Wasit Journal of Engineering Sciences
Journal homepage: https://ejuow.uowasit.edu.iq
Vol. 14 No. 2 June 2026, pp.277-290
https://doi.org/10.31185/ejuow.v14i2.864

ISSN 2305-6932
Original Research

Implementing Adaptive Cruise Control -Based Model Predictive Control using Turtlebot3 Robot

Farah Mahdi Ali [1], Nizar Hadi Abbas [1]
[1] Electrical Engineering Dept. College of Engineering, University of Baghdad, Baghdad

Corresponding Author Email: farah.m.ali@coeng.uobaghdad.edu.iq

Received Dec.25, 2025 Revised Feb.12, 2026 Accepted Feb.14, 2026 Online Jun.1, 2026

ABSTRACT

Adaptive cruise control (ACC) systems have been developed to enhance the performance of traditional cruise control (CC) systems. ACC efficiently reduces congestion and traffic accidents, decreases drivers' workloads, and improves economic efficiency. This paper first validates and verifies a dual-level controller based on model predictive control (MPC) through an experiment, resulting in the formulation of an adaptive cruise control algorithm that improves robustness and safety. Second, two control methodologies, the MPC and conventional proportional integral derivative (PID) control, are tested and compared. The tests are realized using the TurtleBot3 robot as the host vehicle and a toy car as the target vehicle. This will clarify the relationship between the working relationship of practical and theoretical systems, in addition to defining a description of the controller's operation in a real system. The originality of this work lies in the integration of a two-level architecture with actual hardware, including a Light Detection and Ranging (LiDAR) data filtering method that enhances real-time responsiveness. Results from experimental scenarios show that the MPC controller achieves a 32% reduction in distance tracking error, along with improved robustness, compared to a conventional PID controller. Numerical and experimental results have demonstrated the advantages and effectiveness of the MPC method over the conventional PID method when adapting to disturbances or challenging scenarios is critical.

Keywords: Adaptive Cruise Control, Model Predictive Control, Robotic Operating System, Turtlebot3 Robot.

1. Introduction

Adaptive cruise control (ACC) is an intelligent driver assistance system (ADAS) that makes driving safer and more comfortable by autonomously adjusting a vehicle's speed to keep a secure following distance from the preceding vehicle. Traditional cruise control systems operate at a steady speed while ACC dynamically responds to traffic conditions using sensor inputs, such as radar, Light Detection and Ranging (LiDAR), or cameras. In addition, this feature lets ACC adapt to changing traffic situation, such as slowing down or speeding up as needed, without the driver having to respond manually.

The main goal of ACC systems is to manage vehicle speed and spacing while ensuring safety and saving energy. As a result, a lot of attention has been paid to adaptive control in the development of self- driving car technology, leading to the development of fully autonomous vehicles. In [1], a survey stated that mistakes made by drivers cause many car crashes. Due to this, researchers are working harder to solve this problem using onboard sensing, computing, and control to assist drivers. Researchers and engineers are using adaptive cruise control systems to study vehicle dynamics, control algorithms, and collision avoidance strategies for improving intelligent transportation systems.

During the middle of the twentieth century, the first robots were developed. They have benefited from the development of various technologies, including mechanics, controls, computing, and electronics [2]. They help

researchers test theories and algorithms, and simulate real-world situations in a safe environment. Therefore, it is suitable for studies of robotics, artificial intelligence, and control systems, due to its ability to perform precise, repetitive, and independent tasks.

Several studies were conducted on mobile vehicles to advance intelligent transportation systems. Also, mobile robots and autonomous vehicles have been widely used as experimental testbeds in control systems research. Autonomous vehicles can change how we travel, so making them safer, more efficient, and more convenient is developers' goal. Self-driving cars must perform important safety tasks, like motion planning in dynamic environments shared with other vehicles and people, as mentioned in [3].

For example, research by [4] explored the integration of autonomous mobile robots (AMRs) as assistive vehicles in healthcare and logistics, where their ability to navigate autonomously enhances efficiency, reduces costs, and allows healthcare staff to focus more on patient care. Robotic platforms have played an important role in advancing artificial intelligence (AI) and machine learning (ML) techniques, especially in autonomous driving and decision-making systems. To improve deep learning-based (DRL) path planning for mobile robots, another study in [5] presents a step-by-step training method. Before transferring the algorithm to more complex 3D environments, the proposed approach first develops it in a simple 2D environment. Also, [6] implemented the ACC system on a robotic vehicle using an FPGA-based architecture. It compares fuzzy PID, Model Predictive Control (MPC), and conventional PID control methods through simulations and real-world tests, demonstrating effective performance and identifying practical challenges.

Other studies used multiple robots in their experiments, such as the work of [7]. This paper examines the implementation of collaborative and industrial robots in manufacturing, addressing labor cost challenges through mathematical management models. It proposes a creative cost-assessment framework to optimize profitability, energy use, and cost evaluation applications. Moreover, multiple robot cars were used in [8] to examine an adaptive cruise control algorithm for vehicle platooning using smart car platforms as mobile robots. The system maintains a predetermined gap using laser sensors without inter-vehicle communication, thus demonstrating effectiveness through simulations and experiments.

Autonomous wheeled robots were used in [9] to present an adaptive cruise control algorithm. It makes the robot follow an object before it, enabling smooth and safe movement by maintaining distance from obstacles. The system is implemented in Java using an Android smartphone and an IOIO (PIC microcontroller-based boards that allow Android mobile applications to interact with external electronics) interface. Another study [10] utilizes ultrasonic sensors to maintain a safe following distance. It implements super twisting sliding mode control (STSMC) and non-singular terminal sliding mode control (NTSMC) with neural networks to handle uncertainties. The technique, which has been tested on a prototype with Arduino and DC motors, shows high efficiency and robustness, enhancing safety and performance in autonomous driving.

Model predictive control is a widely used optimization-based control strategy in robotics and autonomous systems because it can optimize performance over a given time horizon [11]. Its predictive capability makes it well-suited for applications requiring precise trajectory tracking, obstacle avoidance, and adaptive decision-making.

MPC operates by solving an optimization problem at each time step, where it predicts the future states of a system based on a dynamic model. It selects control inputs that optimize a predefined cost function while satisfying constraints related to the system's physical limits. However, after computing the optimal control sequence, only the first control action is applied, and the process is repeated at each step, allowing the controller to adapt to disturbances and uncertainties in real time [12]. For instance, [13], this paper presents a teaching approach on robotic vehicles (implemented in LabVIEW and deployed on an FPGA-based platform). It demonstrates MPC's effectiveness in velocity and distance tracking, bridging theory and real-world performance through hands-on learning. A recent study by Zhao et al. (2025) integrated MPC with an improved Artificial Potential Field (APF) to achieve reliable obstacle avoidance and trajectory tracking for autonomous robotic vehicles. Their findings confirm MPC's strength in balancing safety, comfort, and tracking accuracy—objectives that align with the present ACC study. In addition, Ali and Ul Hassan (2022) demonstrated the effectiveness of model-based motion control on an industrial 7-DOF robotic manipulator, validating various trajectory-tracking strategies through real experiments.

The TurtleBot3 is used for learning and as a test model for a robot delivery project. A study in [14] involved designing a TurtleBot3 that might operate in manual and automatic modes. The study showed that this method for avoiding obstacles enables TurtleBot to reach its target points safely.

This study utilizes TurtleBot3 as a prototype to implement and validate an ACC controller based on MPC. Unlike previous studies, this work experimentally demonstrates the real-time interaction between MATLAB-

based optimization and TurtleBot3 hardware through LiDAR data filtering and velocity control. It offers insights into real-time control efficacy, sensor integration difficulties, and system adaptability across diverse situations. The novelty lies in the integration of a dual-level architecture, real hardware validation, and quantitative performance comparison with a PID controller.

2. Materials and methods

In the past, obtaining robotic platforms was expensive and required many technical skills from educators. Recent versions fix these problems and provide more chances for hands-on learning. TurtleBot3 is a cheap robot with advanced sensors and an excellent choice for testing and developing adaptive cruise control systems in real-time situations [15]. Its compact size, cost-effectiveness, and modular design make it appropriate for robotics research and autonomous navigation tests.

TurtleBot is the standard Robot Operating System (ROS) platform, the most often used hardware in research, especially in motion planning methodologies. For use in robotics, training, and study, the TurtleBot3 robot is easily adaptable to specific demands and is programmable in MATLAB and Python. The 360-degree LiDAR sensor makes this robot ideal for typical SLAM motion planning applications. [16, 17].

Researchers can use ROS tools to study sensor data and control algorithms. TurtleBot3 allows for real-time communication and control, making it easy to evaluate how effectively ACC systems operate in changing situations. Utilizing TurtleBot3 enables researchers to assess control strategies and optimize algorithms before scaling to larger autonomous vehicles, minimizing development costs and risks.

2.1. Turtlebot3 Hardware

The TurtleBot3 Burger robot is equipped with several essential hardware components that enable its functionality and support advanced control algorithms [16]:

2.1.1. LiDAR Sensor

The 360-degree LDS-01 LiDAR sensor (at the top of the robot) collects distance data from 120 mm to 3,500 mm. It features a 1-degree angular resolution and operates at a sampling rate of 1.8 kHz. The sensor supports USB connections for computers and UART interfaces for embedded systems, making it ideal for real-time mapping and obstacle detection.

2.1.2. Raspberry Pi Development Board

The Raspberry Pi 3 Model B (below the LiDAR sensor) acts as the primary processing unit. While it has moderate computational power, it is well-suited for implementing autonomous algorithms, such as machine learning and digital image processing.

2.1.3. OpenCR Control Board

The OpenCL board, located beneath the Raspberry Pi, functions as the robot's principal hardware controller. The system is built on an STM32F7 microcontroller with an ARM Cortex-M7 core with floating-point capabilities. Furthermore, it encompasses an inertial measurement unit (IMU) consisting of a tri-axial accelerometer, gyroscope, and magnetometer, which enables sensor fusion and motor control.

2.1.4. Additional Components

Dynamixel XL430-W250 motors provide differential drive capabilities and an 11.1 V lithium-polymer (LiPo) battery, ensuring a reliable power supply and mobility. See Figure 1.

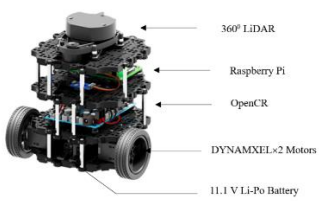

Figure 1. TurtleBot3 Burger.

2.2. ROS

The robot operating system is a framework for studying and researching robotics. It was originally developed by Willow Garage, a robotics research lab based in Menlo Park, California. The initial release of ROS was made in 2007, and it quickly became a widely adopted framework for robotics research and development. After Willow Garage shut down in 2014, the development and maintenance of ROS were transferred to the Open Source Robotics Foundation (OSRF), later renamed the Open Robotics organization [18]. Open Robotics has been responsible for the continued development and maintenance of ROS and its successor, ROS 2.
ROS provides conventional operating system features such as hardware abstraction, low-level device management, implementation of commonly used functionality, inter-process message forwarding, and package maintenance. Linux distributions like Ubuntu, Fedora, and Arch use it since it is BSD-licensed for Linux platforms [16].

2.3. Robot Software

The Raspberry Pi board requires an operating system compatible with its processor. A Linux-based distribution is typically chosen due to its flexibility, development advantages, and cost-effectiveness. For the laboratory experiments conducted in this research, Ubuntu MATE 16.04 was used.

2.4. Network Configuration and Communication Setup

This experiment establishes a connection (through a local Wi-Fi network) between the Raspberry Pi on the TurtleBot3 robot and the external computer. A static IP address was assigned to the Raspberry Pi to ensure stable and consistent communication. Both devices operated within the same network, enabling seamless interaction via the ROS framework.
The external computer (ROS Master) handles node communication. Raspberry Pi (Client) handles sensor data collecting and motor control commands. The system utilized ROS topics such as (/cmd_vel) for publishing velocity commands and (/scan) for receiving LiDAR data, enabling continuous observation of the robot's environment and dynamic obstacle detection. See Figure 2.

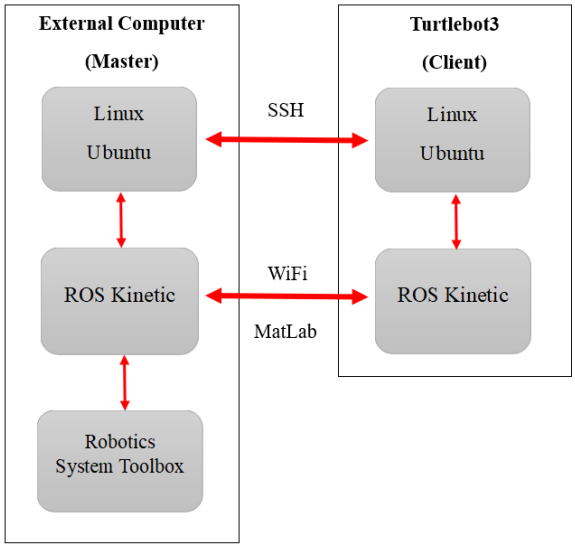

Figure 2. Turtlebot3 Software Architecture.

Remote access to the Raspberry Pi was established using SSH (Secure Shell) to control the robot. MATLAB was used with ROS to implement and develop the model predictive control algorithm. The algorithm dynamically adjusted speed and safe distances based on filtered LiDAR data, ensuring reliable obstacle avoidance and emergency stop. MATLAB for the MPC algorithm and ROS Kinetic for communication with TurtleBot3 were used.

3. Modelling and Control

3.1. Adaptive Cruise Control

The adaptive cruise control system is designed to automatically regulate the speed of a vehicle to keep a safe distance from a preceding vehicle. In this work, the ACC utilizes an MPC framework to compute the desired acceleration based on the relative distance, relative speed, and the vehicle dynamics. The control objectives are to ensure passenger comfort, improve traffic flow, and maintain safety by preventing collisions. The MPC controller predicts future states over a finite horizon and optimizes the control inputs while satisfying vehicle and safety constraints. It uses the LiDAR of the Turtlebot3 to measure relative distances. See Figure 3.

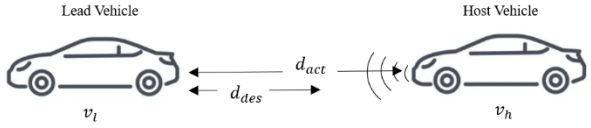

Figure 3. Two vehicles in an ACC Scenario.

A simplified longitudinal dynamics model for the host vehicle is implemented. Position and velocity update equations are described as follows:

$$x_h(t+1) = x_h(t) + v_h(t).\Delta t, \tag{1}$$

$$v_h(t+1) = v_h(t) + a_h(t).\Delta t \tag{2}$$

($x_h(t)$, $v_h(t)$, and $a_h(t)$) are the host vehicle's position, velocity, and acceleration at time t. Δt is the sampling time. The relative distance between the host vehicle and the lead vehicle is modeled as:

$$d_{act}(t+1) = d_{act}(t) + [v_l(t) - v_h(t)].\Delta t \tag{3}$$

($d_{act}(t)$) is the actual distance between the vehicles at time t. ($v_l(t)$) is the velocity of the lead vehicle at time t. A safety distance (d_{safe}) is defined to prevent collisions based on time-to-collision (TTC):

$$d_{safe}(t) = d_{min} + v_h(t).T_{react} \tag{4}$$

The (d_{min}) is the minimum stopping distance. (T_{react}) represents the time delay for the driver or control system to respond to a detected event, such as a braking action by the lead vehicle. It is usually 1–2 seconds for human drivers, but can be much shorter for automated systems. In this work, (d_{safe}) is taken as a constant value equal to 0.5m (corresponds to the minimum measured LiDAR detection accuracy and experimental safety margin based on TurtleBot3 dynamics). Another critical parameter in ACC is the time gap (T_{gap}) (also called time headway), which is defined as the time the host vehicle takes to reach the point where the leading vehicle is at its present speed (typically ranging from 1–3 seconds).

$$d_{gap}(t) = v_h(t).T_{gap} \tag{5}$$

The (T_{gap}) defines the steady-state following distance (d_{gap}) under normal driving conditions. In this work, to simulate the ACC scenario, the Turtlebot3 robot is used as the following vehicle, and a toy car, which the user manually controls, as the lead vehicle. Note relative speed (gap rate) $v_{rel}(t)$ is assumed to be a time-varying function. The relative speed can be calculated as:

$$v_{rel}(t) = [d_{act}(t) - d_{act}(t-1)]/\Delta t \tag{6}$$

Where the absolute lead-vehicle speed is not directly measured but is implicitly captured by the relative speed term. The TurtleBot3 robot receives online commands from the MPC controller running on MATLAB. See Figure 2.

3.2. TurtleBot3 Burger Control

Kinematics is the fundamental science of mechanical systems. The mechanical behaviour of mobile robots must be understood to control them [19]. Kinematic analysis in TurtleBot3 enables the determination of orientation

and position within an environment, independent of the forces that induced any changes; this information facilitates the identification of the device's current location and prospective trajectory [20]. In kinematic equations, x, y, and θ are the state variables while v and ω are control inputs. See Figure 4.

$$\dot{x} = v\cos(\theta)$$
$$\dot{y} = v\sin(\theta) \quad (7)$$
$$\dot{\theta} = \omega$$

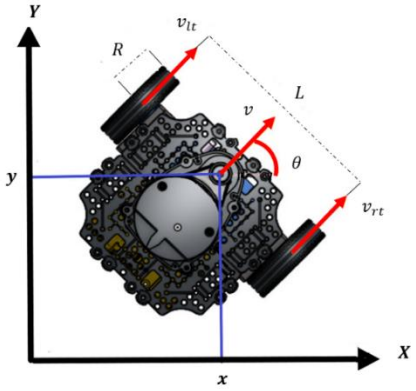

Figure 4. TurtleBot3's kinematics and coordinating system.

Where:

$$v = R\frac{v_{rt} + v_{lt}}{2}$$
$$\omega = \frac{R}{L}(v_{rt} - v_{lt}) \quad (8)$$

All parameter definitions are listed in Table 1 below.

Table 1. Kinematic Parameters' definitions.

Parameters	Definition
x	Position along the x-axis (m)
y	Position along the y-axis (m)
θ	Heading angle (rad)
v	Linear velocity (m/s)
ω	Angular velocity (rad/s)
v_{rt}	Right wheel velocity (m/s)
v_{lt}	Left wheel velocity (m/s)
R	Turtlebot3 wheel radius (0.033 m)
L	Turtlebot3Wheel base (0.16 m)

Odometry is calculated from encoder data to estimate the robot's pose:

$$\mathrm{x}(\mathrm{t}+\Delta\mathrm{t}) = \mathrm{x}(\mathrm{t}) + \mathrm{v}.\Delta\mathrm{t}.\cos\big(\theta(\mathrm{t})\big)$$
$$\mathrm{y}(\mathrm{t}+\Delta\mathrm{t}) = \mathrm{y}(\mathrm{t}) + \mathrm{v}.\Delta\mathrm{t}.\sin\big(\theta(\mathrm{t})\big) \quad (9)$$
$$\theta(\mathrm{t}+\Delta\mathrm{t}) = \theta(\mathrm{t}) + \omega.\Delta\mathrm{t}$$

The TurtleBot3 robot operates as a differential drive system, meaning it has two independently controlled wheels mounted on the same axis. The robot's motion is controlled by adjusting the linear velocity (v) and angular velocity (ω). For this application, only the linear velocity is optimized (lateral motion is eliminated)

since the movement is only in the forward direction (the orientation is never changing), and both wheels have the same velocity ($v_{rt} = v_{lt}$), which makes the angular velocity set to zero (See Figure 5). Thus: $\omega = 0$ and

$$\theta(t + \Delta t) = \theta(t)$$

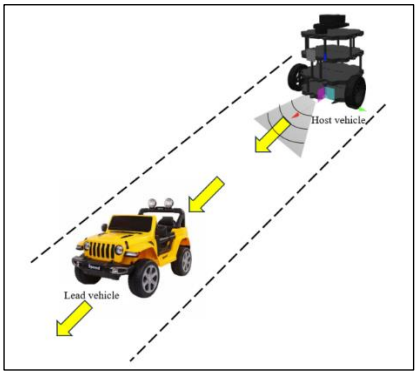

Figure 5. Lead vehicle and Turtlebot3 in an ACC Scenario.

3.3. Model Predictive Control Formulation for Solving the ACC Problem

Model predictive control is an optimization-based finite-time feedback control technique that iteratively solves a problem at every sampling instant to find control inputs [11]. It optimizes control inputs based on a mathematical dynamics model, subject to states and input constraints. The input constraints often represent physical limits. In these cases, the physical system enforces the input constraints if the controller does not adhere to them. In contrast, the output or state constraints are usually desirable. They may not be achievable depending on the disturbances affecting the system. The MPC controller often operates to verify in real-time that the output or state constraints are unattainable and to relax them acceptably. These considerations lead implementers of MPC to set up optimization problems, often using hard constraints for input constraints and some form of soft constraints for output or state constraints [12].

In cruise control (CC) mode, TurtleBot3 must maintain a fixed speed (v_{cruise}) as long as no vehicle is ahead. However, it switched to ACC mode when it detected a vehicle within its range. The first goal is to control the linear velocity (v) of the TurtleBot3 robot so that it maintains a desired following distance (d_{gap}) from the leading vehicle. A second goal is to adjust its velocity smoothly to prevent sudden acceleration or deceleration, ensure passenger comfort, and improve system efficiency. Safety constraints must be considered, too, such as limiting acceleration and maintaining a minimum safe distance. As a result, these goals are formulated as two types of constraints:

Hard constraint: This constraint prevents collisions between the two vehicles. It can be implemented by making the minimum distance between them equal to or greater than a predefined space (d_{safe}). Thus:

$$d(k) \geq d_{safe}$$

Soft constraint: This constraint prevents sudden jerks, thus making the ride more comfortable. It can be implemented by limiting the maximum and minimum acceleration levels to a specified value:

$$-a_{min} \leq a(k) \leq a_{max}$$

An additional constraint is added to restrict the velocity applied to the Trutlebot3's maximum transitional velocity.

$$0 \leq v(k) \leq v_{max}$$

The MPC problem is formulated as a Quadratic Programming (QP) problem because it minimizes a quadratic cost function (J) subject to linear inequality constraints on states and inputs:

$$J = \sum_{k=1}^{N} \left[w_1 \left(\frac{v(k) - v_{target}}{v_{max}} \right)^2 + w_2 \left(\frac{d(k) - d_{gap}}{d_{gap}} \right)^2 \right]$$
$$s.t$$
$$0 \leq v(k) \leq v_{max}$$
$$-a_{min} \leq a(k) \leq a_{max}$$
$$d(k) \geq d_{safe} \quad (10)$$

Where $(v(k))$ and $(d(k))$ are the robot's speed and relative distance at the time step (k), respectively. (w_1) and (w_2) are weights for speed error and distance error, respectively. (d_{gap}) is the desired distance between the two vehicles, and (v_{target}) is the desired velocity. The desired velocity is adjusted according to cruise speed (v_{cruise}) and the distance of the preceding car. While (d_{gap}) is dynamically adjusted according to the robot's current speed, as in Eq. (5). This adjustment adapts to varying drive conditions and scenarios, such as sudden speed and distance changes. In addition, it maintains a suitable gap space without unnecessary acceleration and deceleration, which leads to smoother motion and lower energy consumption.

The control law is updated at each sampling period in the MPC framework; however, the system stability is ensured by the properly designed cost function and constraint sets. The cost function includes terminal constraints and penalizing state deviations; therefore, the MPC guarantees closed-loop stability under the proposed control strategy, even with the time-varying nature of the control law.

At each control step, the optimization problem is solved over a finite prediction horizon (N) to compute the optimal sequence of accelerations (u_{opt}). However, only the first input in this sequence is applied, and the process repeats at the next time step. This receding horizon strategy allows the controller to adapt continuously to environmental changes. The block diagram of the implemented system is shown in Figure 6.

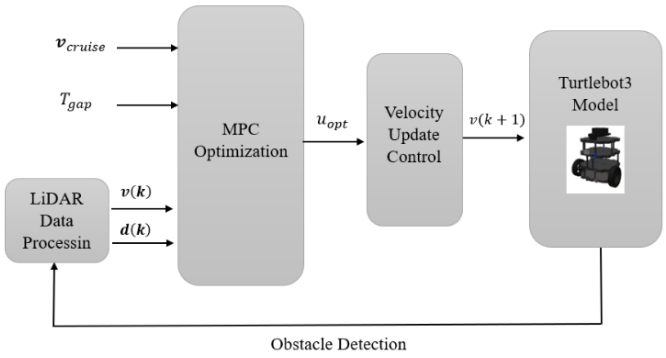

Figure 6. Block Diagram of ACC-based MPC System.

All parameters setting used in this paper are listed in Table 2.

Table 2. Parameters used in the experiment test.

Parameter	Value
d_{safe}	0.5 m
a_{max}	0.5 m/s^2
a_{min}	-0.5 m/s^2
w_1	2
w_2	3
v_{max}	0.22 m/s
v_{cruise}	0.2 m/s
T_{gap}	2 s
N	10

The sampling time was 0.1 seconds, and the closed-loop update rate was equal to 10Hz. For real time solving purpose, the prediction horizon was chosen N=10, resulting in a horizon duration equals 1 second. The quadratic programming problem was solved using MATLAB's (quadprog), with a function tolerance of 10^{-4}.

3.4. Turtlebot3 Data Filtering

A method to filter unnecessary measured data by the LiDAR sensor is introduced to enhance the robot's response time and overall operation efficiency. Since LiDAR provides a large amount of raw data (3600 view readings) and according to the ACC function, narrow angles of these readings are necessary for detecting the preceding vehicle. A method for focusing on the relevant objects within its path is proposed, as shown in Figure

7. Both angles and distance are reduced from the measured data. Thus, a filtering process is applied to the raw LiDAR measurements. The filtered signals are then used as inputs to the ACC controller. Raw scans were filtered by narrowing the field of view to (±15°) and setting a distance limit. This allows the system to ignore "noise data" that is either too far away or irrelevant to the path ahead. The lead target was selected as the closest object in that zone, which is a standard and effective strategy for the ACC system. The analysis is described below.

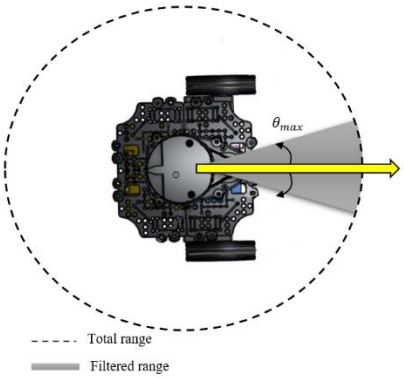

Figure 7. TurtleBot3's total and filtered range.

Let's define the LiDAR data set by:

$$D = \{d_1, d_2, \dots, d_N\}$$
$$\Theta = \{\theta_1, \theta_2, \dots, \theta_N\}$$

Where D and Θ are the sets of distances and angles for the obstacles around the robot, with d_i as the distance to an obstacle at an angle θ_i . So, the LiDAR data can be represented as:

$$\mathcal{L} = \{(d_i, \theta_i) \mid i = 1,2, \dots, N\}$$

The range for angles collected by the sensor is filtered out to restrict the view in front of the robot to a specific limit (θ_{max})and obtained as follows:

$$\mathcal{F}_\Theta = \{(d_i, \theta_i) \mid |\theta_i| \leq \theta_{max}\}$$

Next, the range for distance was also filtered out to a limited distance (d_{max}), Therefore, the ranges become:

$$\mathcal{F}_\mathrm{D} = \{(d_i, \theta_i) \in \mathcal{F}_\Theta \mid d_i \leq d_{max}\}$$

Where $\theta_{max} = \frac{\pi}{12}\ rad$ and $d_{max} = 3\ m$.

The entire filtering process can be written as a composition of sets:

$$\mathcal{L}_\mathcal{F} = \mathcal{F}_\mathrm{D} \circ \mathcal{F}_\Theta(\mathcal{L})$$

Or:

$$\mathcal{L}_\mathcal{F} = \{(d_i, \theta_i) \mid |\theta_i| \leq \theta_{max}, d_i \leq d_{max}\}$$

The final filtered set can be written as:

$$D_{filter} = \{d_i \mid (d_i, \theta_i) \in \mathcal{L}_\mathcal{F}\}$$
$$\Theta_{filter} = \{\theta_i \mid (d_i, \theta_i) \in \mathcal{L}_\mathcal{F}\}$$

This filtering method reduces the workload of the LiDAR input data processed by the ACC controller and lowers the computational effort on the TurtleBot3 platform. Furthermore, this process enables regular real-time execution within the control loop. In all testing scenarios, although no explicit object tracking or clustering was implemented, efficient lead-vehicle detection and stable closed-loop behavior were observed.

3.5. Controller Realization

Three task requirements are needed: sensing, computing, and actuation. In the sensing phase, the distance of the vehicle ahead measured by the LiDAR sensor is fed into the MPC controller. In addition, the relative speed of the car ahead is calculated using Eq. (6).

During the computing phase, a dual-level controller is present. The upper-level controller is a logical entity that shows the robot's mode of operation. It switches between four modes based on the LiDAR data and predefined conditions, as listed in Table 3 and Figure 8.

Table 3. ACC modes description.

State	Condition
Cruise Mode (CM)	A vehicle ahead too far: $d(k) \gg d_{safe}$
Follow Mode (FM)	A vehicle ahead detected: $d(k) \geq d_{safe} + \delta$
Emergency Stop (ES)	A vehicle ahead too close: $d(k) < d_{safe}$
Resume Moving (RM)	A vehicle ahead starts moving: $d(k) \geq d_{safe}$

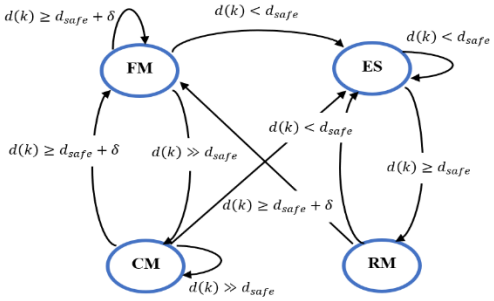

Figure 8. State diagram of the Mode-switch controller.

The δ is an additional allowance added to avoid overly abrupt changes in behaviour and compensate for the uncertainty that may occur due to sensor noise, sudden changes in speed, and slow response by the robot. Next, the lower-level MPC controller is implemented, and the calculated optimal acceleration is used to find the optimal velocity for the robot. Finally, based on the robot's current mode and the MPC optimal acceleration found, the linear velocity is computed and sent to the motors via the /cmd_vel topic using ROS in the actuation phase.

4. Results and Discussions

Several test scenarios were implemented in this experiment to validate the controller. In the first scenario, the ACC vehicle (Turtlebot3) starts from rest and accelerates to a constant cruise speed of 0.22 m/s. A preceding (leader) vehicle is initially stationary. The algorithm's control law is deemed effective if the robot reaches the specified gap distance without hitting with the car ahead. In Fig. 9.a, the gap distance starts from 2.1 m and ends at 0.5 m, the intended safe distance. Fig. 9. b shows the follower's velocity profile (robot), which speeds up from its initial speed of zero to 0.22 m/s, then slow down as it encounters a stopped vehicle ahead. Fig. 9.d shows the optimal acceleration found by the algorithm.

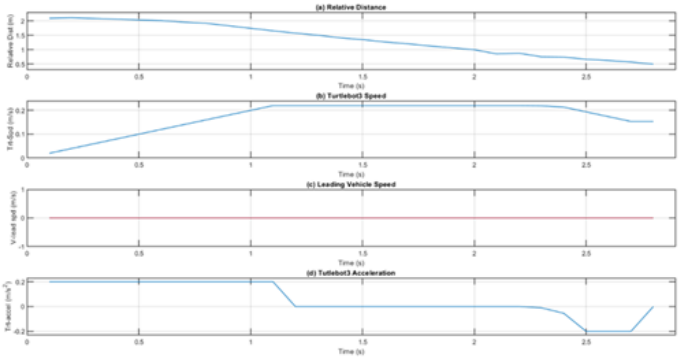

Figure 9. Scenario 1 with a halted vehicle,

In the second scenario, the follower vehicle (robot) facing a leader vehicle with varying speeds. The test was started as the follower starts from rest (0 m/s), while the leader speed up to a maximum speed and subsequently slow down to a stop. Within one second, the leader accelerates at high speed, increasing the gap distance between it and the follower (turtlebot3) to approximately 1.4 m, and then it stops, as shown in Figure 10 (a and c). The follower acts by speeding up to reduce the gap distance, as in Figure 10. (b). Figure 10. (d) shows that control input from the MPC controller initially commands the follower to accelerate to its maximum limit. Since the gap between the two vehicles decreases, a control action commands the host vehicle to decelerate until it reaches the specified gap equal to 0.5 m and stops immediately. The Turtlebot3 manoeuvres the target vehicle successfully without collision. Four successive stops occur during this scenario (at 1st, 3rd, 5th, and 8th seconds). Each time, the MPC controller succeeded in commanding the robot to maintain a safe distance (0.5 m) and avoid collision with the target vehicle.

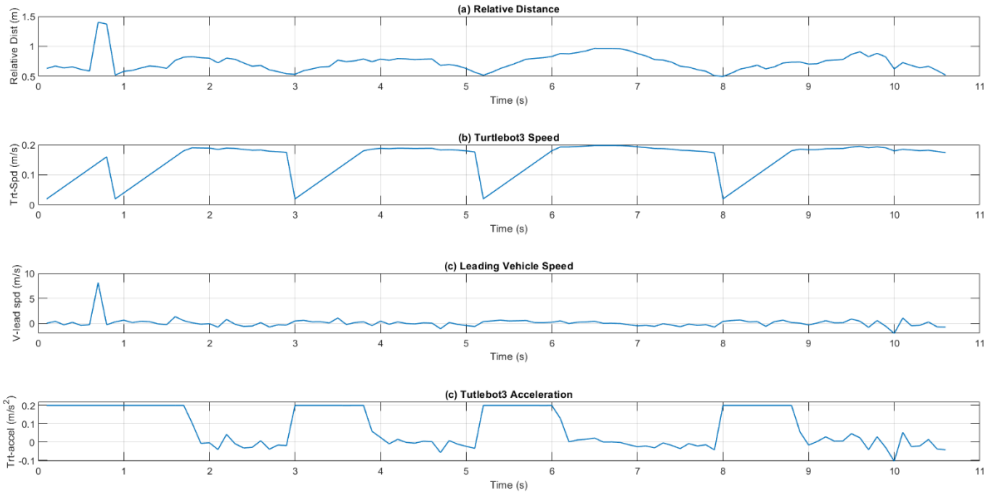

Figure 10. Scenario 2 with variable leading vehicle speed.

In the third scenario, the follower ACC vehicle (TurtleBot3) encounters a leader vehicle moving within a specified speed range. The follower maintains a gap of less than 1 m with several stops, as illustrated in Fig. 11(a–c). Again, the controller managed the follower's acceleration (as in Figure 11. d) to preserve a safe distance and prevent collision. Figure 11. (e) also shows the mode selection for the upper-level controller during the driving cycle.

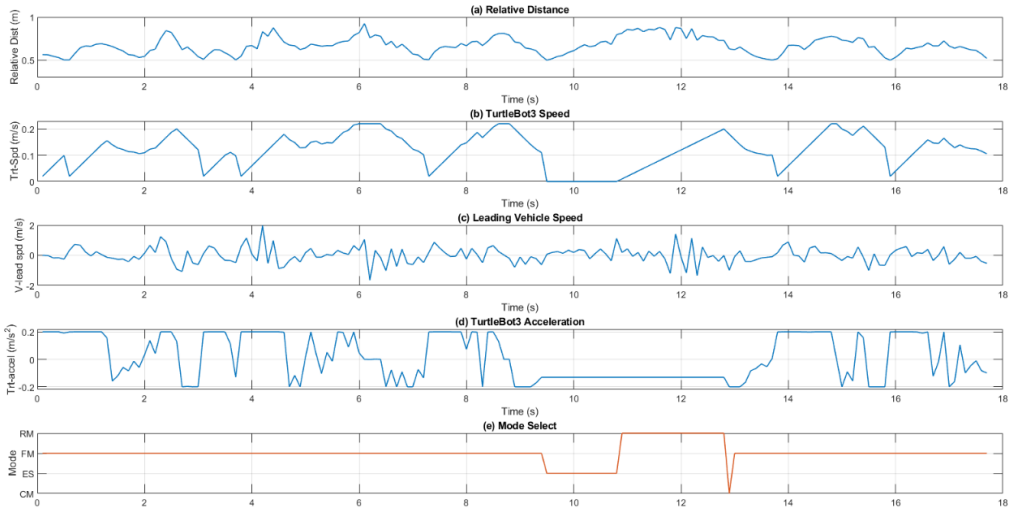

Figure 11: Scenario 3 with limited variable leading vehicle speed

A final test is done using MPC and PID controllers separately to compare the performance and effectiveness of both methods. The Stop-and-Go scenario is conducted with frequent stops and starts, as shown in Figure 12. Two tests under identical conditions were implemented, and several parameters were recorded, such as distance tracking error (Dis_err), response time to changes (T_{resp}), and comfort metric or jerk ((Jrk)). Jerk is the rate of change of acceleration and is an essential metric for evaluating passenger comfort.
Response time is the time the controller takes to react to changes in the speed of the target car or environmental conditions. Table 4 shows the results obtained:

Table 4. The two controllers' results

Parameter	PID	MPC
Dis_err_{max} (m)	2.301	1.557
T_{resp} (s)	1.4	4.4
Jrk_{mean} (m/s^3)	0.011494	0.384988

To complement the qualitative observations, quantitative performance indicators were evaluated for both controllers under identical stop-and-go scenarios. The metrics include rise time, settling time, and the root-mean-square error (RMSE) of the distance tracking signal. These values were computed from the recorded experimental data using MATLAB (Table 4). The results are summarized in Table 5.

Table 5. Quantitative comparison of PID and MPC controllers

Performance Metric	PID Controller	MPC Controller	Difference (MPC − PID)	Comment
Max distance tracking error (m)	2.301	1.557	−0.744 (−32.3 %)	MPC smaller
Response (rise) time (s)	1.4	4.4	+3.0 (+214 %)	MPC slower
Settling time (s)	4.6	3.3	−1.3 (−28 %)	MPC settles faster
RMSE of distance (m)	0.67	0.44	−0.23 (−34 %)	MPC lower error
Mean jerk (m/s^3)	0.0115	0.3849	+0.373	MPC more aggressive

The MPC controller achieved a lower maximum tracking error, confirming its better ability to handle large deviations and external disturbances compared with the PID controller. Although the PID presented a shorter response time, it did so at the cost of stability, often producing overshoot during acceleration or deceleration transitions. In contrast, the MPC emphasized smooth and controlled behavior by predicting future states and optimizing its control inputs over a finite prediction horizon. This optimization process increases computation time, which explains the longer response time observed in Table 5. Nevertheless, the MPC achieved smaller steady-state and root-mean-square errors, demonstrating more robust distance tracking. While the PID provided smoother average control (lower mean jerk) and faster reaction, the MPC offered a better overall balance between tracking accuracy and stability, which is crucial for safe adaptive cruise control applications.

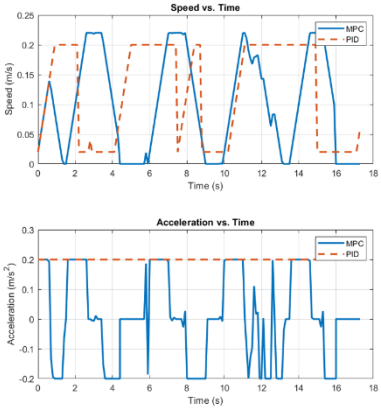

Figure 12: Scenario 4 stop-and-go for MPC and PID schemes.

5. Conclusions

The experimental results demonstrate the effectiveness of the proposed two-level controller using the model predictive control algorithm in managing the adaptive cruise control system for the TurtleBot3 across various scenarios, ensuring both safety and reliability. For the first scenario, the follower ACC vehicle (TurtleBot3) detected a stationary vehicle and executed a smooth deceleration to maintain the specified safe gap of 0.5 m without collision. In the second scenario, the follower responded effectively to the dynamically changing speed of the leader vehicle. The leader accelerated to reduce the increasing gap and decelerated promptly when the target vehicle stopped, maintaining the safe gap distance. The system demonstrated precise control during successive stops, successfully preventing collisions while ensuring smooth transitions between the acceleration and deceleration phases.

In the third scenario, the leader vehicle moved within a certain speed range with several stops. Also, the controller maintained its performance under changing conditions. The controller managed the robot's acceleration to keep the desired safe distance, avoiding collisions. Mode selection during the driving cycle further shows the controller's adaptability to varying traffic situations.

In the final scenario, where a comparison is made between the MPC and PID methods, the results show that MPC provides better performance in outlier scenarios. In contrast, PID is quicker, but its faster response can lead to less precise control in varying scenarios. The MPC performs better when robustness to disturbances or extreme cases is essential. The PID has good performance if the response is the main priority, especially when large deviations are less frequent.

The proposed MPC makes sure that control actions are bounded and that they are still possible within the limits that have been set. The closed-loop system shows stable and smooth behavior in all tested scenarios, as shown by the simulation results.

Declaration of Competing Interest

The authors declare that they have no known competing of interest.

Funding Information

No funding was received from any financial organization to conduct this research.

Author Contributions

Methodology; investigation; data curation; writing—original draft preparation; formal analysis, Farah Mahdi Ali , and review and editing; supervision; project administration Nizar Hadi Abbas; , All authors have read and agreed to the published version of the manuscript.

Acknowledgments

The authors express their gratitude to the Electrical Engineering Department, College of Engineering, University of Baghdad, Baghdad, Iraq for supporting this study.

References

[1] A. Mehra, W. Ma, F. Berg, P. Tabuada, J. W. Grizzle, and A. D. Ames, "Adaptive cruise control: experimental validation of advanced controllers on scale-model cars," in *Proc. IEEE Amer. Control Conf. (ACC)*, Chicago, IL, USA, 2015, pp. 1411–1418, doi: 10.1109/ACC.2015.7170931.

[2] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*, 2nd ed., Springer, 2016.

[3] B. Paden, M. Cap, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Trans. Intell. Veh.*, vol. 1, no. 1, pp. 33–55, 2016.

[4] G. Fragapane, H.-H. Hvolby, F. Sgarbossa, and J. O. Strandhagen, "Autonomous mobile robots in hospital logistics," in *Proc. Springer VS*, 2020, pp. 672–679. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-57993-7_76.

[5] J. Gao, W. Ye, J. Guo, and Z. Li, “Deep reinforcement learning for indoor mobile robot path planning,” *Sensors*, vol. 20, 2020, Art. no. 5493, doi: 10.3390/s20195493.
[6] P. Shakouri, A. Ordys, and G. Collier, “Robotic implementation of the adaptive cruise control: Comparison of three control methods,” in *Mechatronics 2013*, T. Březina and R. Jabloński, Eds. Cham: Springer, 2013. doi: 10.1007/978-3-319-02294-9_80.
[7] A. M. Țîțu, V. Gusan, M. Dragomir, A. B. Pop, and Ș. Țîțu, “Cost calculation and deployment strategies for collaborative robots in production lines: An innovative and sustainable perspective in knowledge-based organizations,” *Sustainability*, vol. 16, 2024, Art. no. 5292, doi: 10.3390/su16135292.
[8] D. L. Luu, C. Lupu, and C. Petrescu, “Simulation and experimentation of adaptive cruise control for robot platooning,” in *Proc. 24th Int. Conf. Syst. Theory, Control Comput. (ICSTCC)*, Sinaia, Romania, 2020, pp. 721–726, doi: 10.1109/ICSTCC50638.2020.9259781.
[9] M. A. Gunawan, F. Sulaiman, T. A. Putra, C. D. Hasudungan, and R. F. Sari, “Object-following robot using adaptive cruise control algorithm with IOIO,” in *Proc. Int. Conf. Intell. Green Building Smart Grid (IGBSG)*, 2014. IEEE, doi: 10.1109/IGBSG.2014.6835251.
[10] R. Alika, E. M. Mellouli, and E. H. Tissir, “Adaptive cruise control of the autonomous vehicle based on sliding mode controller using Arduino and ultrasonic sensor,” *J. Robot. Control (JRC)*, vol. 5, no. 1, 2024, doi: 10.18196/jrc.v5i1.20519.
[11] E. F. Camacho and C. Bordons, *Model Predictive Control*, 2nd ed., London: Springer-Verlag, 2007.
[12] J. B. Rawlings and D. Q. Mayne, *Model Predictive Control: Theory and Design*, Nob Hill Publishing, 2009.
[13] P. Shakouri, A. Ordys, and G. Collier, “Teaching model predictive control algorithm using starter kit robot,” *Eng. Educ.*, vol. 8, no. 2, pp. 30–43, 2013, doi: 10.11120/ened.2013.00018.
Zhao, Lei, Hongda Liu, and Wentie Niu. 2025. "Collision avoidance of driving robotic vehicles based on model predictive control with improved APF" Machines 13, no. 8: 696. doi:10.3390/machines13080696.
Ali, S. H., & Ul Hassan, . E. (2022). Implementation and Analysis of Motion Control Techniques on an Industrial 7 DOF Robotic Manipulator. Journal of Studies in Science and Engineering, 2(2), 33–43. https://doi.org/10.53898/josse2022223.
[14] H. Soebhakti, R. Yulianti, F. A. Risi, and Y. R. Pratiwi, “Obstacle avoidance system using LiDAR on robot Turtle-bot3 Burger,” *Proc. ICAE*, EAI, 2023, doi: 10.4108/eai.5-10-2022.
[15] A. Robin and S. Peter, “Turtlebot3 as a robotics education platform,” in *Robotics in Education (RiE 2019)*, M. Merdan, W. Lepuschitz, G. Koppensteiner, R. Balogh, and D. Obdržálek, Eds. Cham: Springer, 2019, vol. 1023, doi: 10.1007/978-3-030-26945-6_16.
[16] F. H. Martínez, “TurtleBot3 robot operation for navigation applications using ROS,” *Tekhnê*, vol. 18, no. 2, pp. 19–24, 2021.
[17] J. Abdulsaheb and D. Kadhim, “Real-time SLAM mobile robot and navigation based on cloud-based implementa-tion,” *J. Robot. *, 2023, doi: 10.1155/2023/9967236.
[18] Open Robotics. [Online]. Available: https://www.openrobotics.org.
[19] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*, Cambridge, MA, USA: MIT Press, 2011.
[20] J. Escobar-Naranjo, G. Caiza, P. Ayala, E. Jordan, C. A. Garcia, and M. V. Garcia, “Autonomous navigation of robots: Optimization with DQN,” *Appl. Sci.*, vol. 13, 2023, Art. no. 7202, doi: 10.3390/app13127202.