Durum Makinesi İle Robot Davranış Modelleme
Robot Behavior Modelling With State Machines

Muhammed Kasap
Computer Engineering
Yildiz Technical University, 34220 Istanbul, Turkey
muhammed.kasap@std.yildiz.edu.tr

***Özetçe* —Günümüzde robot davranışlarını modelleme konusunda yaygın olarak kullanılan 2 ana yaklaşım mevcut, birisi davranış ağacı ile robot davranış modelleme, diğeri ise durum makinesi ile robot davranış modellemedir. Projemiz, ROS (Robot Operating System), Gazebo simülasyon ortamı ve Python programlama dili üzerinde SMACH kütüphanesi kullanarak, robot davranışlarını bir durum makinesi kullanarak modellemeyi hedefledi. Bu modellemeyi gösterebilmek adına bir örnek bir görev dizisini yerine getirmeyi amaçladık. Görev dizimiz bir etrafında dönme ile başlayıp, daha sonra içerisine konduğu haritanın henüz keşfedilmemiş noktalarını keşfeden bir keşif aşaması içerdi. Keşif bittiğinde başlangıç noktasına geri dönerek görev dizisini bitirdi. Bu görevleri yaparken kullanacağımız , farklı boyutlarda 3 harita ve sayısız başlangıç noktası vardır.**

Anahtar Kelimeler—ROS (Robot Operating System), Gazebo, SMACH, Durum Makinesi, Görev Senaryosu

***Abstract*—Currently, there are two main approaches widely used in modeling robot behaviors: one is modeling robot behavior with a behavior tree, and the other is modeling robot behavior with a state machine. Our project aimed to model robot behaviors using the SMACH library on the ROS (Robot Operating System), Gazebo simulation environment, and Python programming language. To demonstrate this modeling, we aimed to accomplish an example task sequence. Our task sequence started with rotating around a point and then included an exploration phase where the robot explored the undiscovered points on the map it was placed in. Once the exploration was complete, the robot returned to the starting point, concluding the task sequence. During these tasks, we had various maps of different sizes and numerous starting points at our disposal.**

Keywords—ROS (Robot Operating System), Gazebo, SMACH, State Machine, Task Scenario

I. Introduction

As robot technology advances rapidly over time, the emergence of hardware structures that are shrinking in size, becoming more cost-effective, easily procurable, and providing high efficiency and effectiveness expands the application areas of robots. However, the proliferation of these robot applications necessitates the resolution of various issues. Especially in parallel with hardware advancements, the development of software solutions has gained significant importance. Therefore, various algorithms and methods have been developed and implemented.[1]

Initially, robots were limited to performing repetitive tasks in industrial environments using specialized software and controllers. However, later advancements such as the acquisition of mobile capabilities by robots, sensors operating with sensitivity similar to human senses, and the development of information communication features enabled robots to interact with designed environments and humans. This situation led to the increasing complexity of systems.[2] With the expected norm of systems being able to simultaneously perform tasks such as environment perception, task definition, movement, and energy supply, numerous algorithms have been developed for this purpose.

As the diversity and complexity of robotic systems increase in today's world, the effective programming and control of these systems have become increasingly crucial. At this point, the Robot Operating System (ROS) and the Gazebo simulation environment have evolved into powerful tools to optimize and enhance robot programming processes. Alongside these tools, there are two main approaches fundamentally used: one involves the use of behavior trees, and the other involves the use of state machines. This project aims to integrate state machines using ROS and Gazebo to enable a robot to perform a complex task more efficiently.[3][4]

The upcoming project will enable us to understand how state machines can be effectively utilized in the programming and control processes of robotic systems. This knowledge can be applied in future robot programming projects and contribute to the more reliable and flexible management of robotic systems.

II. Design and Methodology

A. Installation and Getting Started

In the scope of this project, we will implement robotic task management using the Python library called SMACH. Our project includes the Robot Operating System (ROS) Noetic version and the Gazebo simulation environment. By opting for the Ubuntu operating system, we will install ROS Noetic and create necessary simulation environments in Gazebo using the Turtlebot3 robot. Consequently, we will develop a modular task completion simulation utilizing the SMACH library. Within the project, we will manage tasks such as the robot rotating around itself, mapping its surroundings, and returning to a specified starting point. The visual representation of the Turtlebot3 model we will use in the simulation, specifically the wafflepi model, is depicted in the image below.

Figure 1 TurtleBot3 WafflePi

B. Rotating Around Itself

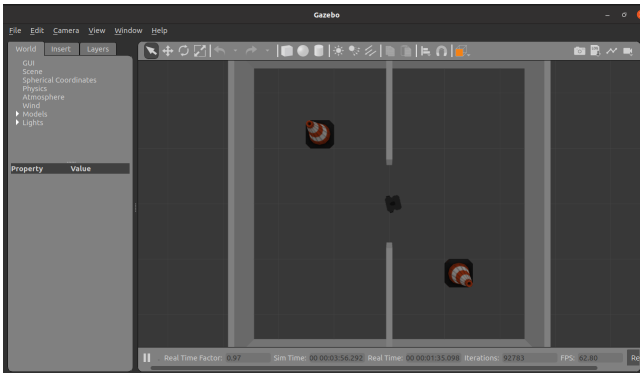

Figure 2 Rotating Around Itself-1

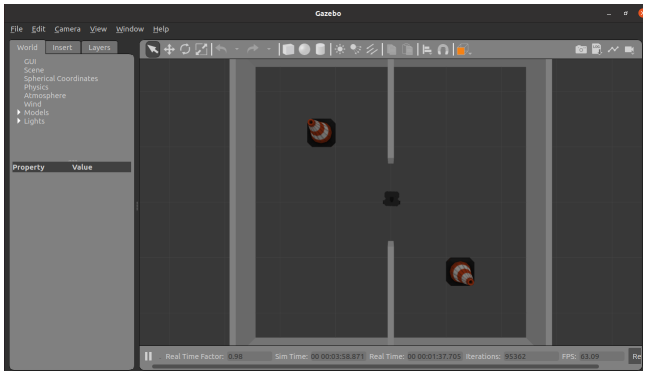

Figure 3 Rotating Around Itself-2

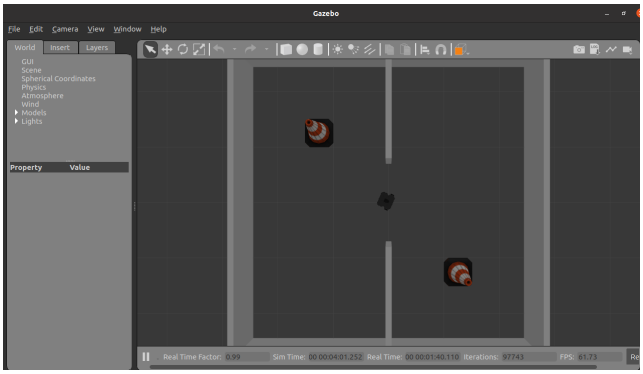

Figure 4 Rotating Around Itself-3

In Figure-2, Figure-3, and Figure-4, as shown above, the simulation environment and the robot that will operate in the environment are the only initial requirements. After meeting these prerequisites, we will provide the angular velocity to rotate the robot, and then stop this rotation after completing one full turn during the specified duration. Once the robot completes one full rotation, it will come to a stop.

In my example, I sent a Twist message with an angular velocity of 0.5 via Geometri-msgs/Twist and, after one full turn, set it back to 0. Despite not receiving any error messages in the majority of my trials, the rotation process failed. However, occasionally, it worked as intended, and the robot stopped after completing one full turn.

C. Exloring the Map

Exploring the map is achieved by running the explore.launch file, which is part of the pre-existing package called explore-lite. This package and file are fundamentally based on Charles DuHadway's explore package. The explore.launch file is a script that navigates the map using move-base until there are no more frontier points remaining on the map.[5]

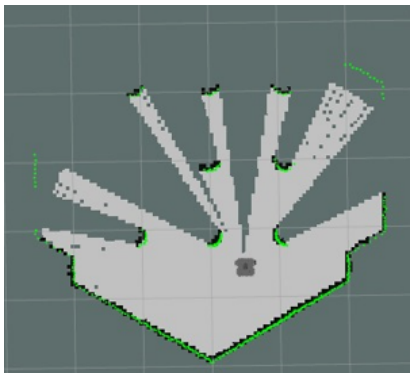

Figure 5 SLAM:Gmapping

In Figure-5, obtained using the SLAM package of the Turtlebot3 robot, you can see the areas that have not been explored yet and our goal is to explore those areas. Our exploration stage, based on the Frontier points shown in Figure-6, discovers these unexplored areas. Figure-7 displays the final state of the exploration process.

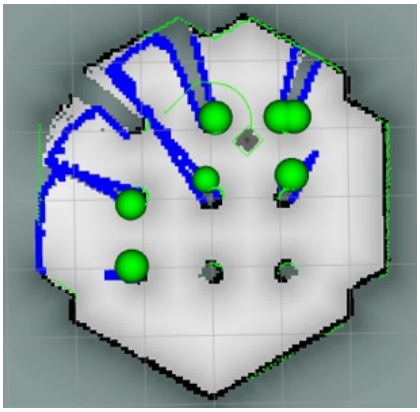

Figure 6 Frontier Spots

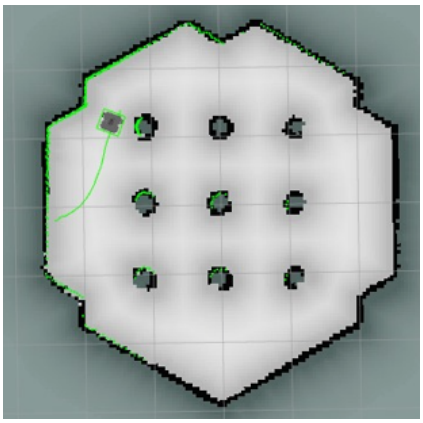

Figure 7 The Final State Of The Map

D. Returning To The Start

In our application, the user can start their robot from any desired location on the map, and after the robot completes the exploration process, it returns to this starting point. In doing so, it utilizes the move-base from the navigation package, setting a target location and providing the robot with the necessary velocity commands to reach that location. The specified addresses are the starting addresses entered by the user at the beginning. Below are examples of return routes to the starting point after exploration for both large and small map scenarios, where the user entered x=0, y=0 for the initial location, and after exploration, the robot returns to the starting point. (x=0, y=0 is the center of the map.)

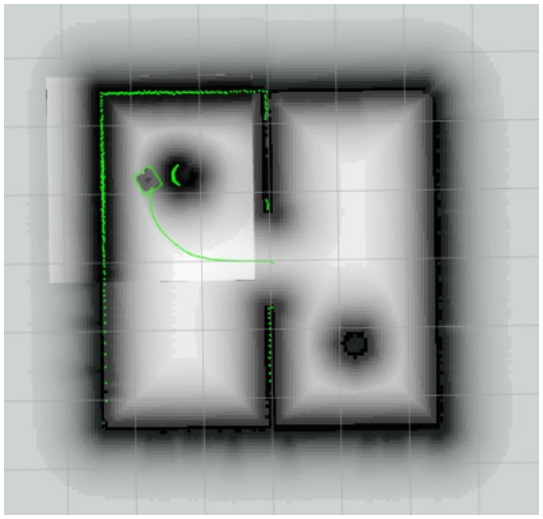

Figure 8 Return to the start on the small map

The maps feature grayscale tones between black and white, representing the cost map used by Turtlebot3. It not only avoids obstacles but also marks areas close to the edges as potential risky points, striving to select the safest path by considering the surroundings.

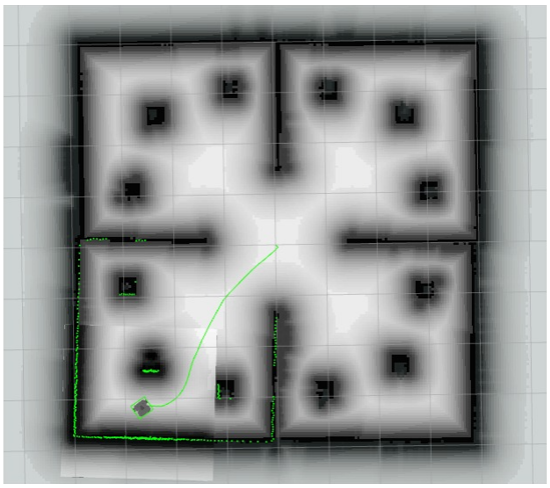

Figure 9 Return to the start on the large map

E. State Machine Formation

After all these implementations, we will use the SMACH library in Python to transform these functionalities into a finite state machine. Subsequently, with the help of the smach-viewer node, we will be able to visualize which state the application is in while it is running. Figure-10 illustrates the state machine for this scenario.

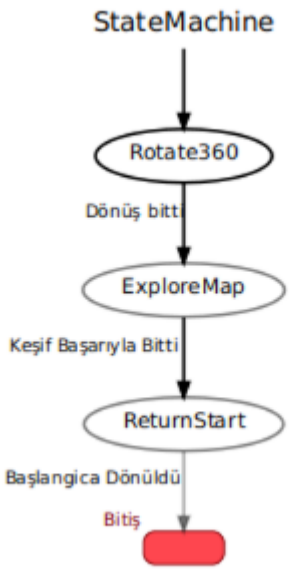

Figure 10 State Machine

III. Experimental Results and Conclusions

A. Experimental Results

Although designing the state machine becomes easy once we have the necessary components, things don't always go as planned, and it is necessary to revisit and modify the required parts. For this, running the program experimentally

multiple times is essential. Tables 1, 2, and 3 show the durations of which states finished on which maps.

Table 1 Rotating Around Itself

Map Size	Time (sec)
Small Map	12,56
Medium Map	12,56
Large Map	12,56

Table 2 Exploration

Map Size	Time (sec)
Small Map	37,66
Medium Map	33,74
Large Map	227,33

Table 3 Returning to the Start

Map Size	Time (sec)
Small Map	22.46
Medium Map	36.40
Large Map	39.84

Sometimes in the program, we need to check various aspects, such as nodes, messages, and the state of the state machine, during both the result phase and the exploration phase. This is why we couldn't consistently work on the same map for every problem we needed to solve.

1) During an initial investigation, we quickly ran the program and examined the output on a small map.
2) When searching for specific information during exploration, we performed the task on a large map to avoid losing the sought-after information.

B. Conclusions

In this project, a modular task completion simulation was developed using state machines on the Turtlebot3 robot. The software, written in the Python programming language and utilizing the Robot Operating System (ROS), successfully controlled the sensors on the Turtlebot3, rotated the robot around itself, performed map exploration, and returned to the starting point. Throughout these tasks, the publisher-subscriber structure in ROS was extensively employed.

The successful completion of this project highlights the potential of state machines and the ROS platform in robotic applications. The obtained results are intriguing in terms of developing modular and flexible robot control systems and can serve as a foundation for similar projects in the future.

IV. Acknowledgements

I would like to express my sincere gratitude to my advisor, Dr. Erkan Uslu, for guiding me throughout this project and providing invaluable support. His expertise in modeling robot behaviors with state machines has been crucial to the success of the project. I am thankful for his encouragement, support, and positive influence, which have significantly contributed to my professional and personal development

References

[1] G. G. ve İ. TÜRKOĞLU, "Robot sistemlerinde kullanılan algoritmalar," no. 1, pp. 17—-31, 2019.

[2] L. S and M. H, "Receding horizon particle swarm optimisation-based formation control with collision avoidance for non-holonomic mobile robots." no. 9, pp. 2075–2083, 2015.

[3] D. Q. Wang T and P. P., "A multi-robot system based on a hybrid communication approach." no. 1, pp. 91–100, 2013.

[4] Ros website. [Online]. Available: https://www.ros.org

[5] Explore lite. [Online]. Available: https://wiki.ros.org/explore$_{l}ite$