Development of an Intelligent Obstacle Avoidance System for Turtlebot3 using ROS and Gazebo

Masters in information technology
Autonomous Intelligent Systems

Member 1: Syed Mostain Ahmed
Matriculation Number: 1390214; Obstacle Avoidance (Group 2)
Email: syed.ahmed@stud.fra-uas.de

Member 2: Farjana Akter
Matriculation Number: 1344037; Obstacle Avoidance (Group 2)
Email: farjana.akter@stud.fra-uas.de

***Abstract*—This paper presents a project on obstacle avoidance using Turtlebot3 and Robot Operating System. The project's main objective was to develop a reliable and efficient obstacle avoidance system for Turtlebot3, which could navigate through complex environments without colliding with obstacles. The project used a Gazebo simulation environment to test the obstacle avoidance system and implemented a Python code to control the Turtlebot3's movement. The methodology involved using various sensors to detect obstacles and implementing a reactive approach to control the robot's movements. The results showed that the system was able to successfully navigate through a simulated environment with various obstacles and avoid collisions. The paper also includes a discussion of the limitations and challenges encountered during the project and provides recommendations for future work. Overall, this project demonstrates the feasibility and potential of using ROS technology and Turtlebot3 for obstacle avoidance applications. The project files and codes can be found here [7].**

***Keywords*—obstacle avoidance, obstacle detection, Turtlebot3, ROS, Gazebo (key words)**

I. INTRODUCTION

In recent years, the development of autonomous robots has gained increasing attention in the field of robotics. One of the primary challenges in developing such robots is to enable them to navigate through complex environments and avoid obstacles without collisions. Obstacle avoidance is essential for autonomous robots that are used in various applications, including robotics, autonomous vehicles, and industrial automation [1]. Turtlebot3 is a popular and cost-effective robot platform that is widely used in research and education [2]. The Robot Operating System (ROS) is a flexible and powerful middleware that provides a platform for developing robot applications [3]. In this project, we develop an obstacle avoidance system for Turtlebot3 using Python programming language based on Robot Operating System (ROS) platform.

The project's primary objective is to create a reliable and efficient obstacle avoidance system that allows Turtlebot3 to navigate through complex environments and avoid obstacles without collisions. The system uses multiple sensors, including a laser range finder, to detect obstacles and implement a reactive approach to control the robot's movements. The reactive approach generates a control command that adjusts the Turtlebot3's movement based on the sensor inputs in real-time. The system can navigate through environments with obstacles of various sizes and shapes.
The project uses a Gazebo simulation environment to test the obstacle avoidance system, and a Python code is implemented to control the Turtlebot3's movement. The Gazebo simulation environment provides a realistic simulation of a real-world environment and allows for testing the obstacle avoidance system under different conditions [4]. The Python programming language is a popular and widely used language in robotics and provides a flexible and powerful tool for controlling the Turtlebot3's movement.

This paper presents the methodology, results, and discussion of the challenges encountered during the project. The paper is organized as follows: Section 2 presents the related work and literature review on obstacle avoidance systems for autonomous robots. Section 3 describes the methodology of the project, including the hardware and software used, and the implementation of the obstacle avoidance system. Section 4 presents the results of the project, including the performance of the obstacle avoidance system under different conditions and also presents the conclusion of the project and highlights the potential of using Python programming language and Turtlebot3 for obstacle avoidance applications in various fields.

II. Related Work

Obstacle avoidance has been extensively researched in the robotics field. Studies have proposed methods such as artificial potential fields, genetic algorithms, and deep reinforcement learning for obstacle avoidance in static and dynamic environments [1]. Some studies have also demonstrated the effectiveness of these methods in real-world scenarios [2]. This paper builds upon existing literature by proposing a hybrid approach using reactive and deliberative methods, implemented using Turtlebot3 and ROS technology for obstacle avoidance.

III. PROJECT IMPLEMENTATION

A) Setup Environment

For developing this project, we need Linux operating system where we can simulate our project. As we don't have Linux operating system, we used Docker Desktop [5] to make our work easier and more efficient. After successfully installed docker desktop, our environment is initially ready for next step. Through a docker windows terminal, we have followed.

```
docker pull tiryoh/ros-desktop-vnc:neotic
```

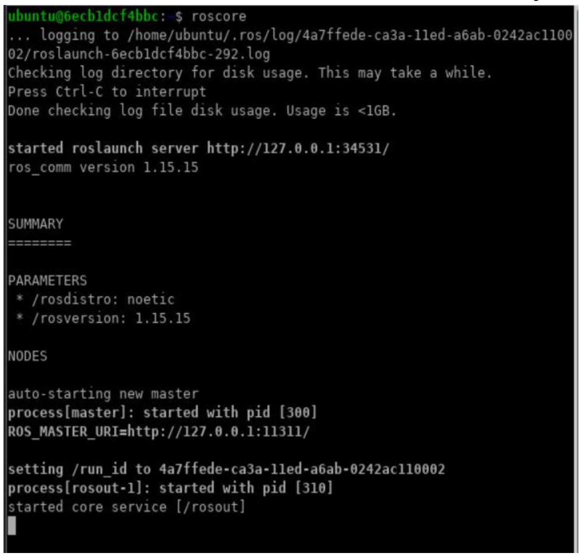
command for pulling the image in our system. Now, docker container is ready for running. Bellow command we used for running docker container.

```
docker run --name ros -p 6080:80 --shm-size=512m tiryoh/ros-desktop-vnc:noetic
```

Once the container has been running, then we must go to browser 127.0.0.1:6080 which will help to do next step with Linux environment.

1) ROS Installation: As we created an autonomous robot that can avoid obstacles, Robot Operating System (ROS) is the most valuable platform.

For ROS Installation, we must open a terminal in Linux operating system, and we can follow all the commands [3]. For us we only need to run the command roscore as we have ros installed already. If

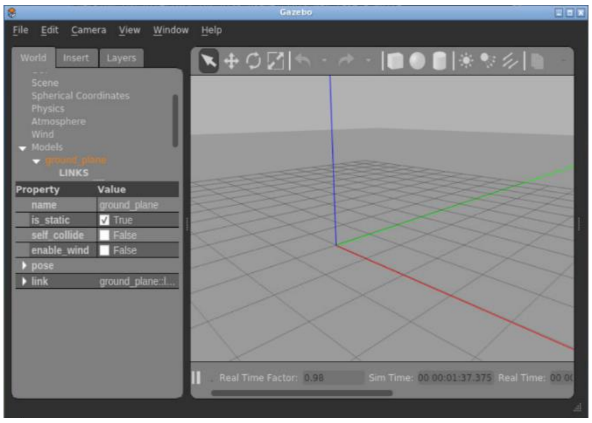


```
ubuntu@6ecb1dcf4bbc: $ roscore
... logging to /home/ubuntu/.ros/log/4a7ffede-ca3a-11ed-a6ab-0242ac1100
02/roslaunch-6ecb1dcf4bbc-292.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://127.0.0.1:34531/
ros_comm version 1.15.15


SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.15.15

NODES

auto-starting new master
process[master]: started with pid [300]
ROS_MASTER_URI=http://127.0.0.1:11311/

setting /run_id to 4a7ffede-ca3a-11ed-a6ab-0242ac110002
process[rosout-1]: started with pid [310]
started core service [/rosout]
```

Figure 1: Running roscore command.

2) Gazebo Simulator: Gazebo is an open-source 3D robotics simulator. Gazebo simulates multiple robots in a 3D environment, with extensive dynamic interaction between objects [4]. For running gazebo simulator, we can follow below command,

```
$ roslaunch gazebo_ros empty_world.launch
```



Property	Value
name	ground_plane
is_static	True
self_collide	False
enable_wind	False
pose	
link	ground_plane::l

Figure 2: Launch empty world in gazebo.

where we can create our own 3d environment. Now our environment is ready for project running.

3) Installing Turtlebot3 robot: Turtlebot3 is the most popular open-source, fully programmable, ROS based mobile robot.

For installing Turtlebot3 robot, we have followed bellow commands step by step,

```
$ sudo apt-get update
```

```
$ sudo apt-get install ros-noetic-navigation
```

```
$ cd Desktop
```

Now, we have created a catkin workspace folder in our Linux desktop that will automatically build all the necessary packages for us.

```
$ mkdir -p catkin_ws/src
```

```
$ cd /home/ubuntu/Desktop/catkin_ws/src
```

For install Turtlebot3 Simulation Package [6], that requires Turtlebot3 and Turtlebot3_msgs packages, we have cloned that have been given following command,

```
$ git clone -b noetic-devel https://github.com/ROBOTIS-GIT/turtlebot3.git
```

```
$ git clone -b noetic-devel https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git
```

Then,

```
$ cd /home/ubuntu/Desktop/catkin_ws/
```

```
$ git clone -b noetic-devel https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git
```

```
$ cd /home/ubuntu/Desktop/catkin_ws/ && catkin_make
```

For launching Simulation World, we have used below command,

```
$ source devel/setup.bash
```

```
$ export TURTLEBOT3_MODEL=burger
```

```
$ roslaunch turtlebot3_gazebo turtlebot3_empty_world.launch
```

At first, we ran a Turtlebot3 empty world, as we shown here,

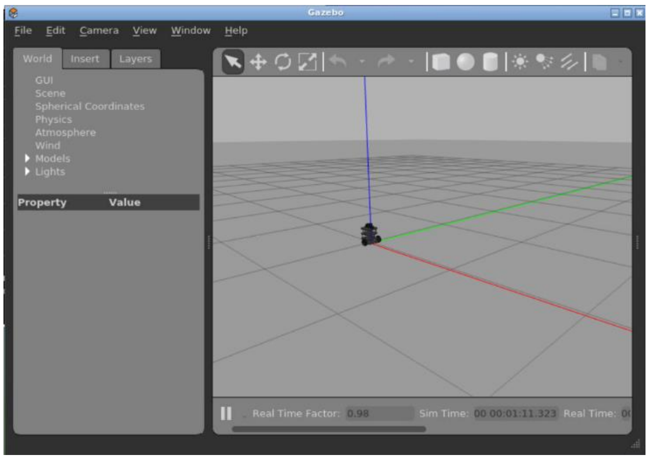


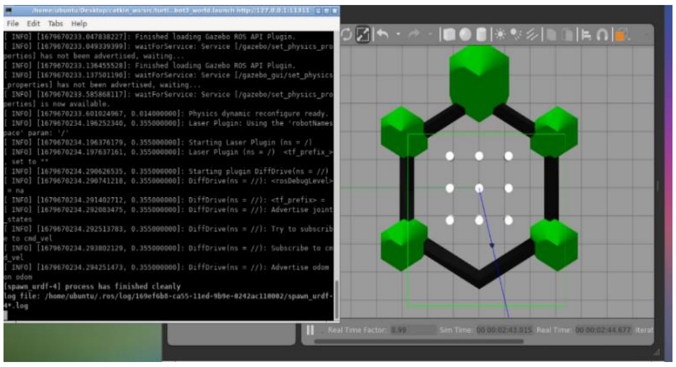
Figure 3: Launch Turtlebot3 empty world.

Now, our environment is totally okay for implementing our project.

B) *Operate Turtlebot3*

As our environment is ready and error-free. Now we will try to develop our main task which is obstacle avoidance by our Turtlebot3 robot. For this reason, we have followed below commands,

```
$ roslaunch Turtlebot3_gazebo Turtlebot3_world.launch
```

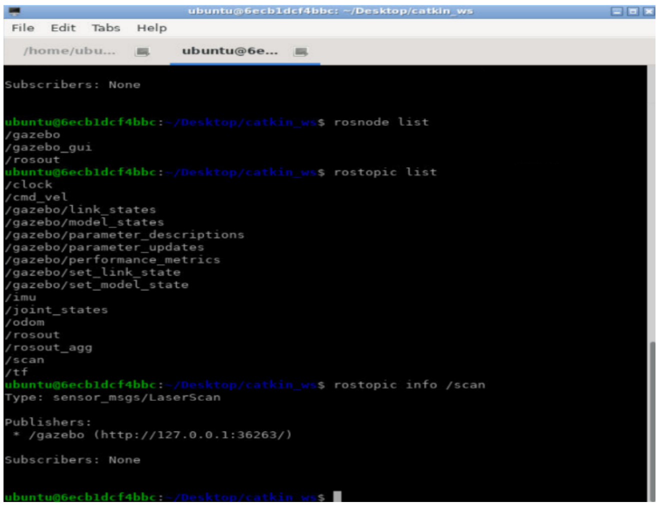


```
/home/ubuntu/Desktop/catkin_ws/src/turtl...bot3_world.launch http://127.0.0.1:11311
[ INFO] [1679670233.047838227]: Finished loading Gazebo ROS API Plugin.
[ INFO] [1679670233.049339399]: waitForService: Service [/gazebo/set_physics_pro
perties] has not been advertised, waiting...
[ INFO] [1679670233.136455528]: Finished loading Gazebo ROS API Plugin.
[ INFO] [1679670233.137501190]: waitForService: Service [/gazebo_gui/set_physics
_properties] has not been advertised, waiting...
[ INFO] [1679670233.585868117]: waitForService: Service [/gazebo/set_physics_pro
perties] is now available.
[ INFO] [1679670233.601024967, 0.014000000]: Physics dynamic reconfigure ready.
[ INFO] [1679670234.196252340, 0.355000000]: Laser Plugin: Using the 'robotNames
pace' param: '/'
[ INFO] [1679670234.196376179, 0.355000000]: Starting Laser Plugin (ns = /)
[ INFO] [1679670234.197637161, 0.355000000]: Laser Plugin (ns = /)  <tf_prefix_>
, set to ""
[ INFO] [1679670234.290626535, 0.355000000]: Starting plugin DiffDrive(ns = //)
[ INFO] [1679670234.290741218, 0.355000000]: DiffDrive(ns = //): <rosDebugLevel>
 = na
[ INFO] [1679670234.291402712, 0.355000000]: DiffDrive(ns = //): <tf_prefix> =
[ INFO] [1679670234.292083475, 0.355000000]: DiffDrive(ns = //): Advertise joint
_states
[ INFO] [1679670234.292513783, 0.355000000]: DiffDrive(ns = //): Try to subscrib
e to cmd_vel
[ INFO] [1679670234.293802129, 0.355000000]: DiffDrive(ns = //): Subscribe to cm
d_vel
[ INFO] [1679670234.294251473, 0.355000000]: DiffDrive(ns = //): Advertise odom
on odom
[spawn_urdf-4] process has finished cleanly
log file: /home/ubuntu/.ros/log/169ef6b8-ca55-11ed-9b9e-0242ac110002/spawn_urdf-
4*.log
```

Figure 4: Launch Turtlebot3 world.

We launched a Turtlebot3_world where we have implemented our python code that can avoid any sizes obstacle.

In addition, we want to mention that our rosnode, rostopic and publisher, subscribers are in below picture,



```
ubuntu@6ecb1dcf4bbc: ~/Desktop/catkin_ws

Subscribers: None


ubuntu@6ecb1dcf4bbc:~/Desktop/catkin_ws$ rosnode list
/gazebo
/gazebo_gui
/rosout
ubuntu@6ecb1dcf4bbc:~/Desktop/catkin_ws$ rostopic list
/clock
/cmd_vel
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/performance_metrics
/gazebo/set_link_state
/gazebo/set_model_state
/imu
/joint_states
/odom
/rosout
/rosout_agg
/scan
/tf
ubuntu@6ecb1dcf4bbc:~/Desktop/catkin_ws$ rostopic info /scan
Type: sensor_msgs/LaserScan

Publishers:
 * /gazebo (http://127.0.0.1:36263/)

Subscribers: None


ubuntu@6ecb1dcf4bbc:~/Desktop/catkin_ws$
```

Figure 5: rostopic, rosnode, ros info / scan

Now, we have run our python file in our Linux terminal by this command,

```
$ python wanderbot.py
```

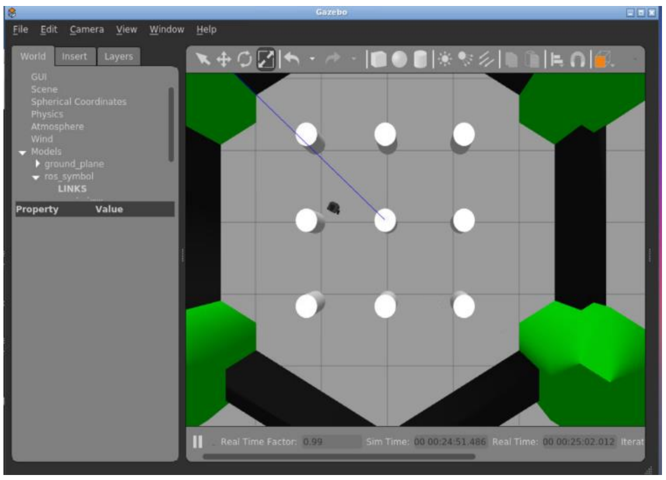
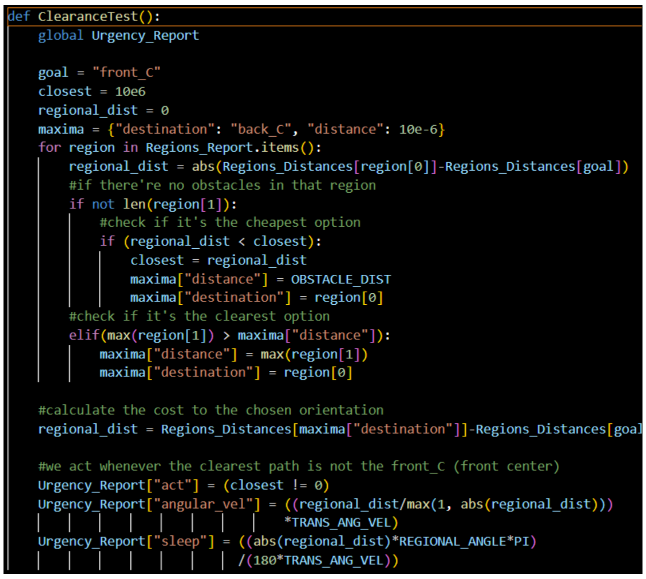


Figure 6: Random picture of our robot avoiding obstacles.

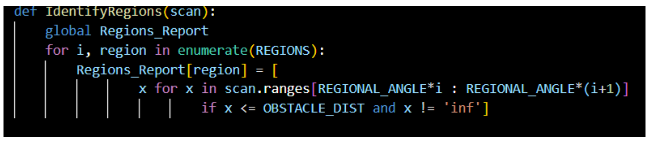
Some important functions of Wanderbot.py code is given below, In ClearanceTest function, the clearest paths are calculated and the appropriate values for the angular_vel and the execution times are assigned.



```python
def ClearanceTest():
    global Urgency_Report

    goal = "front_C"
    closest = 10e6
    regional_dist = 0
    maxima = {"destination": "back_C", "distance": 10e-6}
    for region in Regions_Report.items():
        regional_dist = abs(Regions_Distances[region[0]]-Regions_Distances[goal])
        #if there're no obstacles in that region
        if not len(region[1]):
            #check if it's the cheapest option
            if (regional_dist < closest):
                closest = regional_dist
                maxima["distance"] = OBSTACLE_DIST
                maxima["destination"] = region[0]
        #check if it's the clearest option
        elif(max(region[1]) > maxima["distance"]):
            maxima["distance"] = max(region[1])
            maxima["destination"] = region[0]

    #calculate the cost to the chosen orientation
    regional_dist = Regions_Distances[maxima["destination"]]-Regions_Distances[goal

    #we act whenever the clearest path is not the front_C (front center)
    Urgency_Report["act"] = (closest != 0)
    Urgency_Report["angular_vel"] = ((regional_dist/max(1, abs(regional_dist)))
                                        *TRANS_ANG_VEL)
    Urgency_Report["sleep"] = ((abs(regional_dist)*REGIONAL_ANGLE*PI)
                                    /(180*TRANS_ANG_VEL))
```

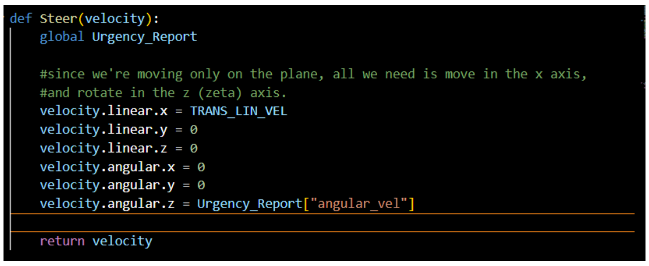
Figure 7: Snippet of ClearanceTest algorithm.

For identify region,



```python
def IdentifyRegions(scan):
    global Regions_Report
    for i, region in enumerate(REGIONS):
        Regions_Report[region] = [
                x for x in scan.ranges[REGIONAL_ANGLE*i : REGIONAL_ANGLE*(i+1)]
                    if x <= OBSTACLE_DIST and x != 'inf']
```

Figure 8: Snippet of identifying regions.



```python
def Steer(velocity):
    global Urgency_Report

    #since we're moving only on the plane, all we need is move in the x axis,
    #and rotate in the z (zeta) axis.
    velocity.linear.x = TRANS_LIN_VEL
    velocity.linear.y = 0
    velocity.linear.z = 0
    velocity.angular.x = 0
    velocity.angular.y = 0
    velocity.angular.z = Urgency_Report["angular_vel"]

    return velocity
```

Figure 9: Snippet of assigning angular velocity

IV. RESULT

Our objective was to make the Turtlebot3 navigate in an environment with obstacles and avoid them without collision. To achieve this, we used Laser Scan data from the robot's sensors and developed a custom algorithm in Python.

The algorithm works as follows: First, we set a safe distance threshold for the robot to avoid obstacles. Next, we analyzed the LaserScan data to detect any obstacles within the safe distance range. If an obstacle is detected, the robot rotates in place until it finds a clear path to move forward. If no clear path is found within a certain time limit, the robot stops and waits for further instructions.

To test our implementation, we launched Turtlebot3 in a simulated environment in Gazebo with randomly placed obstacles. Our algorithm is capable of detecting obstacles and navigating around them to reach the desired destination safely. Future work can include improving the algorithm's efficiency and testing it in real-world scenarios.

References

[1] Y. F. Chen, M. Liu, M. Everett and J. P. How, "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," 2017 IEEE International Conference on Robotics and Automation (ICRA), 2017, pp. 285-292, doi: 10.1109/ICRA.2017.7989037. *(references)*

[2] A. -C. Stan, "A decentralised control method for unknown environment exploration using Turtlebot3 multi-robot system," 2022 14th International Conference on Electronics, Computers and Artificial Intelligence (ECAI), 2022, pp. 1-6, doi: 10.1109/ECAI54874.2022.9847497.

[3] *http://wiki.ros.org/ROS/Installation*

[4] *https://classic.gazebosim.org*

[5] *https://www.docker.com/products/docker-desktop/*

[6] *https://emanual.robotis.com/docs/en/platform/Turtlebot3/simulation/#gazebo-simulation*

[7] *https://github.com/farjana-akter-shifa/ObstacleAvoidanceProject*