approach in the CiberRato environment we found that after 15 attempts the robot never got out of line, did all corners perfectly and averaged around 4300 points. It also should be mentioned that the wheel speed was a fixed constant to which we tested in order to find the optimal value which is 0.15.

Fig. 2. Results for Classic Control

3 Fuzzy Control

3.1 Strategy

When doing this approach, firstly we weren't really sure on how to implement it, but the documentation of the skfuzzy library was pretty helpful both with a very good example on how to implement it and with great depth about the most useful commands.

For fuzzy we also took mainRob.py from lab classes as a baseline and changed the wander function in order to accommodate the fuzzy controller, instead of just interpreting the line like in the previous solution, we spliced the list into the left and right components using splicing and subtraction in order to find a list with 3 elements containing the difference, if this list was 0 the robot would move forward despite the value in the middle, we also gave this 3 numbers different weights in order to simulate how hard the robot should turn (for example [1 -1 1] would turn harder right than [0 0 1]), after having this vector and its sum we set up the fuzzy system. Firstly the we used the previous sum as the input

```
line=(line[::-1][:3]-line[:3])
c=4
#give them the values
#a diferença das listas pode ser
# [ 1 1 1] =7
# [1 1 0] = 6
#  [1 0 1] =5
# [1 0 0] =4
# [0 1 1] =3
#  [0 1 0]= 2
#  [ 0 0 1] =1
# há 7 possibilidades
#consoante o numero e a posiçao dos 1 (à esquerda é maior)
# mais rapido ele deve fazer a ação
for i in range(len(line)):
    line[i]*=c
    c-=c/2

input=line.sum()+7
```

Fig. 3. Vector definition

this number would then be evaluated by the membership functions where if it was 7 the robot would go forward, below 7 left and above it right, the max turn would be achieved at 0 and 15.

We then set up the consequent as a variable that would be a factor in the turn speed (if it came out as 1 the robot would turn the defined wheel speed). The range of the membership functions is from -15 to 15 where both of these mean the maximum turn (left or right, accordingly).

We then set up the 3 rules that we can see in the picture below. And finally the control system as the documentation explained. After getting the output the

```
# Define fuzzy rules

rule1 = ctrl.Rule(line_position['left'], movement['left'])
rule2 = ctrl.Rule(line_position['center'], movement['center'])
rule3 = ctrl.Rule(line_position['right'], movement['right'])
```

Fig. 4. Rules for Fuzz Control

formula below would return the speed that we would send to each motor in order to make the robot moving using a simple if else.

```
#left
if speed<0:
    speed=(speed/10)*-wheel_speed
    print("left" + str(speed))

    self.driveMotors(-speed,+speed)
elif speed>0:
    speed=(speed/10)*wheel_speed
    print("right" + str(speed))
    self.driveMotors(+speed,-speed)

else:
    print("fwd" + str(speed))
    self.driveMotors(wheel_speed,wheel_speed)
```

Fig. 5. Fuzzy Control Formula

3.2 Results

With this approach we found that in 20 attempts the robot got once out of line and that it was a bit slower than the previous method averaging about 3000 points. This lack of performance can be attributed to the fact we only used 3 rules to implement the system, more rules would mean a lot more speed. The value for the wheel speed was also set at 0.15 for a fairer comparison with the previous method.

Fig. 6. Results for Fuzzy Control

4 Conclusion

To sum up this project, we can conclude that the fuzzy system could be better implemented with more rules, but that overall both solutions achieve good results. We couldn't quite grasp where the bug in the 1 deviated run happened using the fuzzy control.