ARTIFICIAL INTELLIGENCE AND ROBOTICS

LM IN ARTICIAL INTELLIGENCE

Assignment 3

SUBMISSION 1

Students :
Álvaro ESTEBAN MUÑOZ

Teacher :
Alessandro SAFFIOTTI
Andrea GOVONI

May 1, 2024

Contents

1 Implement a "GoToTarget" behavior, using the given fuzzy rule-based controller.

1.1 The problem

The idea of this first task of the assignment is to learn how the given fuzzy rule-based controller works by modifying the already implemented "GoToTarget" behavior. This behavior and all the behaviors we will implement during this assignment will be part of the "Controller" component, which is in charge of "Decide action" step of the Robot Control Program (RCP).

Check figure [1]. The second step will be computed with the fuzzy controller given, while we will be in charge of creating the rules and selecting the values for each control variable. Namely for this task we will only have to change the values and may be add some rules to optimize the given "GoToTarget" behavior.

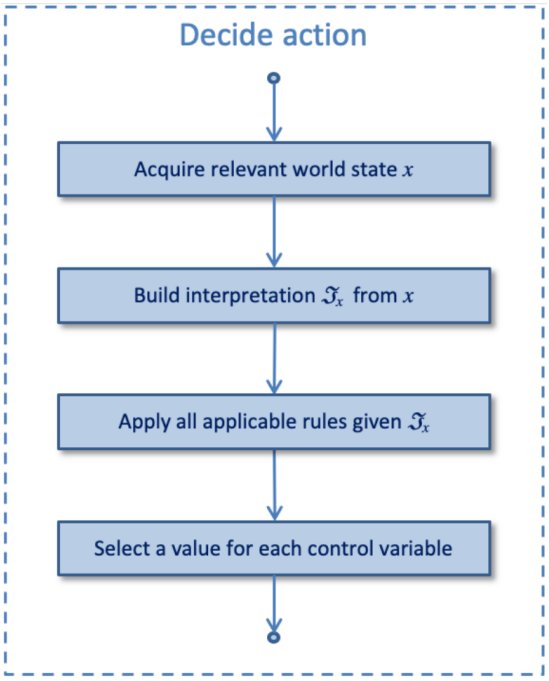

Figure 1: Decide action process

1.2 Objects and Data Structures

We will not enter in details on the given fuzzy controller. However we need to understand the overall structure of the new components. All the classes and functions needed for the proper functioning of the fuzzy rule-based controller are contained in fcontroll module. Inside we can differ the following components

- **FController**: this class is in charge of realizing the action in the above figure [1].
- **FEval**: this one implements the functions used by the FController to evaluate the truth value of the fuzzy sentences.

- **Behavior**: finally, this class is a kind of interface which extends the FController adding the needed functions we have to override in order to implement a new desired behavior.

It is important we understand the main methods that we need to implement for each behavior. These methods are the `update_state` and `setup`. The first one is in charge of updating the chosen state variables that will be important so to activate their associated fuzzy predicates. The second method defines all the rules of our specific behavior, so for example on this first assignment we are given a first set of rules:

- TargetLeft $\wedge\neg$ TargetHere $\implies$ Turn(Left)
- TargetRight $\wedge\neg$ TargetHere $\implies$ Turn(Right)
- TargetAhead $\wedge\neg$ TargetHere $\implies$ Move(Fast)
- TargetHete $\implies$ Move(None)

In our system we also need to define the fuzzy predicates like `TargetLeft` and the linguistic variables like `Move(`a_1`)`.

$$\text{TargetLeft}(\phi, a, b) \begin{cases} 0 & \phi < a \\ 1 & \phi > b \\ \frac{\phi - a}{b - a} & \text{otherwise} \end{cases}$$

$$\text{TargetRigth}(\phi, a, b) \begin{cases} 1 & \phi < a \\ 0 & \phi > b \\ \frac{b - \phi}{b - a} & \text{otherwise} \end{cases}$$

$$\text{TargetAhead}(\phi, a, b, c) \begin{cases} 0 & \phi < a \vee \phi > c \\ \frac{\phi - a}{b - a} & \phi > a \wedge \phi < b \\ \frac{c - \phi}{c - b} & \text{otherwise} \end{cases}$$

$$\text{TargetHere}(\rho, a, b) \begin{cases} 1 & \rho < a \\ 0 & \rho > b \\ \frac{b - \rho}{b - a} & \text{otherwise} \end{cases}$$

These are the mathematical definitions of our fuzzy predicates, even if they seem complex they are just implementations of simple ramp functions.

The given linguistic variables are two, each one with one argument:

- Move(a_1): possible values for its argument [Fast, Slow, None, Back]
- Turn(a_1): possible values for its argument [Left, MLeft, None, MRight, Right]

Each linguistic variables affect to a state variable, namely Move affects the linear velocity (VLin) and Turn affects the rotation velocity (VRot).

1.3 Parameters

Given Parameters As we said in the previous section. There are some parameters that are given and others that are computed during the execution of the program. In this section we describe which parameters were given and which ones we change in order to improve the performance of the "GoToTarget" behavior.

To begin with, notice the fuzzy predicates defined in the previous section have some parameters. On the one hand, ϕ and ρ are the two state variables defined for the behavior, these are the angle and the distance to the target. On the other hand, a, b and c are just parameters indicating the extremes of the ramp functions. Keep in mind even if we call them the same, they change from one function to another. Here is a list of the given values for these parameters.

- TargetLeft: $(a, 30^\circ), (b, 60^\circ)$
- TargetRight: $(a, -60^\circ), (b, -30^\circ)$
- TargetAhead: $(a, -60^\circ), (b, 0^\circ), (c, 60^\circ)$
- TargetHere: $(a, 0.1), (b, 2.0)$

In addition, for the linguistic variables, each argument corresponds to a specific mathematical value for the state variables they affect.

- Move : {Fast: 0.5, Slow: 0.1, None: 0, Back: -0.1}
- Turn : {Left: 40, MLeft: 10, None: 0, MRight: -10, Right: -40}

For example. Move(Fast) corresponds to a value of 0.5 on the Linear Velocity (VLin) variable.

Changed Parameters As we stated on the introduction, for the achievement of this task we were asked to change some parameters in order to improve the perforamce of the "GoToTarget" behavior. We added two new rules, these rules are in charge of turning slowly when te robot is close to the target, just to improve the precision on the last part of the trip.

- TargetLeft $\wedge$ TargetHere $\implies$ Turn(Mleft)
- TargetRight $\wedge$ TargetHere $\implies$ Turn(MRight)

We also changed the range of the angles to consider a target left or right.

- TargetLeft: $(a, 10^\circ), (b, 45^\circ)$
- TargetRight: $(a, -45^\circ), (b, -10^\circ)$

In this way we consider that if the target is further than 45° on each of the two directions it will consider it on the left or right, while the range starts to be consider from 10°.

1.4 Results Obtained

We gave the same goal to the robot and performed two experiments, we tested the behavior with the given parameters and plotted the trajectory followed by the robot as done in assignment 1 [2]. We then did the same with the changes we made [3].

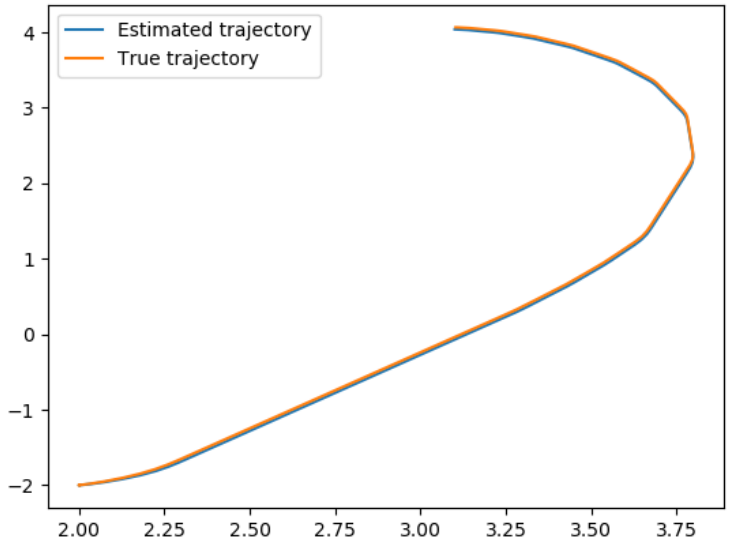

Figure 2: Trajectory performed by the robot with the given parameters.

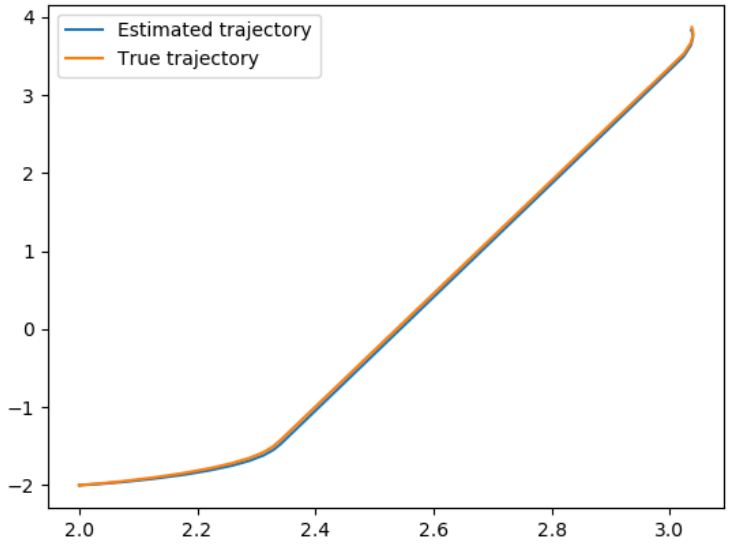

Figure 3: Trajectory followed by the robot with the applied changes.

1.5 Discussion

As we can observe in our results, after adding our two new rules and changing the parameters we obtain a more clean trajectory. Notice the given goal was (3, 4), while on figure [2] the robot passes the coordinate 3 and have to go back, on figure [3] the robot goes directly to the point and therefore does not make useless steps.

2 Implement an "Avoid" behavior to stay away form obstacles

2.1 The problem

For this second task we are asked now to implement our own behavior, namely the robot needs to avoid every object in the room. We won't enter in details on the fuzzy controller since we already saw it on the previous sections.

2.2 Objects and Data structures

To define our behavior we have to implement a set of components we state on previous section. These are the state variables, the fuzzy rules and the linguistic variables.

State variables In order to detect the objects around we will use the sonar sensors from previous assignment. We establish four state variables; frontSonar, leftSonar, rightSonar, backSonar.

Fuzzy predicates For these we use the ramp functions we used on the previous task but in this case they will activate for our new state variables:

$$\text{ObstacleLeft}(\text{leftSonar}, a, b) = \begin{cases} 1 & \text{leftSonar} < a \\ 0 & \text{leftSonar} > b \\ \frac{b-\text{leftSonar}}{b-a} & \text{otherwise} \end{cases}$$

$$\text{ObstacleRight}(\text{rightSonar}, a, b) = \begin{cases} 1 & \text{rightSonar} < a \\ 0 & \text{rightSonar} > b \\ \frac{b-\text{rightSonar}}{b-a} & \text{otherwise} \end{cases}$$

$$\text{ObstacleAhead}(\text{frontSonar}, a, b) = \begin{cases} 1 & \text{frontSonar} < a \\ 0 & \text{frontSonar} > b \\ \frac{b-\text{frontSonar}}{b-a} & \text{otherwise} \end{cases}$$

Fuzzy rules Here we state the defined basic rules for the functioning of the "Avoid" behvior.

- ObstacleRight $\implies$ Turn(Left)

- ObstacleLeft $\implies$ Turn(Right)
- $\neg$ ObstacleAhead $\implies$ Move(Fast)
- ObstacleAhead $\wedge$ ObstacleRight $\wedge$ ObstacleLeft $\implies$ Move(Back)

Fuzzy goal We also need to set a goal which will be used in order to measure the degree of achievement of the task. On the previous task was quite obvious because we just could used the goal, for this task instead, we set the following rule:

- $\neg$ObstacleAhead $\wedge$ $\neg$ObstacleRight $\wedge$ $\neg$ObstacleLeft

Linguistic variables Finally the linguistic variables are the same as for the previous task but changing the parameters. We will see the chosen parameters on the next section.

2.3 Parameters

Parameters were chosen carefully to guarantee safety of the robot. Parameters corresponding to the fuzzy predicates (a, b) where fixed to the range of the sonar sensors, i.e. [0.0, 3.0]. While for the parameters of the linguistic variables we applied the following assignment:

- Move: {Fast: 0.3, Slow: 0.1, None: 0, Back: -0.1}
- Turn: {Left: 30, MLeft: 10, None: 0, MRight: -10, Right: -30}

2.4 Results Obtained

After executing the simulation for about four minutes the robot did not collapsed. Its trajectory and measured time in Gazebo can be checked on figure [4]

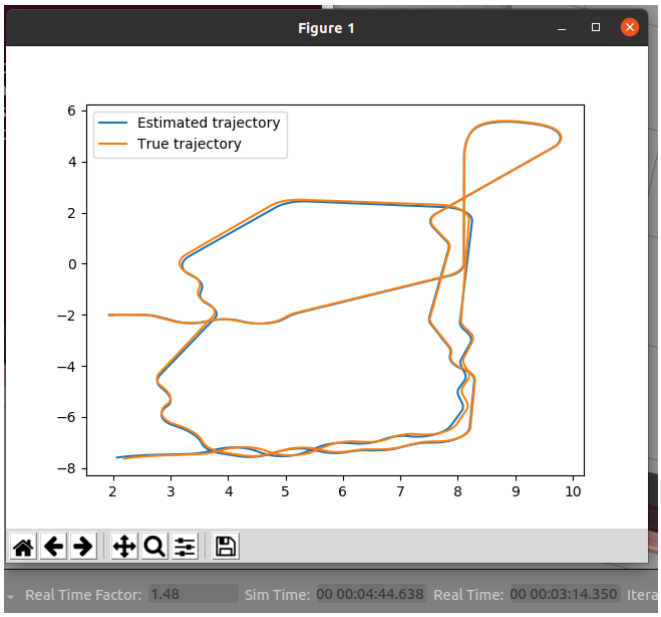

Figure 4: Trajectory followed by the robot using the "Avoid" behavior

2.5 Discussion

It was not easy to reach a good behavior to avoid all obstacles, we had to work on the speed to be sure the robot was not getting too close to the walls or tables. Also the sonar sensors were a bit challenging. In the end we decided to use sonars 1, 2, 7 and 12, which are the ones corresponding to degrees $0°$, $30°$, $180°$ and $330°$, because they were the ones giving the best results.

3 Implement a "FollowObject" behavior to keep a given distance form a perceived object

3.1 The problem

In this task, the proposed behavior is to follow an object that we manually move from the GUI in Gazebo. To avoid problems with the rest of the objects in the world we will assume we can remove them and just focus on the robot following a cube that we will add to the world.

3.2 Objects and Data structures

As done in the previous two tasks, here we define the components for the behavior. All the sections not specified remain as in task 2.

Fuzzy predicates For this task we defined three ramp down functions in case the object is too close to the robot and other three triangle functions which just avoid that the sonar returning the maximum value provokes the robot to detect the object:

$$\text{ObstacleLeft}(\text{leftSonar}, a, b, c) \begin{cases} 0 & \text{leftSonar} < a \vee \text{leftSonar} > c \\ \frac{\text{leftSonar}-a}{b-a} & \text{leftSonar} > a \wedge \text{leftSonar} < b \\ \frac{c-\text{leftSonar}}{c-b} & \text{otherwise} \end{cases}$$

$$\text{ObstacleRight}(\text{rightSonar}, a, b, c) \begin{cases} 0 & \text{rightSonar} < a \vee \text{rightSonar} > c \\ \frac{\text{rightSonar}-a}{b-a} & \text{rightSonar} > a \wedge \text{rightSonar} < b \\ \frac{c-\text{rightSonar}}{c-b} & \text{otherwise} \end{cases}$$

$$\text{ObstacleAhead}(\text{frontSonar}, a, b, c) \begin{cases} 0 & \text{frontSonar} < a \vee \text{frontSonar} > c \\ \frac{\text{frontSonar}-a}{b-a} & \text{frontSonar} > a \wedge \text{frontSonar} < b \\ \frac{c-\text{frontSonar}}{c-b} & \text{otherwise} \end{cases}$$

$$\text{ObstacleCloseL}(\text{leftSonar}, a, b) = \begin{cases} 1 & \text{leftSonar} < a \\ 0 & \text{leftSonar} > b \\ \frac{b-\text{leftSonar}}{b-a} & \text{otherwise} \end{cases}$$

$$\text{ObstacleCloseR}(\text{rightSonar}, a, b) = \begin{cases} 1 & \text{rightSonar} < a \\ 0 & \text{rightSonar} > b \\ \frac{b - \text{rightSonar}}{b - a} & \text{otherwise} \end{cases}$$

$$\text{ObstacleClose}(\text{frontSonar}, a, b) = \begin{cases} 1 & \text{frontSonar} < a \\ 0 & \text{frontSonar} > b \\ \frac{b - \text{frontSonar}}{b - a} & \text{otherwise} \end{cases}$$

Fuzzy rules The rules for this behavior are quite simple. If the robot detects it is too close to the object it stops, other, while the object is in range follow it.

- ObjectLeft $\implies$ Turn(Left)
- ObjectRight $\implies$ Turn(Right)
- ObjectAhead $\implies$ Move(Fast)
- ObjectClose $\implies$ Move(None)
- ObjectCloseR $\implies$ Move(None)
- ObjectCloseL $\implies$ Move(None)

Fuzzy goal Our goal for this behavior is also simple, just keep the object close so we measure the achievement degree by the predicate ObjectClose.

Linguistic variables Once more the linguistic variables are the same as for the two previous tasks.

3.3 Parameters

The only parameters that we need to define here are a, b and c for the functions of the fuzzy predicates. For all triangle functions we use $(a : 1.0, b : 2.0, c : 3.0)$, while for all ramp down functions we use $(a : 0.2, b : 2.0)$. These parameters assure us the robot won't go closer than 0.2 meters keeping always a distance higher than that.

3.4 Results obtained

We plotted the path followed by the robot were we tried to make him perform some sort of circle (figure [5]).

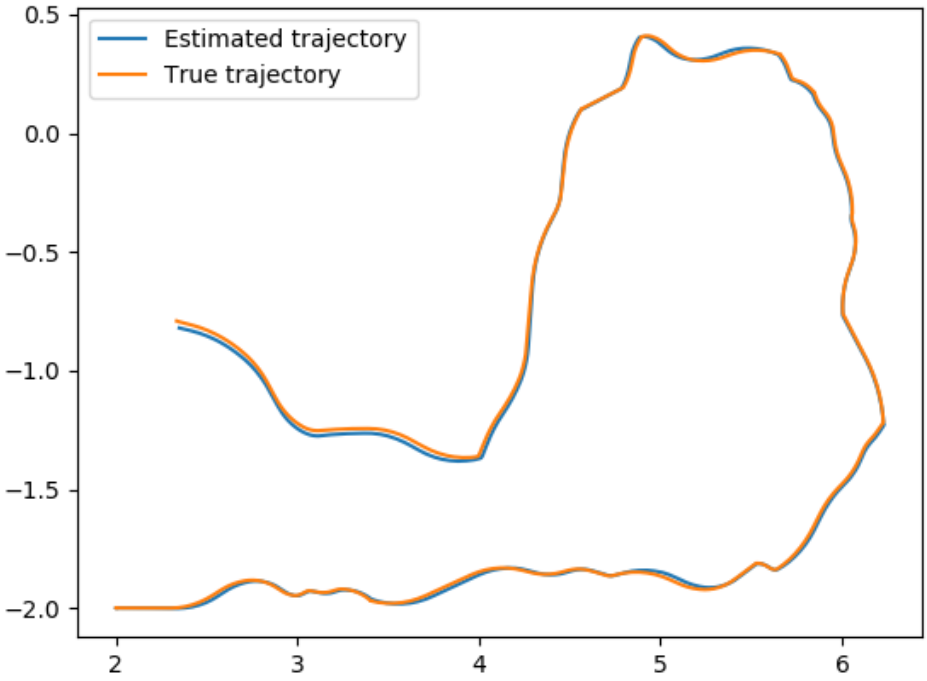

Figure 5: Trajectory followed by the robot using the "FollowObject" behavior

On figure [6] it can be seen the robot with the block used as object to be followed, notice we removed the tables since there is not the center table anymore on the picture.

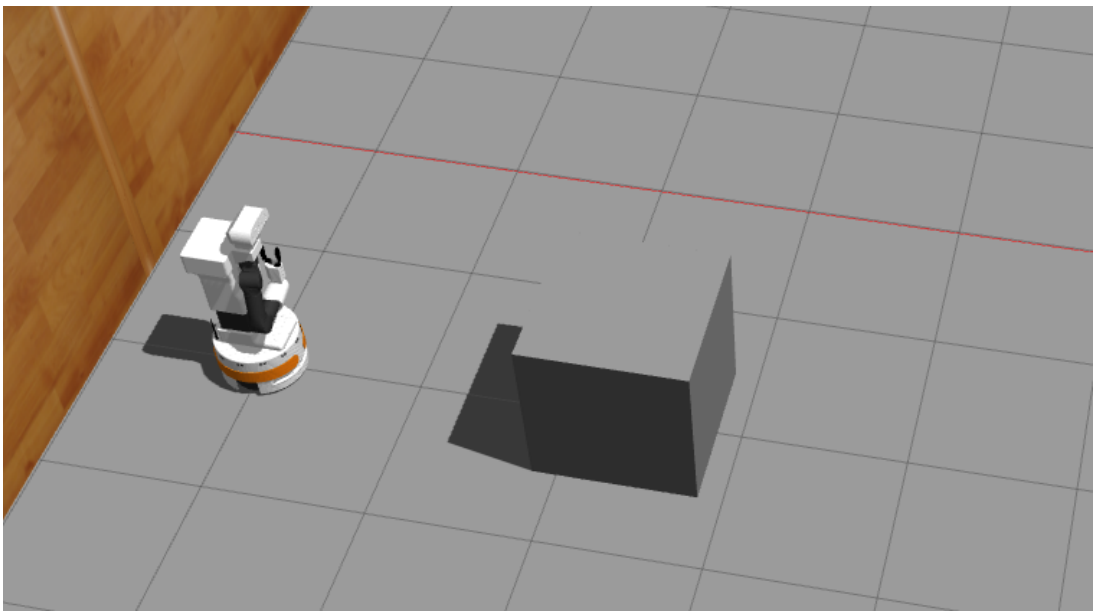

Figure 6: Image of the world

3.5 Discussion

From the results obtained we can confirm our behavior works succesfully. We had to remove the rest of the objects from the world, otherwise the robot was having a lot of problems to not confuse what object it had to follow. Moreover, even without any objects, gettint too close to the walls might "bug" the robot so of course we can also state that this behavior is not perfect.