Fuzzy Logic Controller With Fuzzylab Python Library and the Robot Operating System for Autonomous Mobile Robot Navigation

Submitted: 20th December 2019; accepted: 30th March 2020

Eduardo Avelar, Oscar Castillo, José Soria

DOI: 10.14313/JAMRIS/1-2020/6

Abstract:
The navigation system of a robot requires sensors to perceive its environment to get a representation. Based on this perception and the state of the robot, it needs to take an action to make a desired behavior in the environment. The actions are defined by a system that processes the obtained information. This system can be based on decision rules defined by an expert or obtained by a training or optimization process. Fuzzy logic controllers are based on fuzzy logic on which degrees of truth are used on system variables and has a rule-base that stores the knowledge about the operation of the system. In this paper a fuzzy logic controller is made with the Python fuzzylab library which is based on the Octave Fuzzy Logic Toolkit, and with the Robot Operating System (ROS) for autonomous navigation of the TurtleBot3 robot on a simulated and a real environment using a LIDAR sensor to get the distance of the objects around the robot.

Keywords: *Fuzzy controller, Mobile robot navigation, Obstacle avoidance*

1. Introduction

The goal of autonomous mobile robotics is to build physical systems that can move without human intervention in real world environments [13]. One of the mosts important tasks of an autonomous system of any kind is to acquire knowledge about its environment. This is done by taking measurements using sensors and then extracting meaningful information from those measurements [14].

The controllers of the robots are the mechanism to handle the actuators based on what is received by the sensors. There are many kind of controllers for autonomous robot navigation but in this paper we use fuzzy logic controllers (FLCs) for this task. There are many works that use FLCs for robot navigation where in most of them the controller is implemented only in a simulated way [6, 11, 12], leaving the uncertainty about the behavior that controller could have in a real environment, but also there are works where physical robots are used [7]. In this paper a FLC is created working in both a simulate and real environment. This work pretends to be a starting point to use the fuzzylab library for creating fuzzy logic controllers in Python language for ROS, showing that it is possible create FLCs that operate successfully in real environments. In the next sections we will talk about why use fuzzy logic in controllers and how to create a basic controller for the TurtleBot3 robot.

1.1. Many-valued Logic

Imagine a sensor that can detect the presence of an object up to a distance of 3 meters (m), in the absence of an object the sensor sends a voltage of 0 volts and in the presence of an object within the range it sends a voltage of 5. This is an example of two-valued or bivalent logic because only two values are obtained from the sensor. Now imagine a sensor that cannot only detect the presence of an object, but also measure the distance from the sensor, the sensor sends a voltage depending on the distance to the object. This is an example of many-valued logic where the number of values depends of the sensor resolution.

Usually, when a person talks about the distance to an object using terms such as *near* or *far*, it does not mention the exact distance that it has to the object because it does not really know it. With the visual distance perception to an object we can determine if the object is near or far depending of our own criteria, therefore, it is relative for each person. Going back to the previous example, for a person, *near* could be a distance around or less than 0.75m and *far* could be a distance around or more than 2.25m, but, what happens between those values? is a distance of 1.5m near or far? or is the distance half near and half far?. This uncertainty can be processed and interpreted by fuzzy logic that is a form of many-valued logic.

Considering the distance sensor as the eyes of a mobile robot, we can make the robot interpret the distance as a linguistic variable with the linguistic values *near* and *far* [18] and not only numerical, bringing it closer to a more human rationing. In the "Fuzzy Logic" section we will see how to handle this linguistic variables with fuzzy logic.

1.2. Linear and Nonlinear Systems

Suppose that we want to control the linear velocity of a mobile robot depending of the distance to an object in front of it, simulating a breaking system. The robot has a maximum linear velocity of 0.22 m/s which it can go to if it is above the maximum distance that can be detected by the distance sensor that is 3m and as it gets closer to an object its speed will decrease.

The velocity (v) can be determined with a linear correspondence to the distance (d) described in the Eq. 1.

$$v = \frac{0.22}{3}d \tag{1}$$

The sensor can sense distances above 0.12m, this detects when it is out of range and the distance varia-

ble gets the value of 0 or 3 when this happens as shown in Fig. 1.

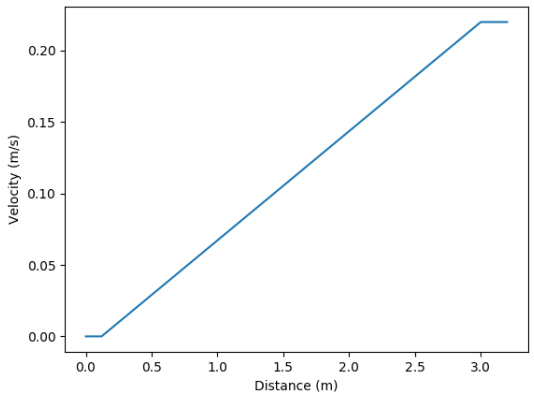


Fig. 1. Velocity control with distance linear relation

The linear behavior of the velocity in respect to the distance is a simple model of how we can determine the velocity, but, what if we want the velocity to have a different behavior in respect to the distance, maybe a smoother transition when the distance changes from 3.1m to 2.9m (see Fig. 1) for example, it is necessary to implement nonlinear models when the system behavior cannot be modeled mathematically. Fuzzy logic allows us to create nonlinear systems depending on how the system designer wants the system to behave. In the "Fuzzy Logic" section we will see the behavior of a nonlinear system.

1.3. Fuzzy Logic

Fuzzy logic and fuzzy sets were introduced by Lofti Zadeh [17] in 1965, these are usefully to model the behavior of nonlinear systems. In fuzzy logic the linguistic values are not entirely true or false, they have some degree of membership defined by membership functions (MFs).

If we section the distance perceived by a robot using a distance sensor in the regions *near* and *far*, we can define partitions with a full membership of each linguistic value. This kind of partitions are called crisp partitions that are a zero-order uncertainty partition [10] where the degree of membership in each region is 1 as shown in Fig. 2, therefore, they do not allow any uncertainty between *near* and *far* linguistic values.

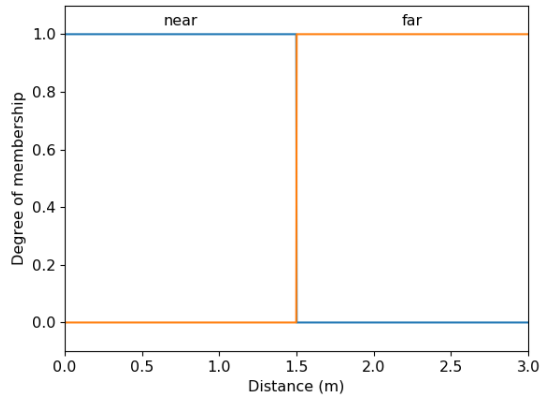


Fig. 2. Crisp distance partitions

This generates a sharp transition from one term to the next. Fig. 3 shows the transition from *near* to *far*.

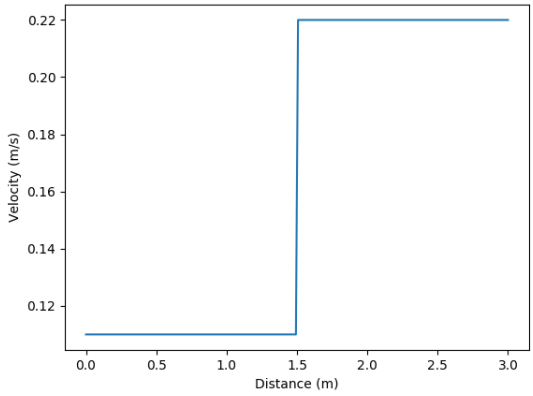


Fig. 3. Sharp transition between linguistic values

With fuzzy logic we can define regions where the degree of membership of the regions are not always 1, this is done using MFs for each linguistic value. There are many different kinds of membership functions, in control tasks, the most used are the trapezoidal-shaped and triangular-shaped MFs. The gaussian MF generates smoother transitions but requires more computational resources but for a better smoother transition visualization we will use it for the example of velocity control. The gaussian function depends on two parameters, σ for standard deviation and center (mean) value.

A rule-based fuzzy system contains rules, fuzzifier, inference, and output processor components. Rule antecedents are in terms of variables that can be observed or measured [10], therefore, in the velocity control example the distance is an antecedent variable.

In Fig. 4 two gaussian MFs are defined for *distance* linguistic value that form a part of the inference process and in Fig. 5 the output of the fuzzy inference system (FIS) that defines the value of the velocity is shown.

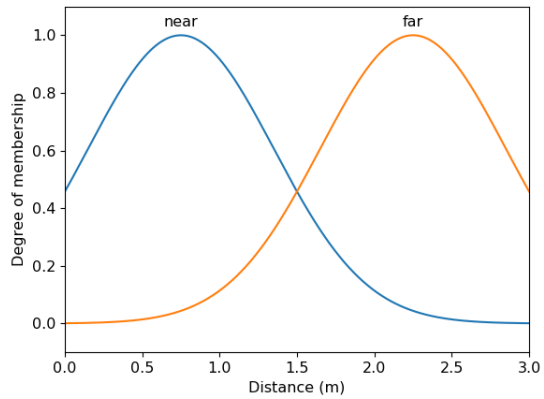


Fig. 4. MFs of *d* (distance) = {*near*, *far*}

In "Design of the Fuzzy Logic Controller" section we explain how to create fuzzy systems with Python language.

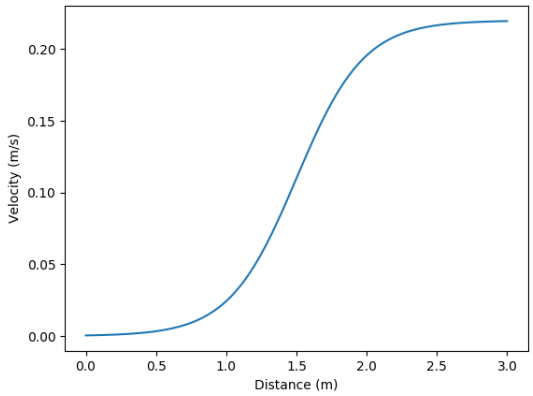


Fig. 5. Smooth transition between linguistic values

2. Turtlebot3 Robot and Robot Operating System

2.1. Robot Operating System

ROS is a platform for robot software development, it is helpful to test algorithms for robot tasks like mobile robot navigation. With Gazebo software we can simulate a robot and create virtual realistic stages, that can be based on real stages. The advantage of working in a simulated environment is that we can avoid damage to the robot due to improper behaviors when testing, in our case, controllers for the autonomous navigation of a mobile robot.

2.2. Turtlebot3 Mobile Robot

TurtleBot is consider the ROS standard platform robot and we used the TurtleBot3 robot burger version that is a differential drive mobile robot, its components are shown in Fig. 6. This has a maximum linear velocity of 0.22m/s and a maximum angular velocity of 2.84rad/s.

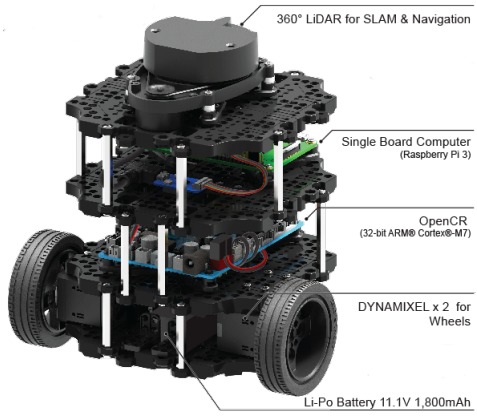


Fig. 6. TurtleBot3 Burger components

The key sensor that we used is the LIDAR sensor with which we can acquire 360 distance readings (d_i for $i = 0, \ldots, 359$) to the objects around it with a range of 0.12m to 3.5m. Based on the work of Boubertakh [3] we created 3 groups of readings to reduce the number of inputs for the robot controller designed in the next section. The first reading starts in front of the robot and taken counterclockwise. The left group *SL* consists of d_i for $i = 43, \ldots, 48$, d_i for $i = 368, 359, 0, 1, 2$ for the front group *SF* and d_i for $i = 313, \ldots, 318$ for the right group *SR* as shown in Fig. 7.

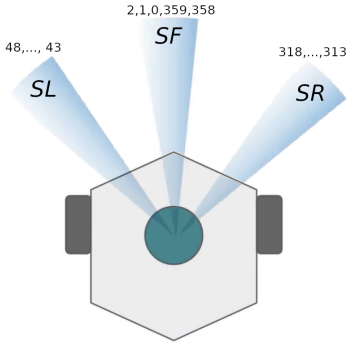


Fig. 7. Groups of sensor readings

The distances measured by the the three groups *SL*, *SF* and *SR* denoted by *dL*, *dF* and *dR* respectively are expressed as follows:

$$\begin{cases} dL = \text{mean}(d_{i=43,\ldots,48}) \\ dF = \text{mean}(d_{i=2,1,0,359,358}) \\ dR = \text{mean}(d_{i=313,\ldots,318}) \end{cases} \tag{2}$$

3. Design of the Fuzzy Logic Controller

There are many software tools for working with fuzzy logic. The MATLAB Fuzzy Logic Toolbox is one of the most used but this is a proprietary software so it is necessary to buy a license in order to use it. The disadvantage of sharing codes developed with proprietary software is that not everyone can replicate your work, it is necessary that the other person has a valid license of the software used and as we want for anyone to replicate those made in this paper we opted to use free software tools. The Python programming language is a interpreted language similar to MATLAB. Scikit-fuzzy is a fuzzy logic toolkit written on Python but it was not used in the experiments because it does not implement the creation of Sugeno-type fuzzy inference systems, for this reason a Python library called fuzzylab [1] based on the source code of the Octave Fuzzy Logic Toolkit [9] was developed.

3.1. Creating the FIS of the FLC

In this section we will explain step by step the creation of a fuzzy controller using the fuzzylab library for the TurtleBot3 robot. The Goal of the controller is for the robot to navigate in a stage without hitting the walls. First a `sugfis` object is defined with the fuzzylab library:

```
>>> fis = sugfis()
```

The controller has the task to determine the angular velocity depending of *dL*, *dF* and *dR* distances obtained by Eq. 2 that are the antecedent variables and the inputs of the FIS. To say that an object is *near* or *far* to the robot we need to define the range that the object's distance can be expected to vary. The minimum LIDAR sensor range is 0.12m and has a considered distance reading error of 0.01m. The input range setting for the

LIDAR sensor is from 0.13 to the maximum range that is 3.5. With this information we can add an input to `fis`:

```
>>> minr = 0.13
>>> maxr = 3.5
>>> fis.addInput([minr, maxr],Name='dF')
```

Now we define the membership functions for the fuzzy variable *dF*. Based on the work of Boubertakh [3] we use trapezoidal membership functions and define d_m, the minimum permitted distance to an obstacle and d_s, the safety distance beyond with which the robot can move at high speed. Those values were defined by experimentation, choosing those that generated an appropriate movement according to our own consideration.

```
>>> dm = 0.3
>>> ds = 0.7
>>> fis.addMF('dF','trapmf',
...      [minr, minr, dm, ds],Name='N')
>>> fis.addMF('dF','trapmf',
...      [dm, ds, maxr, maxr],Name='F')
```

The MF with the name `N` is the MF for *near* linguistic value and the MF with the name `F` is the MF for *far* linguistic value. Fig. 8 shows the plot of the membership functions of *dF* using the `plotmf` function:

```
>>> plotmf(fis,'input',0)
```

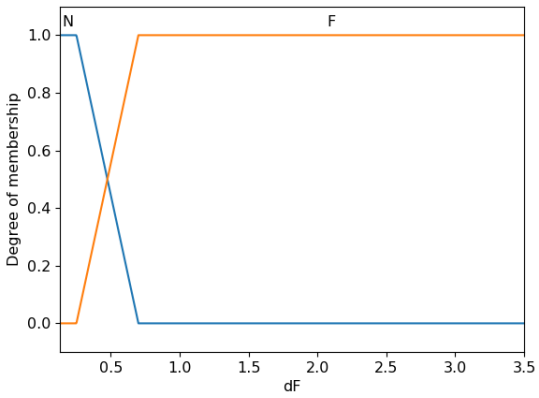


Fig. 8. Membership functions of *dF* variable

In the same way, we add *dL*, *dR* variables and the membership functions with the same parameters:

```
>>> fis.addInput([minr, maxr],Name='dL')
>>> fis.addMF('dL','trapmf',
...      [minr, minr, dm, ds],Name='N')
>>> fis.addMF('dL','trapmf',
...      [dm, ds, maxr, maxr],Name='F')
>>> fis.addInput([minr, maxr],Name='dR')
>>> fis.addMF('dR','trapmf',
...      [minr, minr, dm, ds],Name='N')
>>> fis.addMF('dR','trapmf',
...      [dm, ds, maxr, maxr],Name='F')
```

It is necessary to set some parameters for the robot and others for the controller for the navigation task. For the robot, we fixed the linear velocity ($0 < v \leq 0.22$) with the value of 0.15 (m/s) and we define the angular velocity range ($-2.84 \leq \omega \leq 2.84$) at which the robot can rotate with the value of ± 1.5 (rad/s). Those values were chosen based on the TurtleBot3 Machine Learning tutorial. We add the angular velocity consequent variable to `fis` with values *NB* (*Negative Big*), *ZR* (*Zero*) and *PB* (*Positive Big*):

```
>>> lin_vel = 0.15
>>> min_ang_vel = -1.5
>>> max_ang_vel = 1.5
>>>
>>> fis.addOutput([min_ang_vel,
...      max_ang_vel],Name='ang_vel')
>>> fis.addMF('ang_vel','constant',
...      min_ang_vel,Name='NB')
>>> fis.addMF('ang_vel','constant',
...      0,Name='ZR')
>>> fis.addMF('ang_vel','constant',
...      max_ang_vel,Name='PB')
```

Suppose a simple FIS where the task is to avoid only objects in front, the robot needs to rotate to left (*PB*) or right (*NB*). If we define two simple rules saying that "If *dF* is *N* then *ang_vel* is *PB*" and "If *dF* is *F* then *ang_vel* is *ZR*", the FIS will have the behavior to increase the angular velocity as the distance decreases as shown in Fig. 9, depending on the d_m and d_s values.

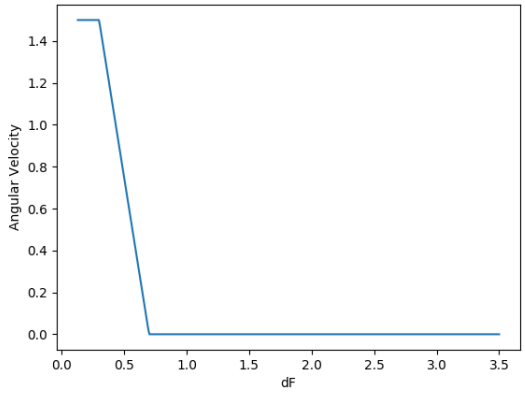


Fig. 9. The angular velocity increase as *dF* decreases

In this case there is a linear relation between *dF* and the angular velocity as in the case of Fig. 5, but the complexity of the FIS behavior increases when there are more than one antecedent variable because the angular velocity is determined depending on the value of *dL*, *dF*, *dR* and the rules defined in the FIS as shown in Fig 10.

The definition of the FIS rules require the use of expert knowledge, with these, we say how the angular velocity we want to be determined considering the different values that can be the antecedent variables.

There are 8 perceptual situations that the robot can have with three input groups and two linguistic values as shown in Fig. 11 and is associate a reaction to each of these situations defined by 8 simple rules:

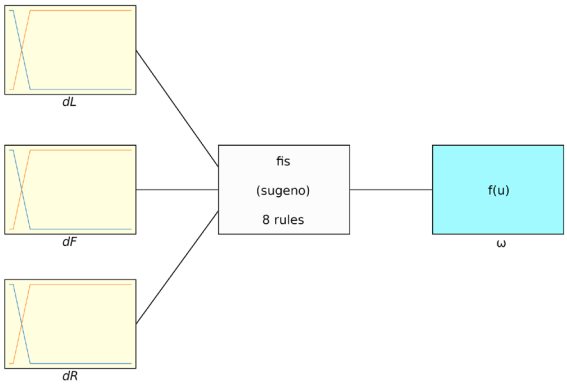


Fig. 10. Controller fuzzy inference system

$$
\begin{aligned}
&R1: IF(dL, dF, dR) \text{ is } (N, N, N) \text{ THEN } \omega \text{ is } NB\\
&R2: IF(dL, dF, dR) \text{ is } (N, N, F) \text{ THEN } \omega \text{ is } NB\\
&R3: IF(dL, dF, dR) \text{ is } (N, F, N) \text{ THEN } \omega \text{ is } ZR\\
&R4: IF(dL, dF, dR) \text{ is } (N, F, F) \text{ THEN } \omega \text{ is } NB\\
&R5: IF(dL, dF, dR) \text{ is } (F, N, N) \text{ THEN } \omega \text{ is } PB\\
&R6: IF(dL, dF, dR) \text{ is } (F, N, F) \text{ THEN } \omega \text{ is } NB\\
&R7: IF(dL, dF, dR) \text{ is } (F, F, N) \text{ THEN } \omega \text{ is } PB\\
&R8: IF(dL, dF, dR) \text{ is } (F, F, F) \text{ THEN } \omega \text{ is } ZR
\end{aligned}
$$

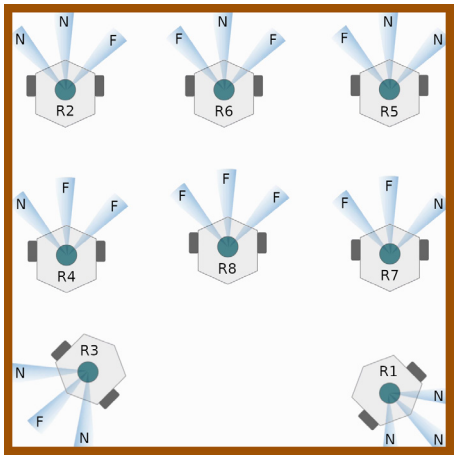


Fig. 11. Different perceptual situations with three inputs and two linguistic values

```
>>> ruleList = [
...     [0, 0, 0, 0, 1, 1], # Rule 1
...     [0, 0, 1, 0, 1, 1], # Rule 2
...     [0, 1, 0, 1, 1, 1], # Rule 3
...     [0, 1, 1, 0, 1, 1], # Rule 4
...     [1, 0, 0, 2, 1, 1], # Rule 5
...     [1, 0, 1, 0, 1, 1], # Rule 6
...     [1, 1, 0, 2, 1, 1], # Rule 7
...     [1, 1, 1, 1, 1, 1]] # Rule 8
```

The first columns specify input membership function indices, the following specify output membership function indices, the penultimate the rule weight and the last the antecedent fuzzy operator, where 1 specifies the "and" operator. The rules are added to `fis` with the `addRule` method:

```
fis.addRule(ruleList)
```

With all the previous steps a FIS is created with the fuzzylab library, we only need to determine *dL*, *dF* and *dR* from the sensor readings and evaluate this values in the fis with the `evalfis` function. This will be explained in the next section.

3.2. The Fuzzy Logic Controller

A controller has the task to determine the actions to take based on the perception of some sensor to resolve some problems. In our case, the controller has the task to calculate the angular velocity at which the robot needs to go based on the readings of the LIDAR distance sensor in order to not hit an object. This task is carried out mostly by the FIS created in the previous section, but part of the controller tasks is to process the sensor readings and to reflect their actions in the system, for this, ROS offers an easy API to realize those actions. To get the readings of the sensor and define *dL*, *dF* and *dR* values we use:

```
>>> dists = rospy.wait_for_message(
...     'scan', LaserScan)
>>>
>>> dL = mean(dists[43:48])
>>> dF = mean(dists[:3] + dists[-2:])
>>> dR = mean(dists[313:318])
```

Once the distances are determined, the new angular velocity is calculated from the FIS and reflects the result by publishing the value using the ROS functions:

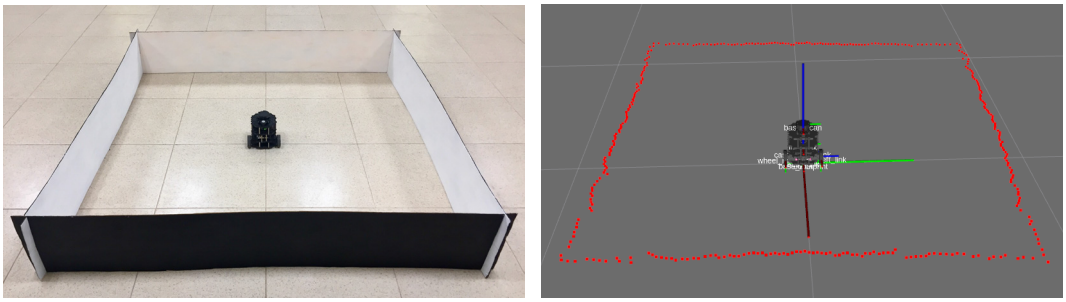
```
>>> new_ang_vel = evalfis(fis, [dL,dF,dR])
>>>
>>> twist = Twist()
>>> twist.linear.x = lin_vel
>>> twist.angular.z = new_ang_vel
>>>
>>> cmd_pub = rospy.Publisher(
...     'cmd_vel', Twist, queue_size=1)
>>> cmd_pub.publish(twist)
```

It is necessary that the robot is initialized to receive the updates. More detailed documentation can be found in the paper tutorial [1] inside the fuzzylab repository, this contains information about the necessary configurations needed to test the controller in the robot, simulated and physical.

4. Experiments and Results

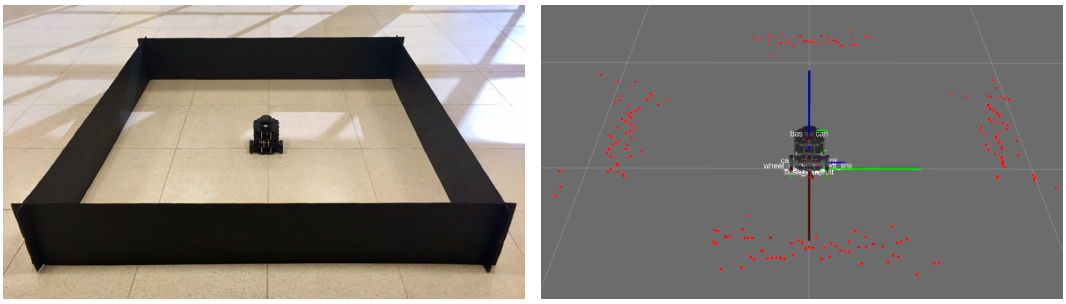
A real stage based on the stage 1 of the TurtleBot3 ML tutorial [2] was created, this is a 4x4 map with no obstacles as shown in Fig. 12(a). In first instance the stage was made with black walls as shown in Fig. 12(c) but when bad readings were observed as seen in Fig. 12(d) they were painted white having better readings as shown in Fig. 12(b).

The controller worked correctly in simulated and real environments, causing the robot to move on stage without hitting the walls, both in the simulated environment shown in Fig. 13(a) and in the real environment shown in Fig 13(b). Different behaviors can be observed manipulating d_m and d_s values. A lower d_m value makes the robot has a closer approximation to the objects.



(a) Stage with white walls

(b) Data acquisiton from 12(a)



(c) Stage with black walls

(d) Data acquisiton from 12(c)

Fig. 12. Physical stage. Data acquisition is better in Fig. 12(a) than Fig. 12(c)

(a) Behavior of the robot in the simulated stage.

(b) Behavior of the robot in the real stage.

Fig. 13. Behavior of the robot in the simulated and real stage

5. Conclusion

The FLC created with the fuzzylab library works correctly in the obstacle avoidance task with the TurtleBot3 robot. In future work more complex controllers can be designed to work in more complex stages, implementing optimization algorithms and evaluating the efficiency in some tasks. For the simplicity of the stage created, the manipulation of the linear velocity was not considered but in more complex stages the determination of the linear velocity could be considered.

Reinforcement learning (RL), an area of machine learning, is a computational approach to understanding and automating goal-directed learning and decision making [15]. Many of the RL algorithms such as Q-Learning [16] have been used to optimize fuzzy logic controllers, starting with the adaptation of the Q-learning algorithm for fuzzy inference systems by Glorennec [8] and Berenji [2] and more recent works [3–5] for control robot navigation. For these reasons, its use in future works with use of the fuzzylab library and the ROS platform has been contemplated.

ACKNOWLEDGEMENTS

We would like to express our gratitude to the CONACYT and Tijuana Institute of Technology for the facilities and resources granted for the development of this research.

Notes

[1]Paper tutorial for the environment configuration https://github.com/ITTcs/fuzzylab/tree/master/tutorials/fuzzylab_paper

[2]Reference tutorial http://emanual.robotis.com/docs/en/platform/turtlebot3/machine_learning/

AUTHORS

Eduardo Avelar* – Tijuana Institute of Technology, Tijuana, Mexico, e-mail: eduardo.avelar17@tectijuana.edu.mx.

Oscar Castillo – Tijuana Institute of Technology, Tijuana, Mexico, e-mail: ocastillo@tectijuana.mx.

José Soria – Tijuana Institute of Technology, Tijuana, Mexico.

*Corresponding author

REFERENCES

[1] E. Avelar. "fuzzylab". https://github.com/ITTcs/fuzzylab, 2019, Accessed on: 2020-05-28.

[2] H. R. Berenji, "Fuzzy q-learning: a new approach for fuzzy dynamic programming", vol. 1, 1994, 486–491, 10.1109/FUZZY.1994.343737.

[3] H. Boubertakh, M. Tadjine, and P.-Y. Glorennec, "A new mobile robot navigation method using fuzzy logic and a modified q-learning algorithm", *Journal of Intelligent & Fuzzy Systems*, vol. 21, no. 1 and 2, 2010, 113–119, 10.3233/IFS-2010-0440.

[4] L. Cherroun and M. Boumehraz, "Intelligent systems based on reinforcement learning and fuzzy logic approaches, "application to mobile robotic"", 2012, 1–6, 10.1109/ICITeS.2012.6216661.

[5] L. Cherroun, M. Boumehraz, and A. Kouzou, "Mobile robot path planning based on optimized fuzzy logic controllers", 2019, 255–283, 10.1007/978-981-13-2212-9_12.

[6] Y. Duan and Xin-Hexu, "Fuzzy reinforcement learning and its application in robot navigation", vol. 2, 2005, 899–904, 10.1109/ICMLC.2005.1527071.

[7] M. Faisal, R. Hedjar, M. A. Sulaiman, and K. Al-Mutib, "Fuzzy logic navigation and obstacle avoidance by a mobile robot in an unknown dynamic environment", *International Journal of Advanced Robotic Systems*, vol. 10, no. 1, 2013, 37, 10.5772/54427.

[8] P. Y. Glorennec and L. Jouffe, "Fuzzy q-learning". In: *Proceedings of 6th International Fuzzy Systems Conference*, vol. 2, 1997, 659–662 vol.2, 10.1109/FUZZY.1997.622790.

[9] L. Markowsky and B. Segee, "The octave fuzzy logic toolkit". In: *2011 IEEE International Workshop on Open-source Software for Scientific Computation*, 2011, 118–125, 10.1109/OSSC.2011.6184706.

[10] J. M. Mendel, *Uncertain Rule-Based Fuzzy Systems: Introduction and New Directions*, Springer, 2017.

[11] S. M. Raguraman, D. Tamilselvi, and N. Shivakumar, "Mobile robot navigation using fuzzy logic controller". In: *2009 International Conference on Control, Automation, Communication and Energy Conservation*, 2009, 1–5.

[12] P. Reignier, "Fuzzy logic techniques for mobile robot obstacle avoidance", *Robotics and Autonomous Systems*, vol. 12, no. 3, 1994, 143 – 153, 10.1016/0921-8890(94)90021-3.

[13] A. Saffiotti, "The uses of fuzzy logic in autonomous robot navigation", *Soft Computing*, vol. 1, no. 4, 1997, 180–197, 10.1007/s005000050020.

[14] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*, The Mit Press, 2011.

[15] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, The Mit Press, 2018.

[16] C. J. C. H. Watkins and P. Dayan, "Q-learning", *Machine Learning*, vol. 8, no. 3, 1992, 279–292, 10.1007/BF00992698.

[17] L. A. Zadeh, "Fuzzy sets", *Information and Control*, vol. 8, no. 3, 1965, 338–353, 10.1016/S0019-9958(65)90241-X.

[18] L. A. Zadeh, "Fuzzy logic = computing with words", *IEEE Transactions on Fuzzy Systems*, vol. 4, no. 2, 1996, 103–111, 10.1109/91.493904.