HONOUR'S PLEDGE FORM

In our honour, We, as listed and signed below, as students from the **Fakulti Teknologi Maklumat & Komunikasi (FTMK), Universiti Teknikal Malaysia Melaka (UTeM)**, have neither knowingly given nor received any inappropriate assistance in academic work on this assignment, except within our group/team members, for the course **DITI 3513 AI in Robotics & Automation**. We have also not plagiarized or been complicit with those who do.

We affirm that any use of **Generative AI** (such as Large Language Models, LLM) is only for assistance, and those parts of the work that had been assisted by such tool(s) had been properly **cited and listed in the References section** of our final report.

We pledge that we have been honest and have observed no dishonesty throughout the duration of receiving this independent learning assessment task until submission.

Group Assignment (10%): Wall-Following Behavior

CLO3: *Demonstrate orally or in written form the solution steps in solving robotics problems using Artificial Intelligence techniques. (A3, PLO4, LOD-C3C)*

	Team Leader	**Member 1**	**Member 2**
Signature:			
Full Name:	Alifah Ilyana binti Nor Azmmi	Puteri Husna Umairah binti Shahrul Nizam	Nurnisa Nabihah binti Khalid
Matric Number:	D032310440	D032310452	D032310494
Date:	11/6/2025	11/06/2025	11/06/2025

Problem Understanding & Analysis

The objective of this project is to develop and implement reactive wall-following behavior using the TurtleBot3 platform. The robot should be able to detect a wall using its onboard LiDAR sensor, maintain a safe and consistent distance from it, and adjust its path accordingly. The system must also react to obstacles or corners by modifying its behavior in real time, using only local sensor data.

Analysis

The TurtleBot3 is equipped with a 360° LiDAR sensor, which provides detailed range data suitable for reactive behaviors. By analyzing segments of the LiDAR scan, the robot can

- Detect the closest distance in front (for obstacle detection),
- Monitor the side (for maintaining wall distance),
- Decide when to move forward or turn.

7- Steps Design of the Wall-Following Behavior

1. Describe the task

The primary task is to develop a reactive behavior that enables the TurtleBot3 to autonomously detect and follow along a wall while maintaining a safe distance. The robot should continuously monitor its surroundings and adapt its movement in real time to stay aligned with the wall, handle corners, and avoid obstacles.

2. Describe the Robot

We use the TurtleBot3(Burger) model, a differential drive robot equipped with the following key components

- **LiDAR (360° rotating sensor)** is used to detect distances to walls and obstacles and create distance maps.
- **OpenCR** handles low-level motor control and interfaces with ROS.
- **Single board Computer (Raspberry Pi)** runs ROS Noetic and the behavior of nodes.
- **Power Supply** for the entire robot motors, controller board, and Raspberry Pi.
- **Wheels** enable the robot to move forward, backward, and rotate.
- **Dynamixel Motors** drives the wheels and provides position, velocity, and current data.

3. Describe the Environment

The robot operates in a controlled indoor environment with

- Flat flooring and clear walls (container cover).
- Corners and turn 90 degrees.
- No dynamic (moving) obstacles.

4. Describe how the Robot Should Act in Response to its environment

Robot response

- **Searching for wall** – Robot rotates left to find a wall within threshold range
- **Wall-Following** – Robot moves forward, adjusting angular velocity based on lateral LiDAR distance to maintain alignment
- **Corner Turning** – If the front LiDAR detects an obstacle, the robot turns left.

5. Implement and Refine each Behavior

We implemented our full wall-following behavior in tb3_my_obstacle.py, a ROS Python node that controls the TurtleBot3's motion based on LiDAR data. The robot continuously evaluates distances to the left, right, and front to determine how to steer

- If there's a wall on either side, the robot maintains a safe distance by slightly turning toward or away from the wall, depending on whether it's too close or too far.
- If an obstacle is detected in front (within a safety threshold), the robot stops and performs a left or right turn to avoid a collision.
- If there are no nearby walls, the robot rotates slightly to search for a wall.

This behavior was implemented using the rospy library, LaserScan messages for LiDAR data, and Twist messages to control movement. Through testing, we refined distance thresholds and turning angles to handle real-world noise and ensure smooth, stable wall alignment on both sides.

6. Test each Behavior Independently

Go back to step 4

- Searching for the wall, we placed the robot in an open space and comfirmed it rotated until it detected a wall.

- Wall-Following, the robot's ability to stay parallel to a straight wall was tested and logged.
- Corner Turning, a wall was placed directly in front of the corner, and we ensured the robot executed a controlled turn.

7. Test Behaviors Together

After verifying each behavior independently, we integrated the logic into a single ROS node (`tb3_my_obstacle.py`). We tested this full behavior loop in a controlled environment using either a Gazebo simulation or a real TurtleBot3 robot.

- On the remote PC: "roscore"
- On the TurtleBot3 (via SSH): "roslaunch turtlebot3_bringup turtlebot3_robot.launch"
- Back remote PC: "rosrun my_obstacle tb3_my_obstacle.py"

We placed the robot near a wall and positioned an obstacle ahead to test transitions between wall-following and avoidance behaviors.

Challenges or Limitations

During the development and testing phase of the TurtleBot3 simulation using ROS and Gazebo, several challenges and limitations were encountered. These include hardware-related issues, software bugs, simulation inconsistencies, and constraints in the implemented logic.

1. Hardware Issues

i. LiDAR Detection Problem

During SLAM and navigation testing, the LiDAR occasionally failed to detect small or low-lying obstacles. This caused the robot to collide with objects it should have avoided.

ii. Sensor Noise

Sensor data sometimes included noise, leading to inconsistent distance measurements. This affected the quality of mapping and path planning accuracy.

2. Software Bugs

i. Inaccurate Map Generation

At times, the SLAM-generated map was unclear or inaccurate. This often resulted from unoptimized SLAM parameters or the robot moving too fast during the mapping process.

3. Simulation Issues (Gazebo)

a. Lag and Latency

Gazebo experienced lag, especially when multiple ROS nodes were running simultaneously. This made debugging more difficult as robot movement did not consistently match the issued commands.

b. Model Collision Errors

Some object models in Gazebo had mismatched collision boundaries, causing the robot to either pass through them or stop unexpectedly due to invisible collision areas.

4. Limitations of Logic

a. Poor Handling of Sharp Corners

The path planning algorithms struggle with navigating sharp corners efficiently. The robot often rotated multiple times before finding the correct path.

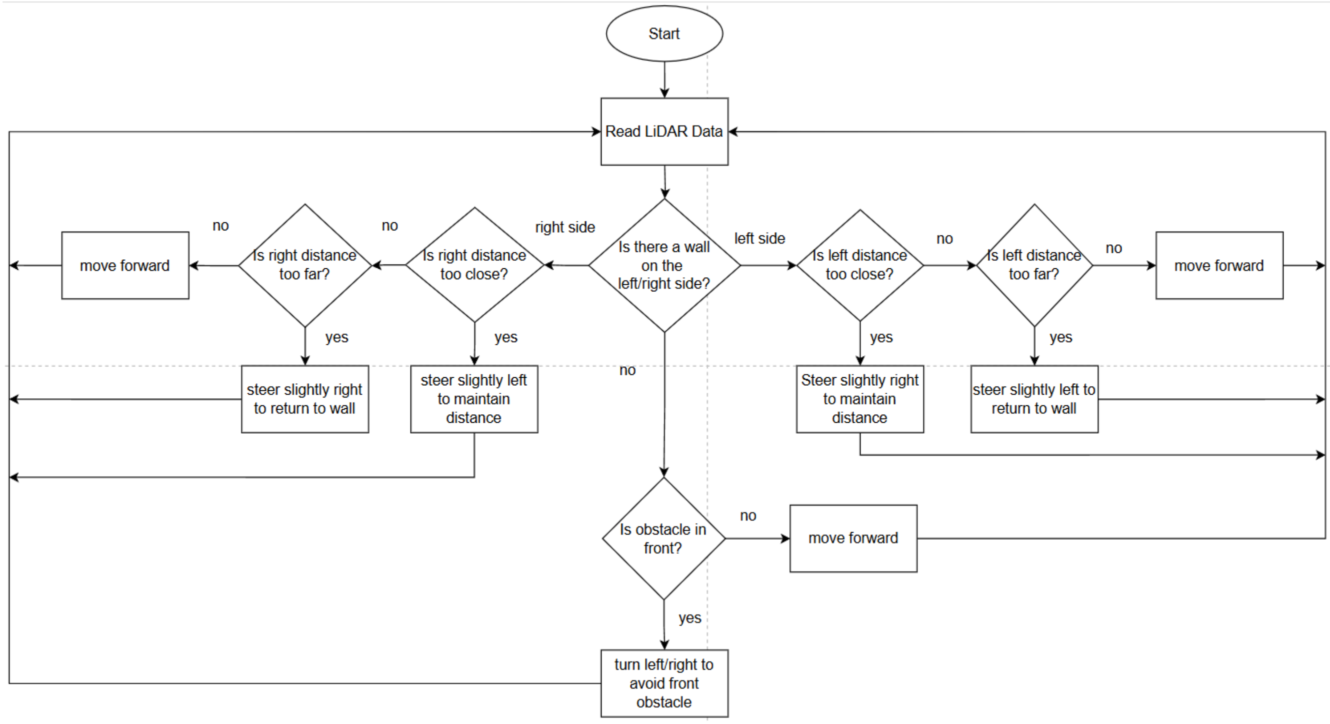
b. No Handling of Dynamic Obstacles

The system was not designed to handle moving obstacles. If an object suddenly appeared in the robot's path, it could not dynamically adjust its route.

c. Over-reliance on Perfect Sensor Data

The navigation logic heavily relied on ideal sensor data. Any inaccuracies led to obstacle detection or mapping failures.

FLOWCHART



Why flowchart?

We chose to use a flowchart because it provides a clear and visual representation of the robot's decision making and movement process. Each step, condition and action is easy to follow using standard shapes and arrows. This makes it ideal for showing how the TB3 robot responds to LiDAR inputs while wall following. Compared to other tools, flowchart is simpler and easier to visualize sequential logic and control flow.

Appendices

i. Wall Follower using LiDAR in Youtube video

https://youtube.com/shorts/CTCOEdeZFXg?feature=share

ii. Coding

my_obstacle.py - C:\Users\User\Documents\my_obstacle.py (3.13.4)

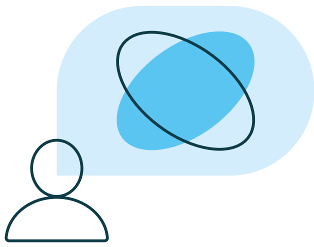
File Edit Format Run Options Window Help

```
#!/usr/bin/env python3

import rospy
import time
import math
from sensor_msgs.msg import LaserScan
from geometry_msgs.msg import Twist

# Constants
MIN_DISTANCE = 0.3   # Minimum safe distance from wall(too close)
MAX_DISTANCE = 1.0   # Desired following distance(too far)
FRONT_THRESHOLD = 0.5  # Distance to trigger front obstacle avoidance
LINEAR_SPEED = 0.2
ANGULAR_ADJUST = 0.2  # Small angle adjustment for wall correction

def get_average_distance(ranges, start_angle, end_angle) :
    # Get a slice of the LiDAR ranges and remove invalid values
    sector = ranges[start_angle:end_angle]
    valid = [r for r in sector if r > 0.0 and not math.isinf(r)]
    if valid:
return sum(valid) / len(valid)
    else:
return float('inf')

def wall_follower() :
    rospy.init_node('wall_follower', anonymous = True)
    pub = rospy.Publisher('/cmd_vel', Twist, queue_size = 10)
    rate = rospy.Rate(10)
    vel = Twist()

    while not rospy.is_shutdown() :
        scan = rospy.wait_for_message('/scan', LaserScan)
        ranges = scan.ranges
        angle_range = len(ranges)

        # LiDAR index mapping based on 360°(TurtleBot3 LiDAR has 0 - 359)
        front = get_average_distance(ranges, 0, 10) + get_average_distance(ranges, 350, 359)
        left = get_average_distance(ranges, 80, 100)
        right = get_average_distance(ranges, 260, 280)

        wall_on_left = MIN_DISTANCE < left < MAX_DISTANCE
        wall_on_right = MIN_DISTANCE < right < MAX_DISTANCE
        obstacle_in_front = front < FRONT_THRESHOLD

        # Reset velocity
```

```
        # Reset velocity
        vel.linear.x = LINEAR_SPEED
        vel.angular.z = 0.0

        if wall_on_left or wall_on_right:
if wall_on_left :
    if left < MIN_DISTANCE :
        rospy.loginfo("Left too close → steer right")
        vel.angular.z = -ANGULAR_ADJUST
        elif left > MAX_DISTANCE :
        rospy.loginfo("Left too far → steer left")
        vel.angular.z = ANGULAR_ADJUST
    else:
rospy.loginfo("Following left wall")
elif wall_on_right :
if right < MIN_DISTANCE :
    rospy.loginfo("Right too close → steer left")
    vel.angular.z = ANGULAR_ADJUST
    elif right > MAX_DISTANCE :
    rospy.loginfo("Right too far → steer right")
    vel.angular.z = -ANGULAR_ADJUST
else:
rospy.loginfo("Following right wall")
else:
if obstacle_in_front :
    rospy.loginfo("Obstacle in front! Turning left")
    vel.linear.x = 0
    vel.angular.z = ANGULAR_ADJUST
else :
    rospy.loginfo("No wall detected, moving forward")

    pub.publish(vel)
    rate.sleep()

    if _name_ == '_main_':
try :
    wall_follower()
    except rospy.ROSInterruptException :
    pass
```

Ethical AI use checklist for students

A checklist is an easy way to guide your usage of AI to avoid any appearance of academic misconduct.



Points to consider **before/during** the assignment

The goal is to check off "yes" for most, if not all, of the points. If you checked "no," consider what steps can be taken to address the issue or complete the task.

	Yes	No
Can I produce **artifacts** of my writing process such as brainstorming, drafting, revisions and/or reflections? **Pro tip #1:** Use Draft Coach or ask your instructor to allow the option of multiple submissions in Turnitin Feedback Studio. **Pro tip #2:** Keep a record of your documents, such as multiple drafts.	☐	☐
Did I follow the instructor's guidance on exactly what and how generative AI tools can be used for this assignment? **Pro tip #3:** Ask clarifying questions as needed.	☐	☐
Did I make the effort to be certain that my writing **style** and voice are evident in my work?	☐	☐
Did I **cite** any AI used to complete my assignment?	☐	☐

Points to consider **after the final submission**

The goal after you've submitted your work is to answer "yes" to ensure you are able to respond to any questions that may arise regarding the ethical use of AI in your assignment. If the answer is "no," what steps can you take to address those areas?

	Yes	No
Am I prepared to **respond calmly** to any questions regarding how and why I used AI in case questions arise? **Consider the following:** • How can I prepare for the discussion? • What reason(s) did I have for using AI to complete a particular passage or section? Was it used for idea generation or to create an outline, for example? • Did my use of AI follow the guidelines given for this particular assignment? • Did I cite my usage correctly according to instructor or institution guidelines?	☐	☐
Am I prepared to discuss my **writing process**? **Consider the following:** • Where/when did I work on the assignment? • Who else read it and possibly provided feedback? • What changes did I make and why? • How did I conduct the necessary research?	☐	☐
Can I clearly outline and explain the **writing choices** I made? **Consider the following:** • What sources did I include or choose not to? • Why did I take this position? • How and why did I select each example/evidence? • Are my personal ideas or experiences evident?	☐	☐
Can I share documentary **artifacts** of my writing process? **Provide the following as needed:** • Notes, outlines, version histories, or drafts - handwritten or electronic, i.e. if you used Turnitin's Draft Coach or the instructor enabled multiple submissions in Turnitin's Feedback Studio. • Feedback from a peer or other trusted reviewer.	☐	☐

To explore more ideas about academic integrity in the age of AI, visit www.turnitin.com/resources/academic-integrity-in-the-age-of-AI