Lawn Mover Robot

A PROJECT REPORT

Submitted by

Team D-19

D. MOHAMMAD ABDULLA **BL.EN.U4AIE21044**

G. TEJDEEP REDDY **BL.EN.U4AIE21048**

G. MUKESH VENKATA SAI **BL.EN.U4AIE21050**

for the course

21AIE213 - Robotic Operating System and Robot Simulation

AMRITA SCHOOL OF ENGINEERING, BANGALORE

AMRITA VISHWA VIDHYAPEETHAM

BANGALORE-560 035

June 2023

Contents

1. Problem Statement

Design and implement an lawn mowing robot using a custom built differential drive robot. The robot should traverse to all the locations of the environment, the boundaries of the
environment should be exclusively given as inputs in terms of boundary coordinated, also assume some static objects placed in the environments. The total number of laps should be
taken as input. The type of sensors can be team's choice but the reason to use the sensor should be clear in the report and the presentation.

2. Introduction

Objective: This project's goal is to investigate the design and operation of a Lawn robot by looking at its essential parts, navigational systems, mow processes, and cognitive decision-making skills.

Lawn Mover robots have become creative ways to automate the time-consuming and laborious chore of cleaning various interior spaces. These machines are made to autonomously move around and mow surfaces such as grass, dry flower, and others, relieving humans of the tiresome task of manual mowing. Lawn robots are becoming more complex, effective, and able to provide dependable mow results because to developments in robotics, sensor technologies, and artificial intelligence.

To observe and comprehend their surroundings, lawn robots often use a variety of sensors, including cameras, LIDAR, ultrasonic sensors, and touch sensors. The robot can move through intricate layouts, recognize obstructions, and avoid collisions with other items and people thanks to these sensors. The robot uses sophisticated mapping and localization algorithms to create an accurate map of the Lawn area and pinpoint its exact placement within the surroundings.

Lawn robot functioning heavily relies on intelligent decision-making. These robots can alter their cutting patterns, optimize their routes, and change the intensity of their mowing based on the particular requirements of the environment thanks to advanced algorithms and machine learning techniques. The robot can effectively mow huge areas while minimizing energy use and maximizing mowing efficacy thanks to its intelligence.

The use of lawn robots has several advantages, such as higher productivity, better mowing results, and less labor-intensive cleaning. Because of their autonomy, these robots allow humans to

focus on more difficult jobs. Additionally, they can mow hard-to-reach places where hand cleaning is difficult, including under objects in lawn or in small spaces. Additionally, particularly in delicate spaces like hospitals and labs, lawn robots help to maintain a sanitary and sanitized atmosphere.

3. How Lawn Mower Robots Work?

Navigation: Lawn mower robots use various navigation systems to move around the lawn and avoid obstacles. They typically employ a combination of sensors, including ultrasonic sensors, infrared sensors, and sometimes even GPS. These sensors help the robot detect objects like trees, rocks, or fences, allowing it to adjust its path accordingly.

Boundary setup: Before operation, the lawn owner needs to define the boundaries of the mowing area. This is typically done by installing a boundary wire around the perimeter of the lawn. The wire emits a low-frequency signal that the robot can detect, helping it understand the boundaries it should stay within.

Mowing Patterns: Lawn mower robots are programmed to follow specific mowing patterns to ensure efficient and thorough coverage of the lawn. They can use various patterns, such as random, spiral, or grid, to navigate the lawn systematically. These patterns help prevent the robot from repeatedly mowing the same areas and ensure even grass cutting.

Grass Detection: Robotic lawn mowers use onboard sensors to detect the height of the grass. This information allows them to determine when the grass has grown beyond a specified length and needs to be mowed. The mower adjusts its cutting height accordingly to maintain an even grass height throughout the lawn.

Cutting Mechanism: Most lawn mower robots are equipped

with rotating blades or a cutting disc to trim the grass. These cutting components are usually located underneath the robot and spin at high speeds to efficiently cut the grass blades as the robot moves across the lawn. Safety features, such as lift sensors, are incorporated to automatically stop the blades if the robot is lifted or tilted.

Battery and charging: Lawn mower robots are powered by rechargeable batteries. When the battery level runs low, the robot autonomously returns to its docking station, which is connected to a power source. The robot aligns itself with the docking station and charges its batteries until they are sufficiently replenished. Once fully charged, it resumes mowing from the point it left off.

Programming and scheduling: Lawn mower robots often come with programming options that allow users to set schedules for mowing. Owners can specify the days and times they want the robot to operate, ensuring that the lawn is consistently maintained without manual intervention.

4.Safety Features

Lift sensors: These sensors detect when the robot is lifted or tilted. If the robot is lifted off the ground, such as when a person tries to pick it up, the lift sensors trigger an immediate response to stop the cutting blades from spinning. This feature helps prevent accidents and injuries by ensuring that the blades only operate when the robot is in contact with the ground.

Obstacle detection: Lawn mower robots are equipped with sensors that detect obstacles in their path, such as trees, rocks, or other objects. When an obstacle is detected, the robot adjusts its course to avoid collision. Some advanced models may even have the ability to navigate around obstacles or work in complex lawn layouts, further enhancing safety.

Boundary detection: Lawn mower robots typically operate within a predefined mowing area set by a boundary wire installed around the perimeter of the lawn. If the robot detects the boundary wire, it recognizes it as the limit of its operating area and changes direction to stay within the defined boundaries. This

prevents the robot from wandering into areas where it shouldn't mow, such as flower beds or water features.

Emergency stop button: Many lawn mower robots have an easily accessible emergency stop button. Pressing this button immediately halts all operation, including the cutting blades and movement of the robot. This feature allows users to quickly stop the robot's actions in case of an emergency or if they need to intervene for any reason.

PIN code or app lock: Some lawn mower robots offer a security feature where they can be locked with a PIN code or through a dedicated mobile app. This prevents unauthorized access or usage, ensuring that only authorized users can operate the robot.

Weather sensors: Certain lawn mower robots are equipped with weather sensors that can detect adverse conditions, such as heavy rain or high humidity. When these conditions are detected, the robot may automatically pause its operation and return to its docking station until the weather improves. This feature helps protect the robot from potential damage caused by inclement weather.

4. Implementation

- Robot Model:

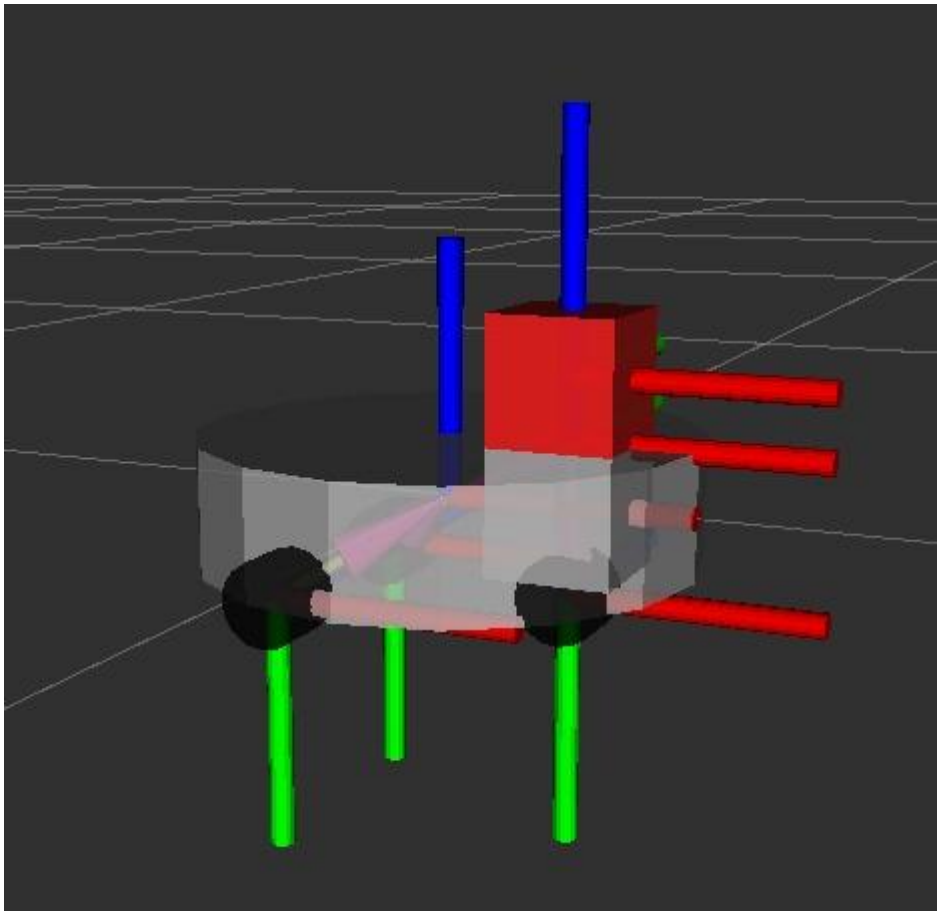
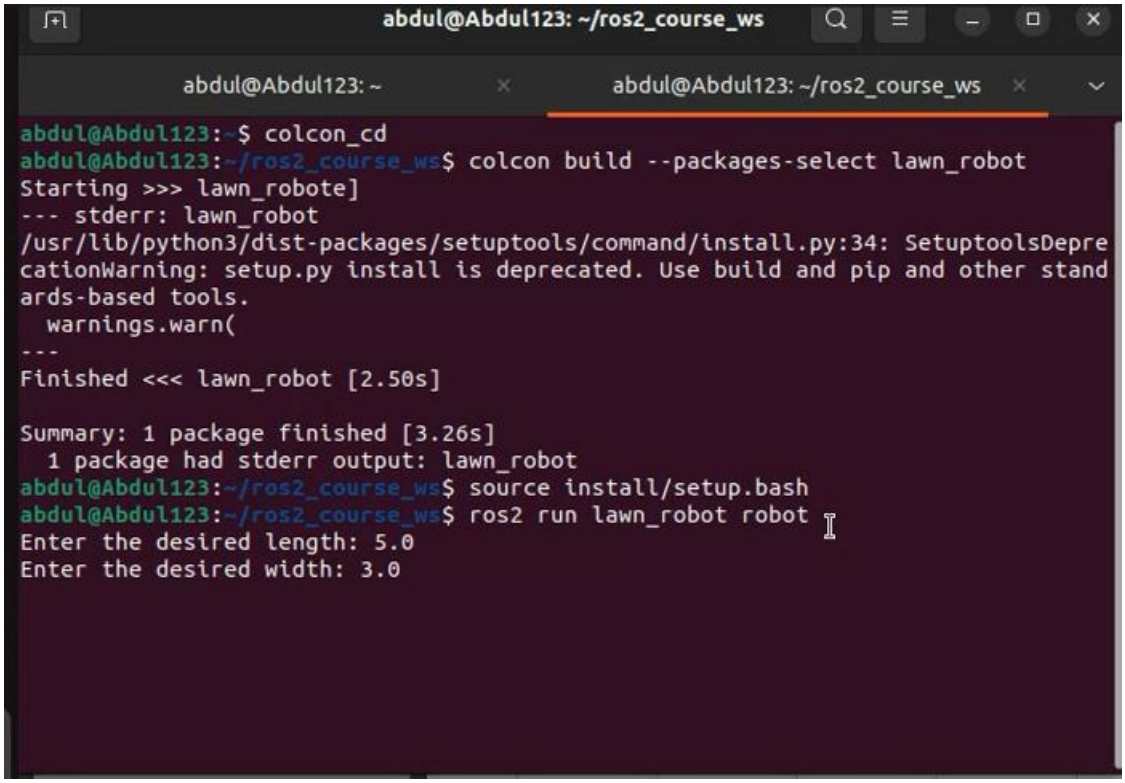
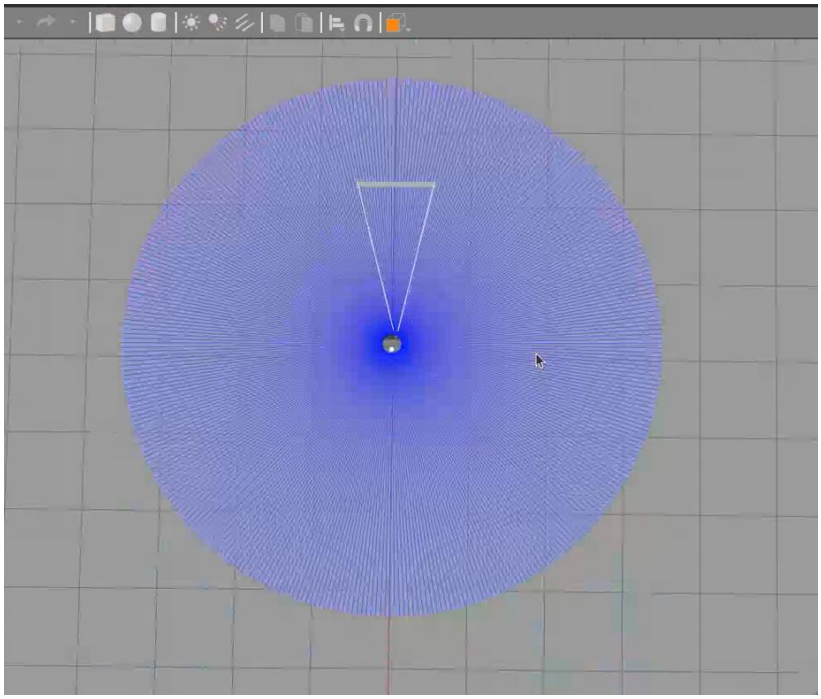


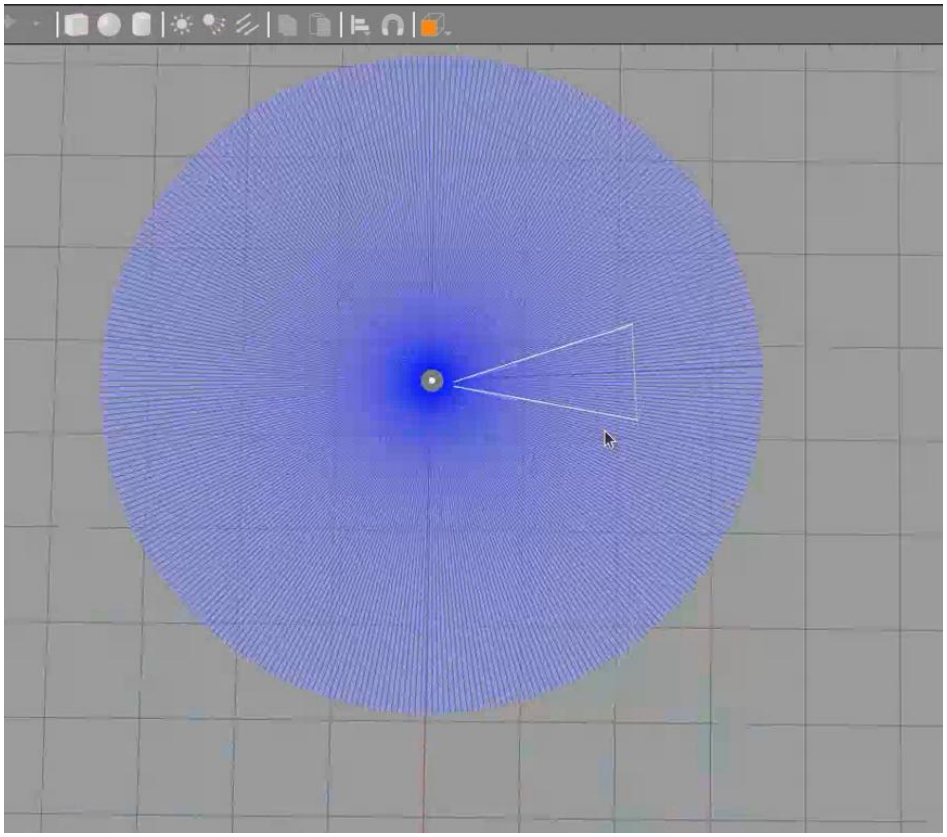
Fig-1, Lawn Moving Robot

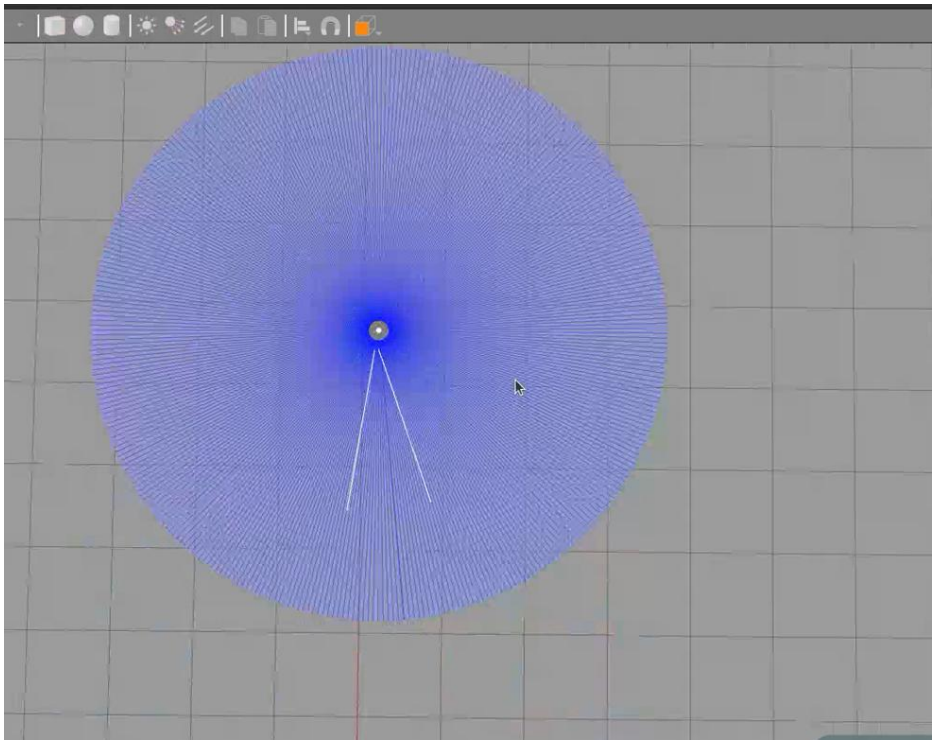
Inputs for Robot



OUTPUT: Robot Running







- **Code :**

  - **URDF :**

```
<?xml version="1.0"?>

<robot name="torroisebot">

  <!-- Define the base link -->
  <link name="base_link">
    <visual>
      <geometry>
        <cylinder length="0.1" radius="0.2"/>
      </geometry>
      <material name="silver">
                        <color rgba="0.75 0.75 0.75 1"/>
      </material>
      </visual>
      <collision>
                <geometry>
                        <cylinder length="0.1" radius="0.2"/>
                </geometry>
      </collision>
      <inertial>
      <mass value="1.0"/>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <inertia ixx="0.083333" ixy="0" ixz="0" iyy="0.083333"
iyz="0" izz="0.016667"/>
    </inertial>
  </link>
  <link name="front_caster">
            <visual>
                <geometry>
                        <box size="0.1 0.1 0.2"/>
                </geometry>
                <material name="silver">
```

```
                    <color rgba="0.75 0.75 0.75 1"/>
                </material>
            </visual>
            <collision>
                <geometry>
                    <box size="0.1 0.1 0.3"/>
                </geometry>
            </collision>
            <inertial>
                <mass value="0.1"/>
                <inertia ixx="0.00083" iyy="0.00083"
izz="0.000167" ixy="0" ixz="0" iyz="0"/>
            </inertial>
        </link>
  <joint name="front_caster_joint" type="continuous">
            <axis xyz="0 0 1"/>
            <parent link="base_link"/>
            <child link="front_caster"/>
            <origin rpy="0 0 0" xyz="0.1 0 0.05"/>
        </joint>
  <!-- Define the wheel links -->
  <link name="wheel_1">
    <visual>
                <geometry>
                    <cylinder length="0.05" radius="0.035"/>
                </geometry>
                <material name="black">
                <color rgba="0.0 0.0 0.0 1"/>
            </material>
   </visual>
     <collision>
                <geometry>
                    <cylinder length="0.05" radius="0.035"/>
                </geometry>
     </collision>
     <inertial>
                <mass value="0.1"/>
```

```
                    <inertia ixx="5.1458e-5" iyy="5.1458e-5"
izz="6.125e-5" ixy="0" ixz="0" iyz="0"/>
     </inertial>

  </link>

  <link name="wheel_2">
   <visual>
                    <geometry>
                         <cylinder length="0.05" radius="0.035"/>
                    </geometry>
                    <material name="black">
                    <color rgba="0.0 0.0 0.0 1"/>
               </material>
                    </visual>
                    <collision>
                    <geometry>
                         <cylinder length="0.05" radius="0.035"/>
                    </geometry>
     </collision>
     <inertial>
                    <mass value="0.1"/>
                    <inertia ixx="5.1458e-5" iyy="5.1458e-5"
izz="6.125e-5" ixy="0" ixz="0" iyz="0"/>
               </inertial>

  </link>

  <link name="wheel_3">
    <visual>
                    <geometry>
                         <cylinder length="0.05" radius="0.035"/>
                    </geometry>
                    <material name="black">
                    <color rgba="0.0 0.0 0.0 1"/>
               </material>
                    </visual>
```

```
    <collision>
                <geometry>
                    <cylinder length="0.05" radius="0.035"/>
                </geometry>
    </collision>
    <inertial>
                <mass value="0.1"/>
                <inertia ixx="5.1458e-5" iyy="5.1458e-5"
izz="6.125e-5" ixy="0" ixz="0" iyz="0"/>
    </inertial>

  </link>

  <!-- Define the base link joint -->
  <joint name="base_joint" type="continuous">
    <axis xyz="0 0 1"/>
    <parent link="base_link"/>
    <child link="wheel_1"/>
    <origin xyz="-0.1 0.1 -0.07" rpy="-1.5708 0 0"/>
  </joint>

  <!-- Define the wheel 2 joint -->
  <joint name="wheel_2_joint" type="continuous">
    <axis xyz="0 0 1"/>
    <parent link="base_link"/>
    <child link="wheel_2"/>
    <origin xyz="-0.1 -0.1 -0.07" rpy="-1.5708 0 0"/>
  </joint>

  <!-- Define the wheel 3 joint -->
  <joint name="wheel_3_joint" type="continuous">
    <axis xyz="0 0 1"/>
    <parent link="front_caster"/>
    <child link="wheel_3"/>
    <origin xyz="0 0 -0.12" rpy="-1.5708 0 0"/>
  </joint>
  <link name="hokuyo_link">
```

```
            <collision>
                <origin xyz="0 0 0" rpy="0 0 0"/>
                <geometry>
                    <box size="0.1 0.1 0.1"/>
                </geometry>
            </collision>
            <visual>
                <origin xyz="0 0 0" rpy="0 0 0"/>
                <geometry>
                    <box size="0.1 0.1 0.1"/>
                </geometry>
            </visual>
            <inertial>
                <mass value="1e-5" />
                <origin xyz="0 0 0" rpy="0 0 0"/>
                <inertia ixx="1e-6" ixy="0" ixz="0" iyy="1e-
6" iyz="0" izz="1e-6" />
            </inertial>
        </link>
        <gazebo>
            <plugin name="diff_drive"
filename="libgazebo_ros_diff_drive.so">
                <left_joint>base_joint</left_joint>
                <right_joint>wheel_2_joint</right_joint>

        <robotBaseFrame>base_link</robotBaseFrame>
                <wheel_separation>0.25</wheel_separation>
                <wheel_diameter>0.07</wheel_diameter>
                <publish_wheel_tf>true</publish_wheel_tf>
            </plugin>
        </gazebo>
        <joint name="hokuyo_joint" type="fixed">
            <axis xyz="0 1 0" />
            <origin xyz="0 0 0.05" rpy="0 0 0"/>
            <parent link="front_caster"/>
            <child link="hokuyo_link"/>
        </joint>
```

```
        <gazebo reference="hokuyo_link">
            <sensor type="ray" name="hokuyo">
                <pose>0 0 0 0 0 0</pose>
                <visualize>true</visualize>
                <update_rate>40</update_rate>
                <ray>
                    <scan>
                        <horizontal>
                            <samples>720</samples>
                            <resolution>1</resolution>
                            <min_angle>-
3.14159</min_angle>

        <max_angle>3.14519</max_angle>
                            </horizontal>
                    </scan>
                    <range>
                        <min>0.10</min>
                        <max>30.0</max>
                        <resolution>0.01</resolution>
                    </range>
                </ray>
                <plugin name="laser"
filename="libgazebo_ros_laser.so">
                    <topicName>/scan</topicName>

        <frameName>hokuyo_link</frameName>
                </plugin>
            </sensor>
        </gazebo>

</robot>
```

➢ Boundary Coordinates Node

```
import rclpy
from geometry_msgs.msg import Twist, Pose
from nav_msgs.msg import Odometry
from sensor_msgs.msg import LaserScan
import math
import time

cmd_vel_pub = None
distance = 0.0
desired_length = float(input("Enter the desired length: "))
desired_width = float(input("Enter the desired width: "))
linear_velocity = 1.0
# Initialize the previous pose with None
previous_pose = None

obstacle_detected = False
obstacle_threshold = 0.5

def odometry_callback(msg):
    global distance, previous_pose
    pose = msg.pose.pose

    if previous_pose is not None:

        delta_x = pose.position.x - previous_pose.position.x
        delta_y = pose.position.y - previous_pose.position.y
        delta_distance = math.sqrt(delta_x**2 + delta_y**2)

        distance += delta_distance

    previous_pose = pose

def laser_callback(msg):
    global obstacle_detected
    ranges = msg.ranges
    for i in range(len(ranges)):
        if ranges[i] < obstacle_threshold:
            obstacle_detected = True
            break
```

```
    else:
        obstacle_detected = False

def timer_callback():
    global cmd_vel_pub, distance, flag, count, breadth, obstacle_detected
    stop = Twist()
    start = Twist()
    start.linear.x = linear_velocity
    count = 0
    breadth = 2

    if obstacle_detected:
        # Obstacle detected, take appropriate action
        print("Obstacle detected! Avoiding obstacle.")
        start.linear.x = 0.0
        start.angular.z = 1.1
        cmd_vel_pub.publish(start)
        time.sleep(2.0)
    elif distance < desired_length:
        # Move forward
        cmd_vel_pub.publish(start)
    else:
        if count <= breadth:
            if flag == 1:
                start.linear.x = 0.0
                start.angular.z = 0.0
                cmd_vel_pub.publish(start)
                time.sleep(2.0)
                start.linear.x = 0.0
                start.angular.z = -1.3
                cmd_vel_pub.publish(start)
                time.sleep(2.0)
                start.linear.x = 0.0
                start.angular.z = 0.0
                cmd_vel_pub.publish(start)
                time.sleep(2.0)
                start.linear.x = 0.2
                start.angular.z = 0.0
                cmd_vel_pub.publish(start)
                time.sleep(2.0)
                start.linear.x = 0.0
                start.angular.z = 0.0
                cmd_vel_pub.publish(start)
```

```
            time.sleep(2.0)
            start.linear.x = 0.0
            start.angular.z = -1.3
            cmd_vel_pub.publish(start)
            time.sleep(2.0)
            start.linear.x = 0.0
            start.angular.z = 0.0
            cmd_vel_pub.publish(start)
            time.sleep(3.0)
            flag = 0
            distance = 0
            pass
        else:
            start.linear.x = 0.0
            start.angular.z = 0.0
            cmd_vel_pub.publish(start)
            time.sleep(2.0)
            start.linear.x = 0.0
            start.angular.z = 1.3
            cmd_vel_pub.publish(start)
            time.sleep(2.0)
            start.linear.x = 0.0
            start.angular.z = 0.0
            cmd_vel_pub.publish(start)
            time.sleep(2.0)
            start.linear.x = 0.2
            start.angular.z = 0.0
            cmd_vel_pub.publish(start)
            time.sleep(2.0)
            start.linear.x = 0.0
            start.angular.z = 0.0
            cmd_vel_pub.publish(start)
            time.sleep(2.0)
            start.linear.x = 0.0
            start.angular.z = 1.3
            cmd_vel_pub.publish(start)
            time.sleep(2.0)
            start.linear.x = 0.0
            start.angular.z = 0.0
            cmd_vel_pub.publish(start)
            time.sleep(3.0)
            flag = 1
            distance = 0
```

```
            pass
        else:
            cmd_vel_pub.publish(stop)
    count += 1

def main(args=None):
    global cmd_vel_pub, flag
    rclpy.init()
    flag = 1
    node = rclpy.create_node('move_distance_node')
    cmd_vel_pub = node.create_publisher(Twist, 'cmd_vel', 1)
    node.create_subscription(Odometry, 'odom', odometry_callback, 10)
    node.create_subscription(LaserScan, 'laser_scan', laser_callback, 10)  #
Subscribe to LaserScan topic
    timer_period = 0.1  # Define the time interval between movements (adjust as
needed)
    node.timer = node.create_timer(timer_period, timer_callback)
    rclpy.spin(node)

if __name__ == '__main__':
    main()
```

- **Sensors Used:**

Laser Scanner:

A laser scanner sensor, also known as a lidar sensor (Light Detection

and Ranging), is a device that uses laser beams to measure distances and create detailed 3D representations of its surroundings. It is commonly used in various applications such as robotics, autonomous vehicles, surveying, mapping, and environmental monitoring.

Use of the sensor in the project:

For obstacle detection and obstacle avoidance.

5. Conclusion

An automatic lawn mover robot essentially has 2 important features that we concluded from our research :

1. Mapping the environment
2. Finding the optimal path to cover all the free places in the environment

These are the basic 2 features offered by such robots.
Our robot too works on a mapping path very similar to that of a mowbot who maps the environment by turning in the opposite direction of the obstacle at a random angle, and the actual mow process of our robot is similar to that of MI lawn bot which travels in a ‘S’ shape to cover all the free space available in the map.