Research Article

DEVELOPMENT OF FUZZY LOGIC AND TKINTER SUPPORTED SEMI-AUTONOMOUS MOBILE SEARCH AND RESCUE ROBOT SOFTWARE

Gamze Nur DAŞDEMİR†, Serhat ÖZEKES††

† Uskudar University, Institute of Science and Technology, Artificial Intelligence Engineering Master's Program 34662 - Istanbul, Turkey

†† Marmara University, Faculty of Technology, Department of Computer Engineering 34854 - Istanbul, Turkey

gamzenurmat@gmail.com, serhat.ozekes@marmara.edu.tr

0009-0005-6660-0900, 0000-0002-7432-0272

Atıf/Citation: Daşdemir G. N., Özekes S., Development Of Fuzzy Logic And Tkinter Supported Semi-Autonomous Mobile Search And Rescue Robot Software, İstanbul Ticaret Üniversitesi Teknoloji ve Uygulamalı Bilimler Dergisi (8) no1, s 103-116 DOI: 10.56809/icujtas.1543400

ABSTRACT

The speed of technological changes in our rapidly developing world is an undeniable fact. With the contributions of science fiction novels and movies, expectations from the robotic universe are increasing at the same rate. Technological products are developed to meet the needs of humanity, or technological advancements lead to new needs. This study, prepared with the concern of how to be more beneficial to humanity, aims to help reduce earthquake disaster damages. The ROSbot 2.0 robot based on the Robot Operating System of the Husarion company was reprogrammed with a new low-cost approach to contribute to coarse search operations in search and rescue efforts using the Gazebo simulator. In the study, the Python programming language was used. A window reflecting the camera data was created with the Tkinter module, and a canvas was developed on this window where objects' lengths could be measured by drawing with the mouse. The suitability of the gap measurements in coarse search operations was calculated using fuzzy logic. Additionally, with the fuzzy logic-supported autonomous driving code, obstacle avoidance and progress towards the target tasks were successfully performed. During autonomous driving, speed and rotation angle values were obtained with the help of fuzzy logic according to laser sensor data to enable the robot to quickly adapt to dynamic conditions. Thus, it was seen that professional search and rescue teams could avoid harm in new collapses caused by fires, aftershocks, etc., that might occur in the wreckage areas, rescuing the victims faster and helping this robot sent on a reconnaissance mission to assist in dividing the wreckage area into sectors.

Keywords: Fuzzy Logic, Gazebo, Python, Robot Operating System, Tkinter

BULANIK MANTIK VE TKINTER DESTEKLİ YARI OTONOM MOBİL ARAMA KURTARMA ROBOT YAZILIMININ GELİŞTİRİLMESİ

ÖZET

Her gün gelişen dünyamızdaki teknolojik değişimlerin hızı yadsınamaz bir gerçek. Bilim kurgu romanlarının ve filmlerinin de katkılarıyla robotik evrenden beklentiler aynı hızda artmaktadır. İnsanlığın ihtiyaçlarına cevap bulmak adına teknolojik ürünler geliştirilmekte veya teknolojik gelişmeler yeni ihtiyaçlara neden olmaktadır. İnsanoğluna daha nasıl faydalı olunur kaygısıyla hazırlanan bu çalışmada bu çalışmada, deprem afeti hasarlarının azaltılmasına yardımcı olunması amaçlandı. Husarion firmasının Robot İşletim Sistemi tabanlı ROSbot 2.0 robotu, Gazebo simülatörü sayesinde arama kurtarma çalışmalarında yapılan kaba aramaya katkı sağlaması amacıyla az maliyetli yeni bir yaklaşımla yeniden programlandı. Çalışmada Python programlama dili kullanıldı. Tkinter modülü ile kamera verilerini yansıtan bir pencere oluşturuldu ve bu pencere üzerinde fare ile çizim yapılarak nesnelerin uzunluklarının ölçülebileceği bir tuval geliştirildi. Kaba arama çalışmalarında boşlukların ölçü uygunluğu, bulanık mantık ile hesaplandı. Ayrıca, bulanık mantık destekli otonom sürüş kodu sayesinde engelden kaçma ve hedefe ilerleme görevleri başarıyla gerçekleştirildi. Otonom sürüş esnasında robotun dinamik koşullara hızlı uyum sağlayabilmesi için hız ve dönme açısı değerleri lazer sensör verilerine göre bulanık mantık yardımıyla elde edildi. Bu sayede profesyonel arama kurtarma ekiplerinin enkaz alanlarında oluşabilecek yangın, artçı sarsıntı

Geliş/Received	**:**	**04.09.2024.**
Gözden Geçirme/Revised	**:**	**10.01.2025**
Kabul/Accepted	**:**	**13.01.2025**

gibi sebeplerle oluşabilecek yeni yıkımlarda zarar görmemesi, kazazedelerin daha hızlı kurtarılması ve keşif görevine gönderilen bu robotun enkaz alanının sektörlere ayırma çalışmalarına yardımcı olabileceği görüldü.

Anahtar Kelimeler: Bulanık Mantık, Gazebo, Python, Robot İşletim Sistemi, Tkinter

1. INTRODUCTION

In today's world, expectations from the robotic universe are increasing with the increase in tools that visualize the imagination. One of the biggest reasons for this situation can be shown as science fiction. American science fiction writer Isaac Asimov, the inventor of the word "robotics", is the best evidence of this. In his book "I, Robot", first published in 1942, the "three laws of robotics" are indelibly engraved everywhere (Vikipedi, 2021). These and similar ideas designed in fictional universes increase the desire for humanoid robots. However, while dreaming of humanoid robots to make our lives easier, the contributions of semi-autonomous or autonomous mobile robots such as delivery and transportation robots (Bogue, 2016), underwater robots (Canlı, 2015), wheeled robots (Levine et al., 1999), cleaning robots (Usa et al., n.d.), soccer-playing robots (Weigel, 2005), space exploration robots (Vikipedi, 2024), which can perform human tasks faster and increase their place in human life every day, should not be ignored. In order to increase this contribution, in this study, Husarion ROSbot 2.0, a commercial robot of Husarion company, was used to add a new one to semi-autonomous mobile robots.

The first 72 hours after an earthquake, which is one of the natural disasters affecting our country, is of great importance (Kundak & Kadıoğlu, 2011). Some situations that both volunteer and professional teams should consider are gas leaks, fire, aftershocks, and new destruction that may occur following aftershocks (Dollarhide & Agah, 2003). Studies that propose shape-shifting marsupial robots to minimize damage in situations that could harm trapped people or aid teams (Murphy, 2000), advocating wireless communication for short distances in search and rescue robots and the use of collapsible wheels for difficult terrain conditions (İşeri, 2017) and autonomous human detection with machine learning methods advocating a 6-legged design (Ajith, 2024) are available.

In this study, a semi-autonomous mobile robot that can be both manually controlled from the keyboard and used autonomously was programmed to help the teams to save their lives and rescue the victims faster due to unpredictable real-world conditions. The robot's camera is always operational so that the environment can be observed by the user. It is aimed to guide the search and rescue teams by sending the robot to a possible building wreckage for exploration purposes, to help the teams to determine the entry points to the wreckage or the areas where they can move in the wreckage, to obtain faster and more human-like results by using fuzzy logic method, to find the survivors faster if there are any in the wreckage, and to help to carry out the work of dividing the building wreckage into sectors faster.

2. MATERIAL AND METHOD

Within the scope of the study, the ROSbot 2.0 robot of the Husarion company was developed in the ROS (Robot Operating System) environment. The usability of the robot was tested in the Gazebo simulator. It was coded using the Python programming language. By combining the camera image data with the blank canvas window created via the Tkinter module in the same popup window, an image canvas that can be drawn on was obtained. In this way, the width, height, and depth measurements of objects or gaps through which the robot is supposed to pass could be measured in real-time. The gap measurements, inquiring whether the robot can pass, were sent to the fuzzy logic code that provides transition suitability based on the robot's dimensions. Thus, a faster solution was developed against the difficulty of keeping each user's measurements in mind or integrating the measurements of robots designed in different dimensions into the system. Additionally, the robot was also enabled to perform obstacle avoidance and navigation tasks in autonomous driving mode. During the task, speed and rotation angle values were obtained using the fuzzy logic method, similar to transition suitability, due to their effectiveness in uncertain and complex environments and the inability to define sharp boundaries. In areas deemed necessary, the robot was enabled to perform autonomous mapping. In general, the operating principle of the robot is shown in the diagram below.

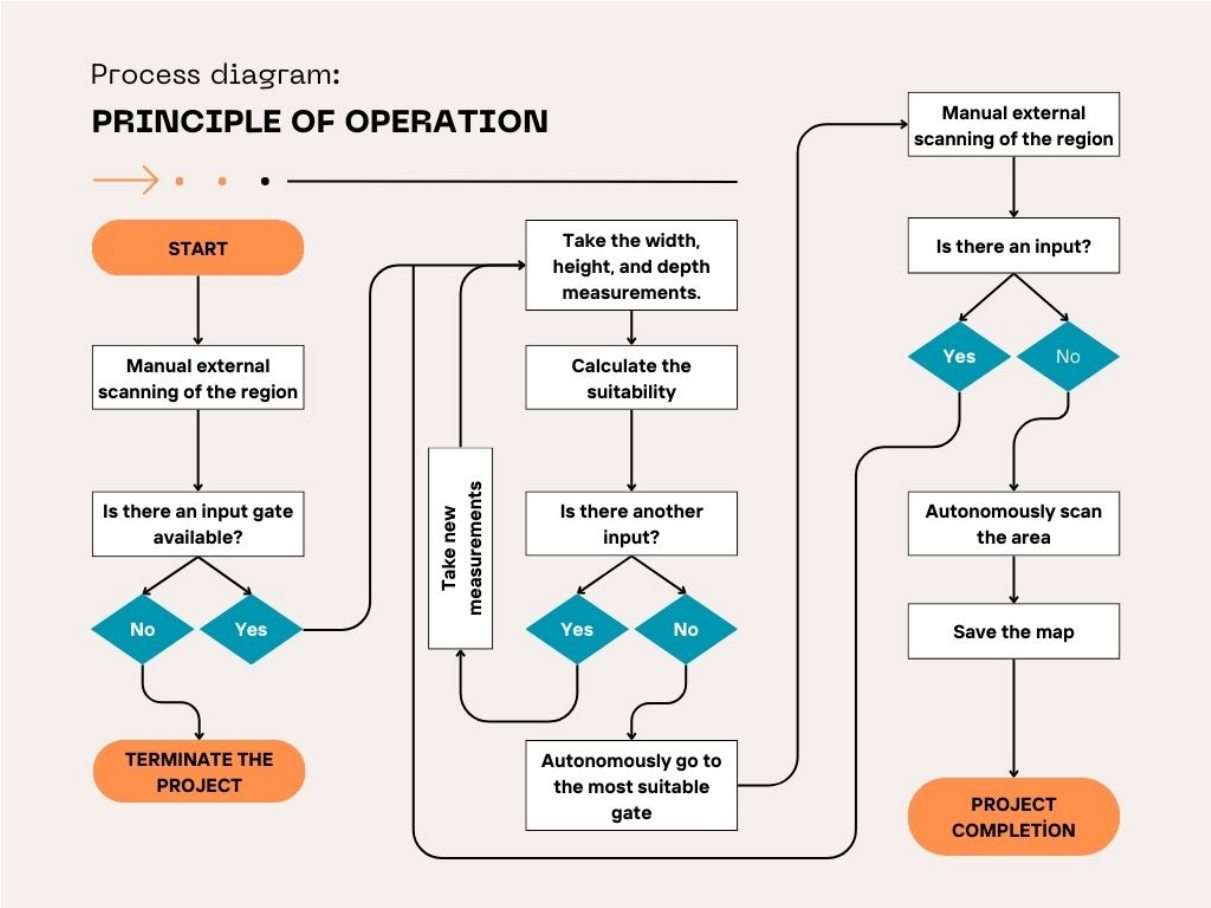


Figure 1: Flow chart

2.1. Robot Operating System

The Robot Operating System (ROS) is a flexible software development kit that provides the various tools and libraries needed to create robotic software (Tekerek et al., 2022). It offers many powerful features to help developers with tasks such as message routing, compute distribution, code reuse, and implementation of cutting-edge algorithms for robotic applications (Joseph & Cacace, 2018). With the included Gazebo software, you can simulate a robot and create virtual realistic stages based on real stages (Avelar et al., 2020).

The environment, built from scratch using the Gazebo simulator, was designed as a walled maze with multiple entrances from the outside to test the functionality of the robot's software. Since the focus was on the robot's software instead of its skills, instead of simulating harsh terrain conditions, the focus was on entrances where it could calculate length measurements to provide a basis for the detection of entrances that a human could pass through in the real world.

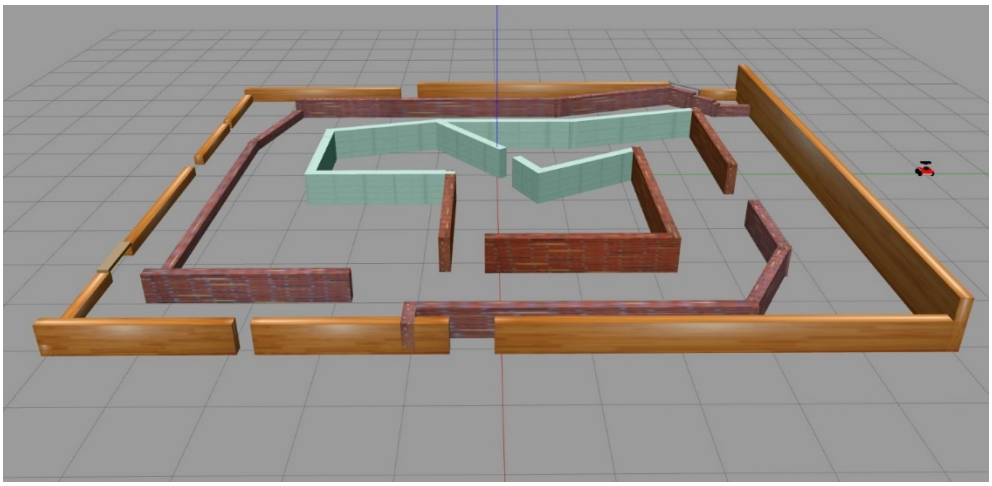


Figure 2. Simulation environment

2.2. ROSbot 2.0 (ROSbot 2.0)

Figure 3. ROSbot 2.0 (ROSbot 2.0) (Husarion, 2024)

The ROSbot is a ROS-supported 4x4 drive autonomous mobile robot platform equipped with LIDAR, RGB-D camera, IMU, encoders, and distance sensors, available in three versions: "2R," "2 PRO," and "2." The LIDAR sensor of the ROSbot 2.0 robot used in this study can perform 720 readings at 360 degrees up to 12 meters. The sensor data was divided into 5 different regions, but to reduce the number of inputs, distance readings from the *front (front), front left (fleft)*, and *front right (fright)* regions were considered. In this way, a 108-degree field of view was set. Although the LIDAR sensor has a range of 12 meters, readings were limited to a maximum of 10 meters, and movements were determined based on the minimum values of the distances from sensor data. The reading regions made counterclockwise were expressed in the following Python code block:

```
    regions_ = {
        'right': min(min(msg.ranges[540:612]), 10),
        'fright': min(min(msg.ranges[612:684]), 10),
        'front': min(min(msg.ranges[684:720]), min(msg.ranges[0:36]), 10),
        'fleft': min(min(msg.ranges[36:108]), 10),
        'left': min(min(msg.ranges[108:180]), 10),
    }
```

Figure 4: Lidar sensor reading regions

2.3. Fuzzy Logic

The concept of fuzzy logic is described by its developer Prof. Dr. Lotfi Asker Zadeh as "a system of reasoning and computation in which the objects of reasoning and computation are classes with blunt boundaries." (Zadeh, 2015) is defined as.

In the universal set E, given $x \in E \; ve \; A \subseteq E,$ the set A, which can have infinitely many membership degrees within the range [0,1], is called a "fuzzy set" and is denoted by Ã. The membership function for Ã is expressed as $\mu_{\tilde{A}}(x)$ (Zadeh, 1965).

$$\tilde{A} = \{(x, \mu_{\tilde{A}(x)}) | x \epsilon E\}$$

Fuzzy logic, which is not based on sharp conclusions like classical logic, can better represent the real world, which is full of uncertainties. Applications that work with fuzzy logic approach to enable machines to exhibit human-like behaviors will narrow the distance between the robotic universe and human beings.

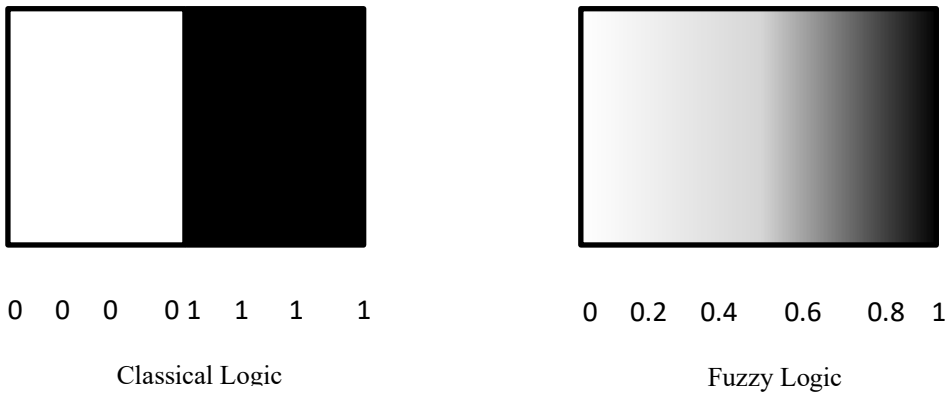
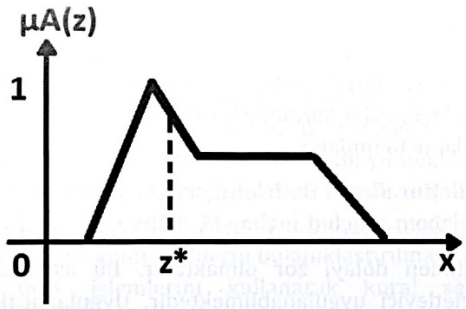


Figure 5: Schematic representation of classical logic and fuzzy logic (Yılmaz & Şahin,2023)

Three different fuzzy logic codes were written for the developed robot using the scikit-fuzzy library of the Python programming language. In this library, the Mamdani method is used as the fuzzy inference method. In this method, which requires expert knowledge and can be applied to solve any problem in fuzzy logic, both input and output variables are expressed with membership functions in closed form. In the final step of the method, a defuzzification process is performed to convert the outputs into a crisp result. In this study, the "center of gravity method," also known as the "centroid" defuzzification method, was used. In the center of gravity method, a numerical result corresponding to the membership degree located at the centroid of the shape obtained from the inference is found, leading to a precise result.



$$z^* = \frac{\int \mu c(z).zdz}{\int \mu c(z)dz}$$

Figure 6: Graphical representation and mathematical equation of the Center of Gravity Method (Çetinkaya, 2023)

The first fuzzy logic code gives the percentage of probability that the robot can pass through the gates. In this code, which was imported into the main code, the length measurements of the robot were used for the width, length and depth variables and membership values were calculated according to the triangular membership function. Figure 7 shows the membership functions. While creating the membership functions for transition suitability, the

dimensions of the robot were taken into account. The lengths were adjusted by adding and subtracting a margin of error of four centimeters.

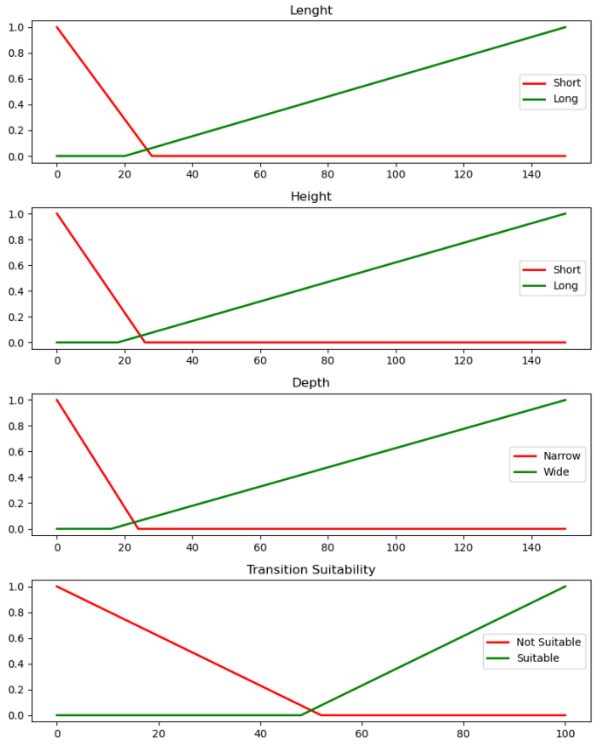


Figure 7. Transition eligibility membership functions

The other fuzzy logic codes prepared were imported into the Python code that enables autonomous driving. During the progression towards the target, separate speed and rotation angle values are obtained when encountering an obstacle. To enable smoother transitions, quicker obstacle avoidance, and more human-like responses, Z and S-shaped membership functions were preferred for sensor data, rotation angle values, and speed values. This can be seen in Figure 8 and Figure 9.

.

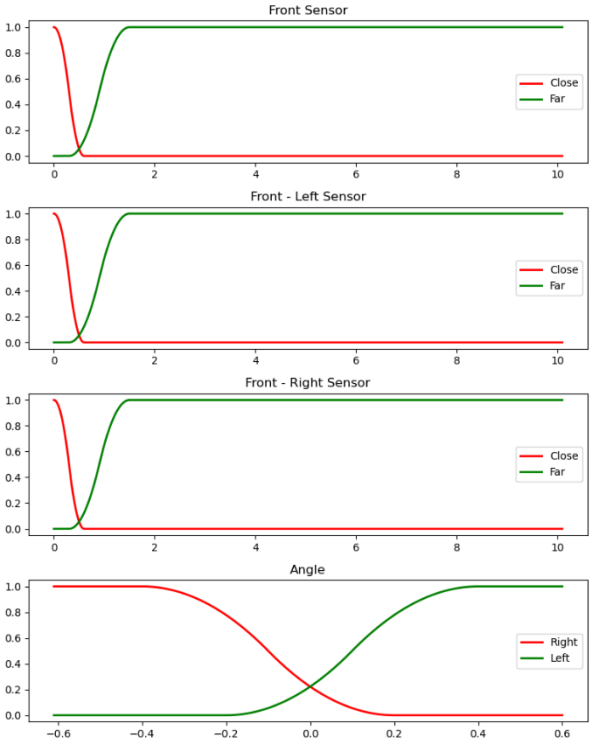


Figure 8. Rotation angle membership functions

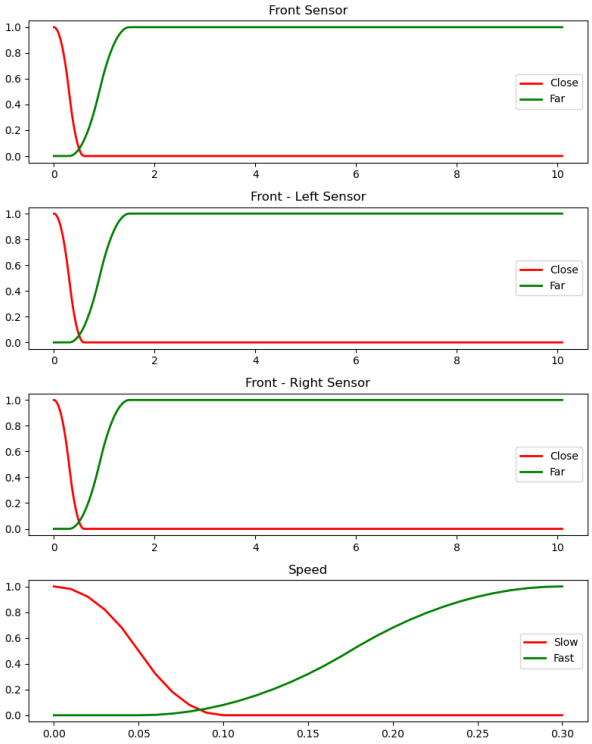


Figure 9. Speed value membership functions

2.4. Python codes

2.4.1. Main code

Thanks to the code created in the study, the reprogrammed robot has the functions of laser scanning, depth measurement, image processing and user interactive computation. Its main structure depends on the choices made by the user from the menu presented. The study provides detailed options against possible real-world problems and user errors. The depth camera was used to measure the width and height of the objects and the depth of the inputs, while the laser sensor data was used to measure the distance to the objects in autonomous driving mode. The images from the robot camera were processed in ROS and a GUI was created with Tkinter, a Python library. It was then updated in the Tkinter canvas and mouse click events were linked to the canvas. In this way, the distance between two points selected by the user on the camera images was calculated. For drawing, the mouse movement was highlighted with a red line. It is shown in Figure 6.

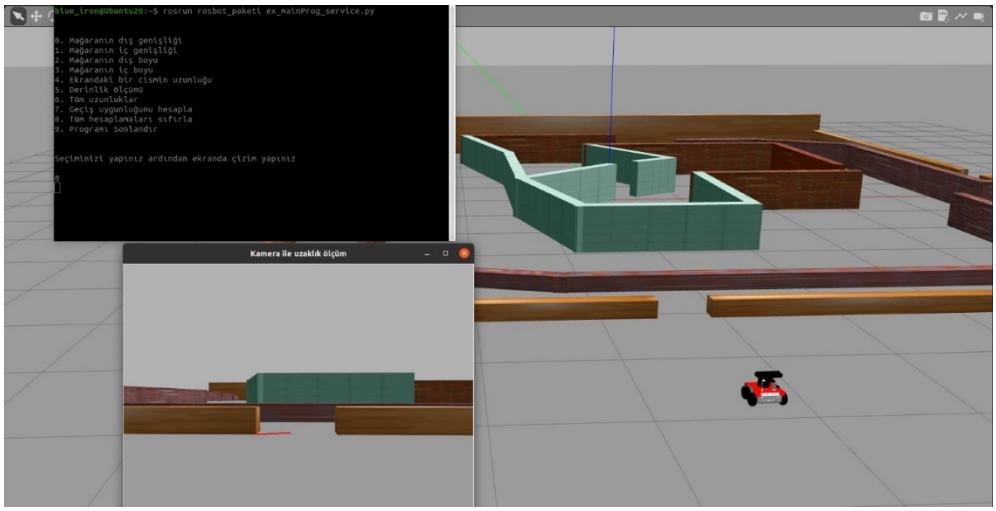


Figure 10. Menu options and drawing made on robot camera

The canvas created with the help of the Tkinter library was used to process the image data received from the robot camera in the same window, transforming it into an image canvas on which drawings can be made. While obtaining the distance data from objects to the robot, depth camera data was used instead of laser data, leading to more accurate measurements. As a result of the measurements, an approximate constant value corresponding to 100 cm was found. A simple ratio formula was used to convert the measured pixel values into length measurements. The lengths obtained from the measurements were sent to the imported fuzzy logic code. At the same time, the position of the robot was recorded and marked with a lettering system during the measurement. The imported fuzzy logic code calculates the probability of the robot passing through the gap without getting stuck, based on the robot's width, height, and depth dimensions. Figure 12 shows the probability percentage, position mark, and coordinates. A Python code named "fuzzy_perm" was created to calculate the passage permission. Parameters w (width), h (height), and d (depth) were defined for the function. The boundaries for each length were determined using the robot's actual dimensions, with a ±4 cm margin of error. The probability value was scaled by half. The maximum length in the measurement universe was set to 150 centimeters. The width, height, and depth measurement values of the passages were set as follows:

```python
def fuzzy_perm(w, h, d):
    universe = np.arange(0, 151, 1)
    perm_universe = np.arange(0, 101, 1)

    width_short = mf.trimf(universe, [0,0,28])
    width_long = mf.trimf(universe, [20,150,150])

    height_short = mf.trimf(universe, [0,0,26])
    height_long = mf.trimf(universe, [18,150,150])

    depth_tight = mf.trimf(universe, [0,0,24])
    depth_wide = mf.trimf(universe, [16,150,150])

    perm_neg = mf.trimf(perm_universe, [0,0,52])
    perm_pos = mf.trimf(perm_universe, [48,100,100])
```

Figure 11: The parameters of the fuzzy logic code named fuzzy_perm

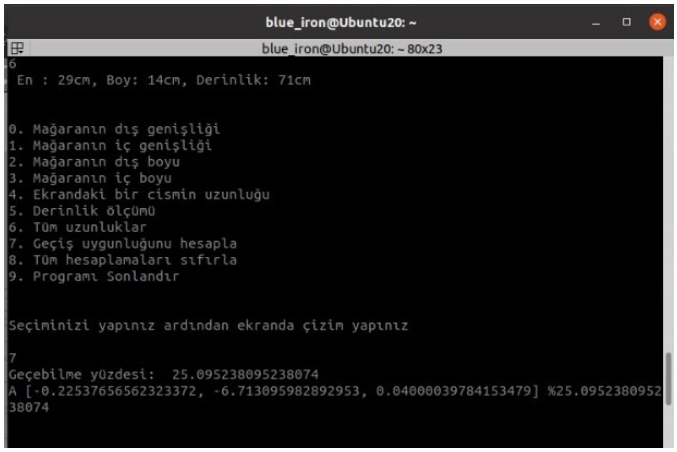


Figure 12. Point A, pass percentage, coordinate information

At the same time, the robot's odometric data was taken and served to a third Python code that acts as a client and server.

2.4.2. Autonomous driving code

This Python code was written to enable the robot to avoid obstacles it may encounter on its way to the target. The data of the 108-degree viewing angle on the front of the robot which has 360-degree scanning laser sensor, was used. This data was used for 8 different possible situations of the robot (Yayan & Erdoğmuş, 2021):

```python
    if d_min < regions['front'] and d_min < regions['fleft'] and d_min < regions['fright']:
        change_state(4)  # go to goal

    elif regions['front'] < d_min and d_min < regions['fleft'] and d_min < regions['fright']:
        change_state(2)  # turn right

    elif d_min < regions['front'] and d_min < regions['fleft'] and regions['fright'] < d_min:
        change_state(1)  # turn left

    elif d_min < regions['front'] and regions['fleft'] < d_min and d_min < regions['fright']:
        change_state(2)  # turn right

    elif regions['front'] < d_min and d_min < regions['fleft'] and regions['fright'] < d_min:
        change_state(1)  # turn left

    elif regions['front'] < d_min and regions['fleft'] < d_min and d_min < regions['fright']:
        change_state(2)  # turn right

    elif regions['front'] < d_min and regions['fleft'] < d_min and regions['fright'] < d_min:
        change_state(2)  # turn right

    elif d_min < regions['front'] and regions['fleft'] < d_min and regions['fright'] < d_min:
        change_state(3)  # follow the wall

    else:
        change_state(5)  # unknown case
```

Figure 13: 8 different decision states

If there is no obstacle within a distance of up to 0.5 meters in front of the robot, or if an obstacle is encountered at a distance closer than the minimum in the front left (fleft) and front right (fright) regions, the robot does not make any deviation while advancing towards the target point. In case of encountering any obstacle, the robot is made to turn in the opposite direction of the obstacle. For the turning movement, it obtains new speed and rotation angle values from two different imported fuzzy logic codes. For both values, the distance is set to a maximum of 10 meters, objects up to 0.6 meters are considered close, and objects between 0.3 and 1.5 meters are considered far. The rotation angle values are set in the range (-0.6, 0.6) (rad/s) as follows:

```
def fuzzy_angle(var_front, var_fleft, var_fright):
    dist_range = np.arange(0, 10.1, 0.01)
    angle_range = np.arange(-0.61, 0.61, 0.01)

    front_close = mf.zmf(dist_range, 0, 0.6)
    front_far = mf.smf(dist_range, 0.3, 1.5)

    fleft_close = mf.zmf(dist_range, 0, 0.6)
    fleft_far = mf.smf(dist_range, 0.3, 1.5)

    fright_close = mf.zmf(dist_range, 0, 0.6)
    fright_far = mf.smf(dist_range, 0.3, 1.5)

    angle_right = mf.zmf(angle_range, -0.4, 0.2)
    angle_left = mf.smf(angle_range, -0.2, 0.4)
```

Figure 14: Rotation angle parameters

The speed of the robot was set as follows: maximum 0.30 (m/s), (0,0.1)(m/s) range as slow, (0.05,0.3)(m/s) range as fast:

```
def fuzzy_speed(var_front, var_fleft, var_fright):
    dist_range = np.arange(0, 10.1, 0.01)
    speed_range = np.arange(0, 0.31, 0.01)

    front_close = mf.zmf(dist_range, 0, 0.6)
    front_far = mf.smf(dist_range, 0.3, 1.5)

    fleft_close = mf.zmf(dist_range, 0, 0.6)
    fleft_far = mf.smf(dist_range, 0.3, 1.5)

    fright_close = mf.zmf(dist_range, 0, 0.6)
    fright_far = mf.smf(dist_range, 0.3, 1.5)

    speed_low = mf.zmf(speed_range, 0, 0.1)
    speed_high = mf.smf(speed_range, 0.05, 0.3)

```

Figure 15: Speed parameters

Figure 16 shows the choices made by the robot when it encounters an obstacle, and the speed and rotation angle values from the fuzzy logic codes. Figure 17 shows the arrival at the target.

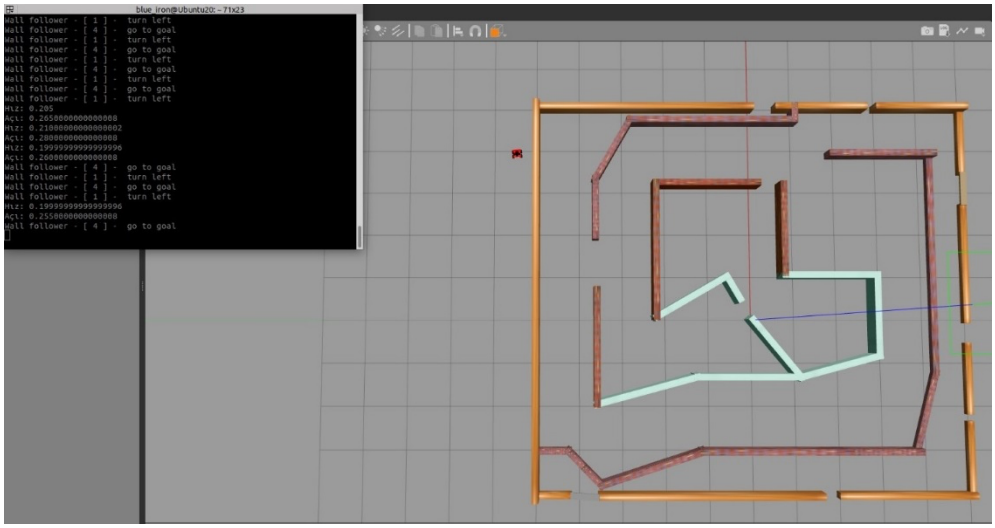


Figure 16. Progress to the target coordinate

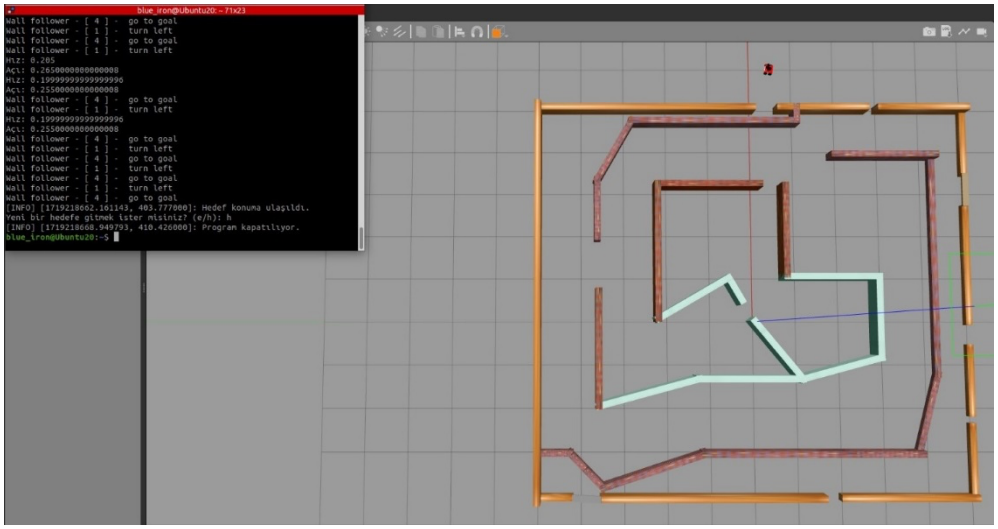


Figure 17. End of autonomous driving mission

3. FINDINGS AND RECOMMENDATIONS

It is very important to synchronize the repetition rate of the camera data with the repetition rate of the created canvas. Otherwise, problems such as delays in the image and incorrect coordinates in the drawings were observed. When the generated image canvas is clicked, the software inherently processes this information as pixel data. Since the purpose of the canvas was to take length measurements of objects by drawing on them, the pixel data had to be converted into length measurements. To solve this problem, objects with known length measurements were placed in front of the robot at different distances to answer the question "how many cm in length do the measured pixel values correspond to?".

In the user-controlled exploration to take the length measurements of the passage, the recording of the coordinates that the robot passed was also performed in the main code. The reason for taking the coordinate list is to ensure that the robot follows a backward route when it is sent back to the appropriate passage. Because when autonomous driving is selected for the return, the robot, preferring the shortest route to the goal, may enter the maze environment. It was concluded that this situation could be problematic in real-world applications. Since the first scan to the coordinates of the passage to be given as the target is user controlled and the paths that the user chooses are predicted to be unobstructed, it was seen that it would be smoother to return the way it came.

The list of coordinates received during the first scan had to be sent to the Python code that provides autonomous driving. We wanted to provide this in the main code, but in a classical Python coding, the import method for communication between codes does not cause problems, while the architecture of the ROS environment creates problems in the classical method. As a solution, ROS offers a so-called client-server. For this reason, the main code was also turned into a server that broadcasts the coordinate data. The coordinate list is sent to a third Python code that acts as a bridge to communicate with the autonomous driving code. The third Python code was designed to be both a client and a server. It acts as a client for the coordinates broadcast from the main code and as a server to send the coordinates to the autonomous driving code. The problem with this structure is that the robot has to re-detect its position at each coordinate point and calculate how far it is from the target. Because this caused the robot to reach the target very slowly. In order to solve the time problem, the coordinates in the list from the main code were re-listed in the same order but in a unique way. In the coordinate values, positive numbers after the comma were rounded to the next higher number and negative numbers after the comma were rounded to the next lower number. Finally, the elements in the new list were removed in fives to reduce the list and make it faster by setting more distant targets.

In the experiments, it was determined that the preference for more sensor data created too many situations and the robot had difficulty in the decision-making process. For this reason, the values read from the front of the robot were limited to 108 degrees.

Once the appropriate gate is detected and entered into the maze, the mapping code recommended by Husarion makes path planning according to the corners in the environment. Therefore, in the multi-door environment simulated in this study, it was seen that the robot had difficulty in deciding that the mapping was finished. However, it was found that it was still able to map unfamiliar environments quite successfully.

4. RESULTS AND DISCUSSION

Thanks to the fuzzy logic theory, very successful results were obtained in the measurements made with the Tkinter module without resorting to methods such as marking objects for measurement. It was observed that the robot, which was operated in the Gazebo simulator in the ROS environment, correctly performed the tasks of avoiding obstacles and going to the target coordinate in autonomous driving mode.

In navigation studies with uncertain inputs, it has been observed that fuzzy logic is advantageous for large, uncertain, and complex systems because it can model many conditions in a single system, thereby reducing code complexity, is closer to human thinking, and is shorter and more understandable compared to if-else code blocks, which require definite boundaries and become difficult to manage with many conditions.

When the studies and their results are evaluated, it is seen that there has been a considerable contribution to the robotics universe, as well as being aware of the shortcomings. It is argued that the use of fuzzy logic in the development of control mechanisms of autonomous mobile vehicles is more successful than classical robot control mechanisms in the results obtained in sensor data balancing errors and computational costs in path planning (Bayar et al., 2014). In another study conducted in the ROS environment using Python with fuzzy logic support, they created their own libraries to use the Sugeno type as the fuzzy inference method. This study shows that robots trained in the Gazebo simulation environment can perform the same way in the physical environment (Avelar et al., 2020). Thus, working only in the simulation environment initially will provide good preparation for the physical environment. In the research conducted by Yüksek, A. and colleagues, which compares studies on path planning for robots, it is mentioned that fuzzy logic studies are limited and that the use of fuzzy logic in the solution of real-time navigation problems in the conducted studies is beneficial (Yüksek et al., 2024). However, it was found that fuzzy logic applications coded in Python programming language in ROS environment are limited in the literature. In addition, there is no application using the Tkinter library, which is used to perform measurements manually with the traditional approach. Therefore, the low-cost methods used in this study are intended to contribute to future studies.

It should not be forgotten that anxious people build the world we live in. It is hoped that this study will be a horizon for new researchers with anxieties.

Acknowledgment

This publication is prepared from the master's thesis of Gamze Nur DAŞDEMİR, which was prepared under the supervision of Prof. Dr. Serhat ÖZEKES in the Artificial Intelligence Engineering Thesis Master's Program at Üsküdar University, Institute of Science.

Conflict of Interest Statement

No conflicts of interest were declared by the authors.

REFERENCES

Ajith, V. (2024). Search and Rescue Robot. *International Journal of Science, Engineering and Technology*, *12*(1). https://doi.org/10.61463/ijset.vol.12.issue1.112

Avelar, E., Castillo, O., & Soria, J. (2020). Fuzzy logic controller with fuzzylab python library and the robot operating system for autonomous mobile robot navigation. *Journal of Automation, Mobile Robotics and Intelligent Systems*, *14*(1), 48–54. https://doi.org/10.14313/JAMRIS/1-2020/6

Bayar, V., Akar, B., Yayan, U., Yavuz, H. S., & Yazici, A. (2014). Fuzzy logic based design of classical behaviors for mobile robots in ROS middleware. *INISTA 2014 - IEEE International Symposium on*

Innovations in Intelligent Systems and Applications, Proceedings, 162–169. https://doi.org/10.1109/INISTA.2014.6873613

Bogue, R. (2016). Growth in e-commerce boosts innovation in the warehouse robot market. *Industrial Robot*, *43*(6), 583–587. https://doi.org/10.1108/IR-07-2016-0194

Canlı, G. A. , K. İ. , C. M. O. , T. Ö. S. (2015). DÜNYADA VE ÜLKEMİZDE İNSANSIZ SUALTI ARAÇLARI (İSAA-AUV & ROV) TASARIM VE UYGULAMALARI. *Gidb Dergi*, *4*, 43–75.

Çetinkaya, A. (2023). *Bulanık Mantık ve Python Uygulamaları*. İGÜ .

Dollarhide, R. L., & Agah, A. (2003). Simulation and control of distributed robot search teams. *Computers and Electrical Engineering*, *29*(5), 625–642. https://doi.org/10.1016/S0045-7906(01)00048-9

Husarion. (2024). *Husarion*. Husarion Sp. https://husarion.com/tutorials/ros-tutorials/1-ros-introduction/

İşeri, M. C. (2017). *DESIGN AND IMPLEMENTATION OF A MOBILE SEARCH AND RESCUE ROBOT*.

Joseph, L., & Cacace, J. (2018). *Mastering ROS for robotics programming : design, build, and simulate complex robots using Robot Operating System and master its out-of-the-box functionalities* (2nd ed.). Packt Publishing LTD.

Kundak, S., & Kadıoğlu, M. (2011). *İLK 72 SAAT*. AFAD-T.C. BAŞBAKANLIK Afet ve Acil Durum Yönetimi Başkanlığı.

Levine, S. P., Bell, D. A., Jaros, L. A., Simpson, R. C., Koren, Y., Member, S., & Borenstein, J. (1999). The NavChair Assistive Wheelchair Navigation System. In *IEEE TRANSACTIONS ON REHABILITATION ENGINEERING* (Vol. 7, Issue 4).

Murphy, R. (2000). Marsupial and shape-shifting robots for urban search and rescue. *IEEE Intelligent Systems and Their Applications*, *15*(2). https://doi.org/10.1109/5254.850822

Tekerek, M., Gök, M., Akçam, Ö. Ş., & Aydemir, H. (2022). *Otonom Mobil Robotlar İçin ROS El Kitabı* (Vol. 1). Necmettin Erbakan University Publications.

Usa, V. C., Palacin, J., Salse, J. A., Valgaiion, I., & Clua, X. (n.d.). *IMTC 2W3-Instrumentation and Measurement Technology Conference Building a Mobile Robot for a Floor-Cleaning Operation in Domestic Environments*.

Vikipedi. (2021, August 22). *https://tr.wikipedia.org/wiki/%C3%9C%C3%A7_robot_yasas%C4%B1#Kaynak%C3%A7a*.

Vikipedi. (2024, April 22). *https://tr.wikipedia.org/wiki/Curiosity_(ke%C5%9Fif_arac%C4%B1)*.

Weigel, T. (2005). *KiRo – A Table Soccer Robot Ready for the Market*. IEEE.

Yayan, U., & Erdoğmuş, A. K. (2021). *ROS ile Robotik Uygulamalar* (A. Yazıcı, Ed.; 1st ed.).

Yılmaz, H., & Şahin, M. E. (2023). *Bulanık Mantık Kavramına Genel Bir Bakış*. https://dergipark.org.tr/tr/pub/takvim/issue/78360/1262874

Yüksek, A. G., Zontul, M., & Yüksek, E. (2024). A COMPREHENSIVE SURVEY OF PATH PLANNING ALGORITHMS FOR AUTONOMOUS VEHICLES AND MOBILE ROBOTS. *Research and Findings in Engineering Sciences-2024*, 1–66.

Zadeh, L. A. (1965). Fuzzy Sets.pdf. In *Information and Control* (Vol. 8).

Zadeh, L. A. (2015). Fuzzy logic - A personal perspective. *Fuzzy Sets and Systems*, *281*. https://doi.org/10.1016/j.fss.2015.05.009