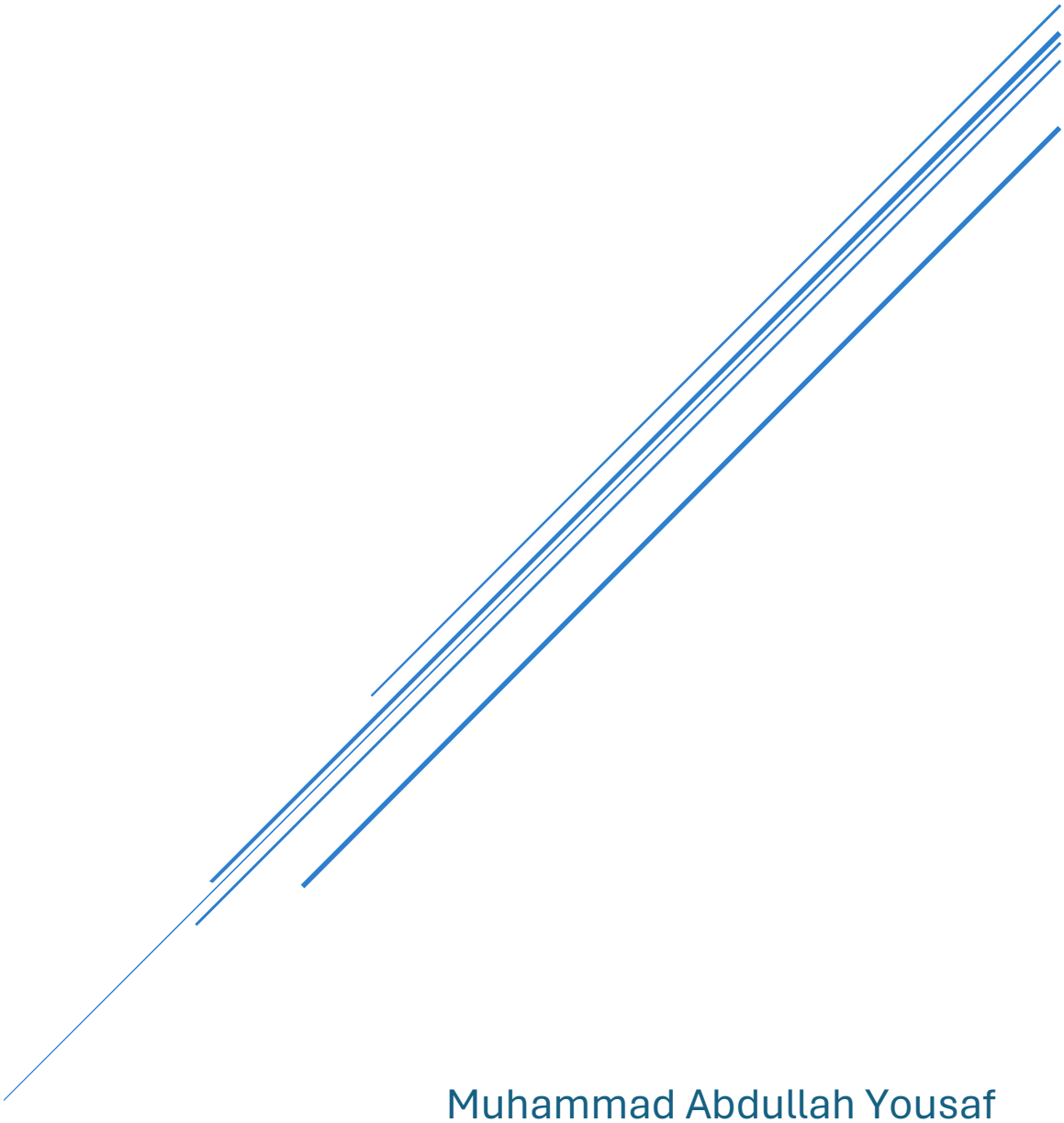
MOBILE ROBOTICS

Assignment 02



Muhammad Abdullah Yousaf
450767

Wall-Following Algorithm Implementation for TurtleBot3 in ROS Noetic

Introduction

This report describes the development and implementation of a wall-following algorithm for the TurtleBot3 robot using ROS Noetic. The algorithm utilizes a PID controller to adjust the robot's path for precise navigation.

1. System Setup

1.1 ROS Environment

ROS Distribution: ROS Noetic

Simulator: Gazebo 11

Programming Language: Python 3

1.2 Nodes

wall_follow.py: This node subscribes to laser scan data, computes control commands, and publishes these commands to the robot's velocity controller.

2. Libraries Utilized

2.1 rospy

Used for ROS communication, including topic subscription, message publishing, and node management.

2.2 geometry_msgs

Provides ROS message types such as Twist, which is used to represent linear and angular velocities.

2.3 sensor_msgs

Supplies ROS message types like LaserScan, which represents data from laser sensors.

2.4 numpy

Used for array manipulations and numerical computations on sensor data.

3. System Architecture

3.1 Sensor

The TurtleBot3's laser range finder delivers real-time distance measurements to nearby objects.

3.2 Perception

The `wall_follow` node processes laser scan data to identify the wall.

3.3 Control

The `PIDController` class calculates the robot's angular velocity based on the distance from the wall, aiming to maintain a set distance.

3.4 Action

The `cmd_vel_pub` publishes the calculated velocity commands to the robot's velocity controller, guiding its movements and rotations.

4. Code Implementation

4.1 PIDController Class

Function: Implements the Proportional-Integral-Derivative (PID) control algorithm.

Parameters: Proportional gain (Kp), integral gain (Ki), and derivative gain (Kd).

Computation: Computes the control output based on error, integral, and derivative terms.

4.2 WallFollowerNode Class

Initialization: Initializes the ROS node and subscribes to the laser scan topic.

Publishing: Publishes velocity commands to the `cmd_vel` topic.

Configuration: Defines desired wall distance (`obstacle_threshold`), movement speed (`move_speed`), and rotation speed (`rotation_speed`).

4.3 laser_callback Function

Data Processing: Processes laser scan data to identify the wall.

Error Calculation: Calculates the error between the desired and actual distances to the wall.

PID Computation: Uses the PID controller to determine the angular velocity.

Publishing: Publishes the computed velocity commands to the robot.

5. Key Features

5.1 PID Control

Adjusts the robot's orientation precisely to maintain a consistent distance from the wall.

5.2 Obstacle Avoidance

The algorithm checks for obstacles in front of the robot and adjusts its path accordingly.

5.3 Flexibility

PID gains can be adjusted to fine-tune the robot’s response for optimal wall-following performance.

6. Running the Code

6.1 Start ROS Master

```
roscore
```

6.2 Launch TurtleBot3 Simulation

```
roslaunch turtlebot3_gazebo turtlebot3_house.launch
```

6.3 Run the Wall-Following Node

```
rosrun wall_follow_turtlebot wall_follow.py
```

7. Conclusion

This implementation delivers a functional wall-following system for the TurtleBot3 using PID control. It demonstrates the efficacy of PID control in achieving robust and precise robot navigation. Future enhancements could include more advanced obstacle avoidance strategies and adaptations to various environmental conditions.

8. Future Work

8.1 Enhanced Obstacle Avoidance

Incorporating more sophisticated algorithms to better handle dynamic environments.

8.2 Environmental Adaptation

Adapting the algorithm to perform well in different environmental conditions, such as varying lighting or floor textures.

8.3 Real-World Testing

Extending the simulation results to real-world scenarios to validate the robustness and accuracy of the wall-following behavior