International Journal of Electrical Engineering and Ethics- Volume 6 Issue 2, March- 2023

RESEARCH ARTICLE OPEN ACCESS

Stabilization and Motion Control of Two-Wheeled Self-Balancing Mobile Robot Using ROS2 and Gazebo Simulator

Nedal ALqerem[1], Moayed ALmobaied[2]

1(*Electrical Engineering Department, Islamic University of Gaza/ Engineering College, Palestine*
Email: nalqerem@students.iugaza.edu.ps)

2 (*Electrical Engineering Department, Islamic University of Gaza/ Engineering College, Palestine*
Email: malmobaied@iugaza.edu.ps)

Abstract:

Nowadays, the use of robots and intelligent machines is growing exponentially, not only in terms of the number of possible applications but also in their complexity. These technologies have proven effective in solving problems and improving solutions in various industries, including manufacturing processes and transportation. The Robot Operating System (ROS) is one of the most widely used tools for robot programming. ROS1 was initially developed in 2007 and has since gained popularity among the open-source robotics community, particularly for academic projects. However, it is not yet widely used in industry because it lacks some of the most critical requirements, including real-time capabilities, safety features, and security measures. A new version of ROS, ROS2, was released in 2017 to address the limitations of ROS1 in terms of security and reliability. ROS2 offers quality-of-service improvements, support for embedded systems, and real-time capabilities. ROS2 was developed with the goal of being used in industrial projects and made compatible with industrial applications. As more sensors, actuators, and controllers are added to a robot, programming it with custom code becomes increasingly complex and difficult to manage. ROS2 offers a solution to this problem by providing a powerful robot application that can be used to create software with additional communication between subprograms. The primary goal of this paper is to explore the control features and tools provided by ROS2 and the Gazebo Simulator, which is commonly used in conjunction with ROS2. In this study, a self-balancing robot will be used as the robotic application system. It is a nonlinear, underactuated system with a single input and multiple outputs. Firstly, the well-known PID and LQR controllers will be implemented and programmed with ROS2. Then, the proposed controllers are used to stabilize the system and visualize the result in a real-time implementation using the Gazebo Simulator.

Keywords — **PID, LQR, ROS1, ROS2, SDF, Gazebo, RQT.**

I. INTRODUCTION

The two-wheeled self-balancing mobile robot (TWSBMR) is a nonlinear system consisting of a two-wheeled chassis and an inverted pendulum body. It is an interesting underactuated system that enables three degrees of freedom of motion (pitch, yaw, and straight movement) with only two-wheel rotations [1].The problem of balancing a two-wheel self-balancing mobile robot has been extensively researched and is often used as a benchmark for evaluating the performance of control systems. A two-wheeled self-balancing mobile robot (TWSBMR) can exhibit both linear and nonlinear behavior, depending on the desired motions such as straight-line motion or turning, and the control algorithms used to achieve these motions. Over the past fifteen years, the control of TWSBMRs has received significant attention due to their real-time applications in various fields, including launching rocket missiles, operating the popular Segway transporter, balancing bicycles, designing humanoid robots, and many other applications [2]. The Robot Operating System 2 (ROS2) is a collection of open-source frameworks, tools, and libraries used for the research and development of modern robots. It is widely used as middleware for robots in both academic and industrial settings [3, 4], and it is popular in the robotics community due to its design for distributed real-time systems. Considered the most powerful tool for developing modern robotic software, ROS2 serves as a standard for robotic applications that can be used on any robot [5, 6]. Every robot platform's software stack requires a wide range of components, including communication, architecture, networking modules, hardware drivers, and robot algorithms. Instead of developing these tools from scratch, developers can access them all in one place through ROS2. Fig.1 provides a summary of the features of ROS2, highlighting the key characteristics and capabilities of this powerful robotic software platform.

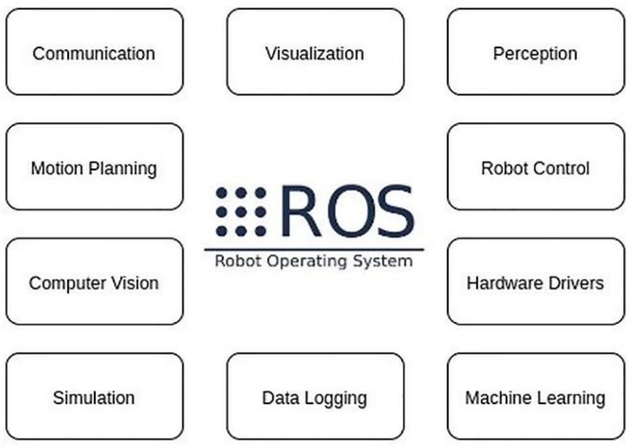


Fig.1 The ROS2 Content.

Although a large robotics community has contributed new features to ROS1 since its start in 2007 [7], the design and performance limitations of ROS1 led to the development of ROS2, which effectively addresses these issues. The architecture and features of ROS1 no longer satisfy the rapidly evolving requirements of the robotics industry today due to various issues, such as its limitation to a single robot system and single platform, poor real-time performance and stability, high network requirements, and a lack of sufficient confidentiality. Consequently, ROS1 is increasingly failing to meet the complex task requirements demanded by robots in current environments. While ROS 2 was established with an improved design that is suitable for use in both industry and academia, it adds additional features including scalability, security, performance, and an enhanced communication stack that uses real-time data. This is particularly useful for distributed systems with many robots and sensors, and it relies on the Data Distribution Service (DDS) protocol [8]. ROS 2 also addresses issues related to non-ideal networks where message delivery may be delayed or insecure. Table I provides a comparison between ROS1 and ROS2 [9].

TABLE I
THE COMPARISONS BETWEEN ROS1 AND ROS2.

ROS1	ROS2
Uses TCP/IP for communication	Uses the DDS communication protocol
CMake is the only build system used by ROS1.	Various build systems are available for use with ROS2.
Central registration and discovery are performed by ROS1 Master. If the master fails, the information transmission channel is subject to failing.	Use the distributed discovery system (DDS). A special API is available in ROS2 to get data on nodes and topics.
ROS1 is only available for the Ubuntu operating system.	Ubuntu, Windows 10, and OS X all operate with ROS2.
Use Python 2 and C++ 03	Use Python 3 and C++11

There are several simulation tools integrated with ROS2, and the Gazebo Robot Simulator will be used in this paper. The simulator was utilized to construct the self-balancing robot from scratch and generate SDF format files [10] that accurately describe all the robot's characteristics and parts, including sensors, surface properties, joint friction, and other relevant properties for a robot. These SDF format files can be used for robot control, motion planning, visualization, and simulation. The integration of ROS2 and Gazebo is facilitated by a collection of Gazebo plugins [11] that enable support for a diverse range of existing robots and sensors. ROS2 utilizes SDF formats to visualize a robot model. This file format can be used to represent various robot characteristics, such as their shape, color, joints, and other relevant properties.

The main contribution of this work is an exploration of the control features and tools provided by ROS2, employing a two-wheeled, self-balancing mobile robot as a robotic application. The well-known PID and LQR controllers will be implemented and programmed with ROS2, and the ROS2 plot tool will be used to compare the response curves between PID and LQR controllers in real-time data. The Gazebo Robot Simulator will be used to show the simulation of the TWSBMR in a live environment.

The rest of this paper is structured as follows: In Section II, the core functionalities of ROS2 are discussed. The Gazebo Simulator is briefly described in Section III. The mathematical model, PID controller, and LQR controller of a two-wheeled self-balancing mobile robot are presented in Section IV. Section V shows the implementation of a two-wheeled self-balancing mobile robot using ROS2 and the Gazebo Simulator. Finally, some conclusions are drawn in Section VI.

II. ROS2 CORE FUNCTIONALITIES

As mentioned earlier, ROS2 (Robot Operating System 2) is a robust framework for robotics development. It is open-source software that provides tools, interfaces, and components for building sophisticated robots. It allows developers to connect actuators, sensors, and control systems through a software system that includes various features and tools to facilitate software development. These features and tools can be summarized into two main points that aid in understanding the big picture of ROS2:

A. *Code separation and Communication Features*

ROS2 offers a mechanism to divide your code into reusable blocks and provides communication features that make it easy to connect all your subprograms. For instance, you can create a node for the camera, another for the hardware, and another for navigation, and each independent block can communicate with the others in a powerful and scalable way. Some of the main communication features include the following:

- **Nodes:** In ROS2, processes that perform computation are called nodes, and a system can contain multiple nodes. Nodes communicate with one another through the publication of messages to topics.

- **Topics** are used to exchange messages in a publisher-subscriber pattern. Messages contain information that is transmitted and can be of a standard data type (integer, boolean, etc.) or a combination of other types.

- **Publisher-subscriber:** In the publisher-subscriber pattern, those who send information are called publishers, and those who receive messages are called subscribers. A publisher does not send information directly to a subscriber but rather makes it available to anyone who is subscribed to that topic. Subscribers will only subscribe to publishers whose messages are relevant to them.

- **Parameters**: Parameters in ROS2 are associated with individual nodes and are used to configure nodes at start up and during runtime without the need to modify the code. The lifetime of a parameter is linked to the lifetime of the node.

- **Launch files:** Launching multiple nodes one by one is not practical, as it can be time-consuming and confusing. With roslaunch, it is possible to launch all the nodes at the same time.

- **ROS2 bags:** in ROS2, a message stream can be recorded and saved to be replayed later.

In ROS2, a node can publish and subscribe to multiple topics, and it is possible to have multiple subscribers and publishers for the same topic. The messaging system of ROS controls the details of communication between different nodes via the anonymous publish/subscribe system. Fig.2 illustrates the publisher, subscriber, and communication between multiple nodes over a topic that contains a specific type of message.

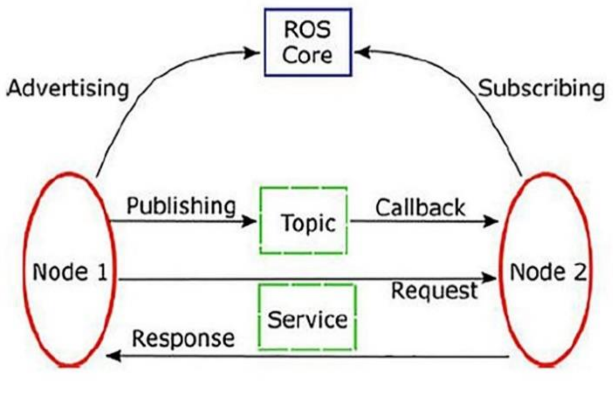


Fig. 2 ROS2: nodes, topics and messages.

B. ROS2 Tools

The second key benefit of ROS2 is that it provides numerous tools and plug-and-play libraries that can save a significant amount of time, and most importantly, prevent the need to reinvent the wheel. Consider the difficulty of computing a trajectory that enables a robot to avoid obstacles while also collaborating with other robots. This task may seem daunting and require extensive knowledge of mathematics, algorithms, and related topics. However, with ROS2 tools, all that is required is a little time to install a library and learn how to use it. This enables access to a vast range of collaboration tools and libraries that can simplify this complex task. ROS2 provides a range of tools that can help simplify the development of robotics applications. Some of the most important tools include:

- **Command-Line Tool:** ROS2 includes a comprehensive set of command-line tools that can be used to monitor and run a ROS2 system. These tools allow developers to work with nodes, topics, services, and other objects without requiring a graphical user interface. With over 45 command-line tools available, developers can access all the core functionality and ROS2 tools they need. ROS2 provides commands that can be used to launch node groups and monitor topics, services, parameters, and other system elements.

- **RQT Tool:** RQT is a graphical user interface (GUI) framework that provides a collection of tools and interfaces in the form of plugins, as well as the ability to manage multiple windows on a single screen. The RQT tool includes numerous plugins that allow for visualizing a running ROS2 system, displaying nodes, and the connections between them, as well as facilitating debugging and understanding the running system and its structure. It can monitor velocity, encoders, position, or any variable that can be defined as a number that changes over time. The RQT tool can quickly provide an overview of projects with limited information or that are growing rapidly. RQT provides information about where the data is coming from and where it is going.

III. GAZEBO SIMULATOR

Gazebo is a 3D dynamic simulator that can accurately and efficiently simulate robot populations in complex indoor and outdoor environments [12, 13]. It is considered one of the best robotics simulators available. It is open-source and widely accessible for simulating robots before building them. Gazebo offers much more reliable physics simulations, as well as a set of sensors and programming interfaces. It is commonly used in real-world scenarios to test robotics algorithms, design robots, and automate control methods. The integration between ROS2 and Gazebo Simulator is provided by a package of Gazebo plugins [11] that support several existing robots and sensors. Because the plugins use the same message interface as the rest of the ROS2 system, it is possible to build ROS2 nodes that work with both simulation and hardware. In this paper, a TWSBMR will be used as an example of a simulation-ready application. Fig.3 shows a TWSBMR loaded into a Gazebo simulator.

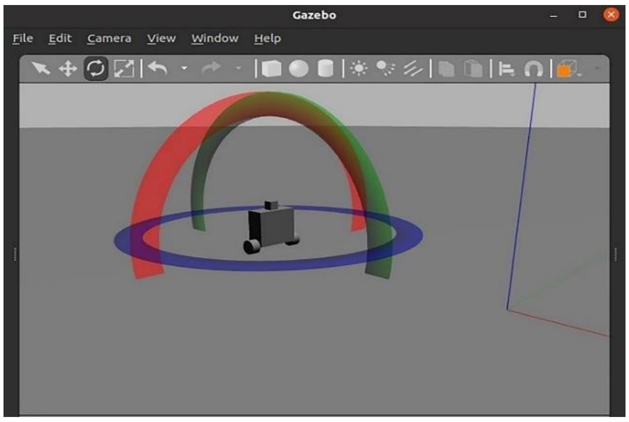


Fig. 3 Gazebo simulator with TWSBMR.

The integration of Gazebo with ROS2 is accomplished through a set of packages called "gazebo_ros_pkgs." This package contains a Gazebo plugin module for communicating between Gazebo and ROS2. Gazebo can send simulated sensor and physics data to ROS2, and ROS2 can send actuator commands back to Gazebo. Gazebo can also be configured to use a robot's ROS API [14]. After that, all robot software can be run on both the real robot and the simulator.ROS2 and Gazebo have the same relationship as ROS2 and the hardware of a real robot. In ROS2, the controller receives data from both the simulation and the real hardware on one topic and publishes it to both the simulation and the hardware on another. Simulation and real robot control can be performed simultaneously, allowing for easy comparison of their behavior and functionality. In fact, whether ROS2 controls a real robot or a simulation model of a robot makes no difference. The structure of the Gazebo simulation model is illustrated in Fig.4. It consists of a model that describes the robot, its plugins, and Gazebo libraries. The model receives data from the Joint Command Interface and sends data back through the Joint State Interface as feedback. The same holds true for the hardware model illustrated in Fig.5, which is linked to the ROS2 controller using the same interfaces. The hardware model includes components such as a controller, actuators, and sensors, which in Gazebo are replaced by simulated ones via plugins and libraries [15].

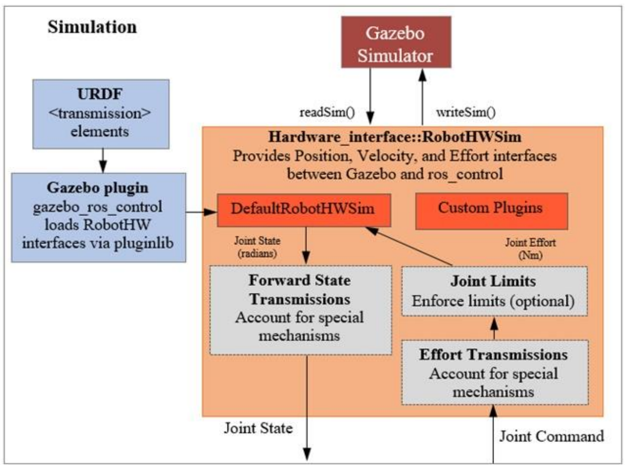

Simulation
Gazebo Simulator
URDF <transmission> elements
Gazebo plugin gazebo_ros_control loads RobotHW interfaces via pluginlib
readSim()
writeSim()
Hardware_interface::RobotHWSim
Provides Position, Velocity, and Effort interfaces between Gazebo and ros_control
DefaultRobotHWSim
Custom Plugins
Joint State (radians)
Joint Effort (Nm)
Forward State Transmissions Account for special mechanisms
Joint Limits Enforce limits (optional)
Effort Transmissions Account for special mechanisms
Joint State
Joint Command

Fig. 4 Principal robot simulation in Gazebo.

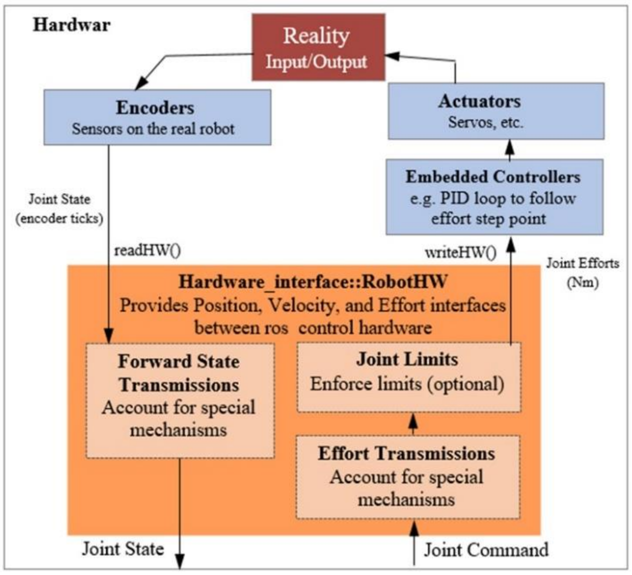

Hardwar
Reality Input/Output
Encoders Sensors on the real robot
Actuators Servos, etc.
Embedded Controllers e.g. PID loop to follow effort step point
Joint State (encoder ticks)
readHW()
writeHW()
Joint Efforts (Nm)
Hardware_interface::RobotHW
Provides Position, Velocity, and Effort interfaces between ros control hardware
Forward State Transmissions Account for special mechanisms
Joint Limits Enforce limits (optional)
Effort Transmissions Account for special mechanisms
Joint State
Joint Command

Fig. 5 Principal robot hardware.

IV. Mathematical Model of Two-Wheeled Self-Balancing Mobile Robot

The schematic diagram of the TWSBMR system is illustrated in Fig.6. The robot consists of two parts: the wheels and the pendulum, which represents as the mass. The robot with its three degree of freedom, can move linearly, which is represented by position x; rotate around the z-axis (yaw) with an associated angle ψ; and rotate around the y-axis (pitch) with an associated angle θ .The system's inputs are torque τ_L, which is applied to the robot's left wheel, and torque τ_R, which is applied to the robot's right wheel. Table II contains the parameter values for the TWSBMR.

TABLE II

PARAMETER VALUES FOR THE TWO-WHEEL SELF-BALANCING MOBILE ROBOT.

Symbol	Parameter	Value	Unit
M	The mass of the pendulum without wheels	0.388	**Kg**
m	wheel mass (left/right)	0.0377	**Kg**
l	length from the center of the pendulum to the wheel axis	0.157	**m**
d	distance between the left and right wheels	0.175	**m**
r	radius of wheels	0.0316	**m**
J, k	wheel moment of inertia with respect to the wheel axis and the vertical axis	0.000233, 0.00002443	**kgm^2**
I1 I2 I3	moment of inertia of the pendulum with respect to the frame at the center of mass of the inverted pendulum	0.01687, ,0.01517 ,0.05897	**kgm^2**

A dynamical model is first established in order to design a controller for the TWSBMR, based on nonlinear robot models cited in the literature [1]. The nonlinear equations of the system are as follows:

$$\left(M + 2m + \frac{2J}{r^2}\right)\ddot{x} - ML\left(\dot{\psi}^2 + \dot{\theta}^2\right)sin(\theta) + (ML\cos\theta)\ddot{\theta} = \frac{\tau_R+\tau_l}{r} \qquad (1)$$

$$(I_2 + ML^2)\ddot{\theta} + (ML\cos(\theta))\,\ddot{x} + (I_3 - I_1 - ML^2)\dot{\psi}^2(\sin\theta)(\cos\theta) - MLg\sin\theta = -(\tau_R + \tau_l) \qquad (2)$$

$$\left(I_3 + 2\mathrm{K} + \mathrm{m}\frac{d^2}{2} + \mathrm{J}\frac{d^2}{2r^2} - (I_3 - I_1 - \mathrm{ML}^2)\,(\sin\theta)^2\right)\ddot{\psi} + (ML\dot{x} - 2(I_3 - I_1 - ML^2)\dot{\theta}\cos\theta)\dot{\psi}\sin\theta = \frac{(\tau_R-\tau_L)d}{2r} \qquad (3)$$

For simplicity, it will be assumed that the TWSBMR only moved in a straight line and that its two wheels acted as a single unit. This resulted in a rotation about the z-axis (yaw), with angle ψ equal to zero. Considering the above assumptions, equations (1) and (2) are linearized about $\theta = \pi$ at the upright equilibrium position. Assume that $\theta = \pi + \emptyset$ where ($\emptyset$ represents a small angle from the vertical upward direction). The state-space form of the linearized equations for the TWSBMR Will be:

$$\begin{bmatrix}\dot{\emptyset}(t)\\ \ddot{\emptyset}(t)\\ \dot{x}(t)\\ \ddot{x}(t)\end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0\\ \frac{Mgl(2J + Mr^2 + 2mr^2)}{2I_2j + 2I_2mr^2 + 2MJL^2 + MI_2r^2 + 2ML^2mr^2} & 0 & 0 & 0\\ 0 & 0 & 0 & 1\\ \frac{-(M^2gL^2r^2)}{2I_2j + 2I_2mr^2 + 2MJL^2 + MI_2r^2 + 2ML^2mr^2} & 0 & 0 & 0\end{bmatrix}$$

$$+ \begin{bmatrix} 0\\ \frac{-(4j + 2Mr^2 + 4mr^2 + 2Mlr)}{2I_2j + 2I_2mr^2 + 2MJL^2 + MI_2r^2 + 2ML^2mr^2}\\ 0\\ \frac{r(2Ml^2 + 2Mrl + 2I_2)}{2I_2j + 2I_2mr^2 + 2MJL^2 + MI_2r^2 + 2ML^2mr^2}\end{bmatrix} u(t) \qquad (4)$$

In this study, the variables of interest are the robot's pitch angle ($\emptyset$) and position (x), and the output equation can be written as:

$$y(t) = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}\emptyset(t)\\ \dot{\emptyset}(t)\\ x(t)\\ \dot{x}(t)\end{bmatrix} \qquad (5)$$

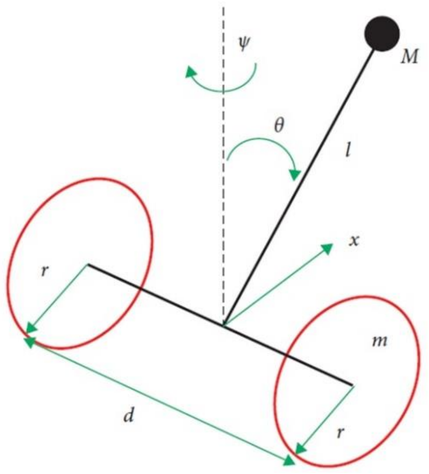


Fig.6.The schematic diagram of the TWSBMR system.

A. *PID Controller*

There are many different types of control algorithms that have been researched in the literature for the TWSBMR system. One of the most popular controllers is the well-known PID type algorithm [16, 17]. In this work, the Ziegler-Nichols method was used to design PID controllers. Using the pitch angle as the output and torque as the input, and utilizing the system model in equation (4), the analysis of the tuning process using MATLAB showed that the optimal response of a PID controller for controlling the pitch angle of the robot is achieved by setting KP = 1.62, KI = 7, and KD = 0.0938.

B. *LQR Controller*

The linear quadratic regulator (LQR) is a widely known linear system controller in industry. The LQR is the most popular approach for designing state-space feedback controls that take into account the states of dynamical systems and control input to make the best possible control decisions [18, 19]. LQR controllers use a linear mathematical model of a system in state-space form. An optimal LQR estimates the controller's gains using the system model described in equation (4).The aim of the controller is to minimize the cost function:

$$J = \int (x^T Qx + u^T Ru)dt \qquad (6)$$

The weighting matrix Q is a symmetric positive semidefinite matrix, while R is defined as a symmetric positive definite matrix. For this controller design, only four states will be considered, and as mentioned before, for simplicity, the yaw angle and the velocity of the yaw angle will be neglected. The control scheme is presented using the system model described in equation (4). Therefore, using the parameter values for the TWSBMR in Table II, the linearized model is given by equation (7).

$$\boldsymbol{A} = \begin{bmatrix}0 & 1 & 0 & 0\\ 49 & 0 & 0 & 0\\ 0 & 0 & 0 & 1\\ -5.93 & 0 & 0 & 0\end{bmatrix} \quad \boldsymbol{B} = \begin{bmatrix}0\\ -792.1\\ 0\\ 221.55\end{bmatrix} \qquad (7)$$

The gain $K = R^{-1}B^TP$ of control law $u = -kx$ is obtained by applying the performance index described in equation (6), where P is a positive definite symmetric constant matrix that can be obtained by solving the algebraic Riccati equation:

$$A^TP + PA + PBR^{-1}B^TP + Q = 0 \tag{8}$$

One of the popular approaches for calibrating the LQR controller is to select Q and R as follows: $Q = C' * C$ and $R = [I]$ .The controller gain K was computed using MATLAB as follows:

K= [-1.6204 -0.2352 -1.0002 -0.6330].

V. Two-Wheeled Self-Balancing Mobile Robot Implementation Using ROS2 and Gazebo Simulator

In this section, the TWSBMR is used to assess the core capabilities and tools of ROS2, and the results are displayed in real-time using the Gazebo Robot Simulator. First, the software architecture of the TWSBMR is introduced. It will start with a functional requirements analysis and then go into the details of the implementation and the functional subtasks. The following is a list of the functional requirements that must be met in order to develop a software architecture for the TWSBMR control system using ROS2:

- Retrieve data from the IMU sensor to determine the pitch angle of TWSBMR.
- Retrieve odometer data from the wheel encoders to know the robot's position.
- The TWSBMR can be controlled by velocity messages.
- The TWSBMR can balance itself vertically and reach the desired position.

The software architecture of the TWSBMR is shown in Fig.7.

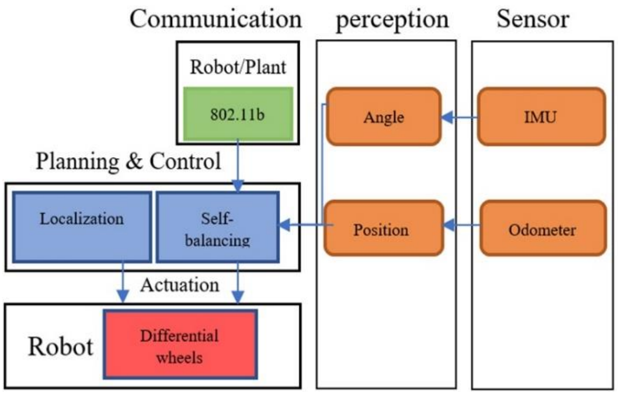


Fig. 7 TWSBMR software architecture.

An important principle of ROS2 is that robotics software should be designed and developed as a distributed system. The purpose is to separate the functions of a complicated system into discrete parts that work with each other to produce the desired behavior of that system. In ROS2, these parts are known as nodes, and their interactions are known as topics and, in some cases, services. A software architecture for controlling the TWSBMR in ROS2 should be designed as a distributed system consisting of nodes and topics that interact with one another to produce the required system behavior. Table III shows node names, types, and functions that are used in building the software architecture for the TWSBMR in ROS2.

TABLE III

NAMES OF NODES, TYPES OF NODES, AND NODE FUNCTIONS

Names of Nodes	Types of Nodes	Functions of Nodes
PID	Publisher/ Subscriber	Receives a data message from the IMU sensor node, then processes it and sends the results of the processing to the motor driver node.
LQR	Publisher/ Subscriber	Receives a data message from the IMU sensor node, then processes it and sends the results of the processing to the motor driver node.
Selfbalancing/ diff_drive	Subscriber	Receives data messages from the PID controller node to move the wheels.
Selfbalancing/ Imu_plugin	Publisher	Determines the robot's pitch angle and position and sends data to the PID or LQR controller node.

A. *ROS2 RQT- graph tool*

As mentioned earlier, ROS2 provides various tools and plug-and-play libraries that facilitate understanding of the running system. One such useful tool is RQT, which includes a wide range of plugins. Among them, the RQT-graph plugin is especially helpful in visualizing a running ROS2 system by displaying nodes and the connections between them. Fig.8 illustrates the data communication between the publisher and subscriber nodes for the TWSBMR using the RQT graph tool.

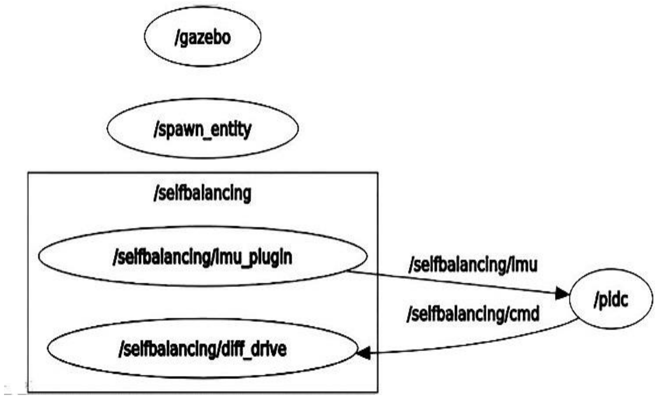
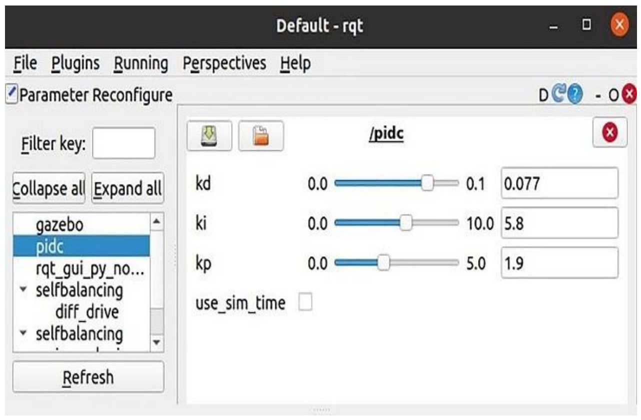


Fig. 8. RQT graph for TWSBMR nodes and topics.

This design separates the software into three ROS2 nodes: one for the device driver, one for the IMU sensor, and one for the PID or LQR algorithms. These nodes communicate with each other, as shown above, via two ROS2 topics. This structure is called a ROS2 communication graph, where the nodes are the graph vertices, and the topics are the graph edges.

B. ROS2 RQT- dynamic reconfigure tool

When building a control system that requires precise control parameters to reach its target at runtime, it is necessary to continuously tune the control system parameters. This involves updating the value of a parameter while a node is running. ROS2 provides an efficient solution for this through a tool called RQT-dynamic reconfigure. It allows users to update parameters on the parameter server at runtime and apply those changes to a specific active node. As a result, a node can detect changes in a parameter's value without needing to restart it.In this study, the RQT reconfigure tool was used to change the PID control parameters' values during runtime to observe their effect on TWSBMR, using the Gazebo Simulator. Fig. 9 displays the slide bars built using the RQT reconfigure tool, which were utilized to change the PID controller node's values for TWSBMR.



/pidc

kd	0.0	0.1	0.077
ki	0.0	10.0	5.8
kp	0.0	5.0	1.9

use_sim_time

Fig. 9 RQT dynamic reconfigure tool for ROS2 PID parameters

C. ROS2 RQT- plot tool

The final ROS2 tool introduced in this paper is the RQT_plot tool, which can plot any numeric value published by ROS2 topics and can also have multiple plots on the same graph. It is useful for visually monitoring the data generated by one or multiple nodes. The RQT_plot tool has the ability to plot the pitch angle, angular velocity, and linear acceleration for TWSBMR.

Figs. 10, 11, and 12 show the TWSBMR response curve using the PID controller, while Figs. 13, 14, and 15 show the response curve using the LQR controller. Both of these response curves were plotted using the RQT_plot tool, which can be used to plot the response of a real robot.

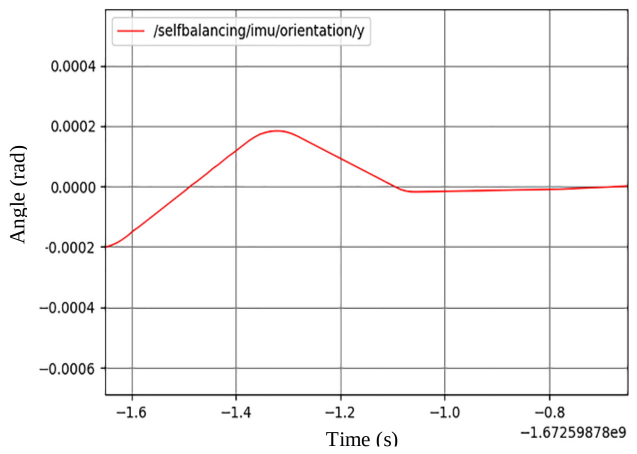


Fig. 10. PID- the response of the pitch angle (θ) of the robot using the RQT_plot tool.

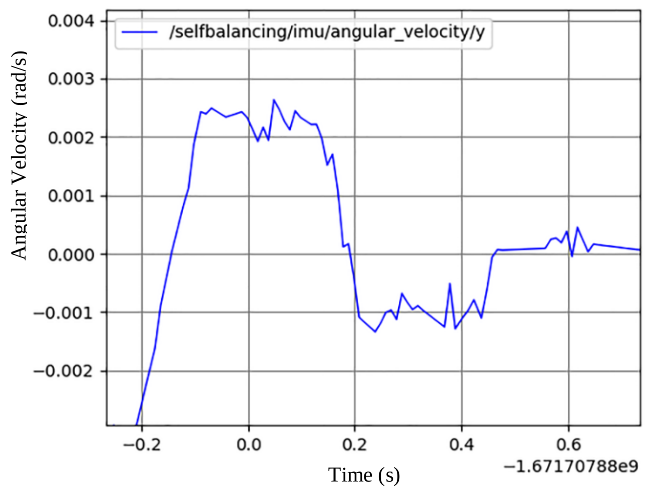


Fig. 11 PID- the response of the angular velocity ($\dot{\theta}$) of the robot using the RQT_plot tool.

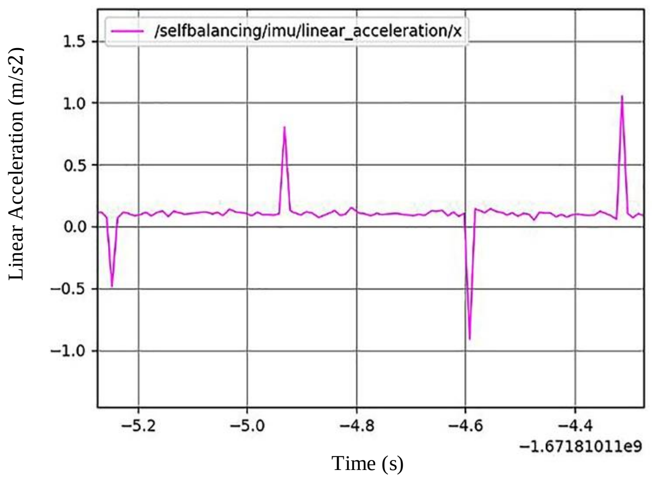


Fig. 12 PID- the response of the linear acceleration ($\ddot{x}$) of the robot using the RQT_plot tool.

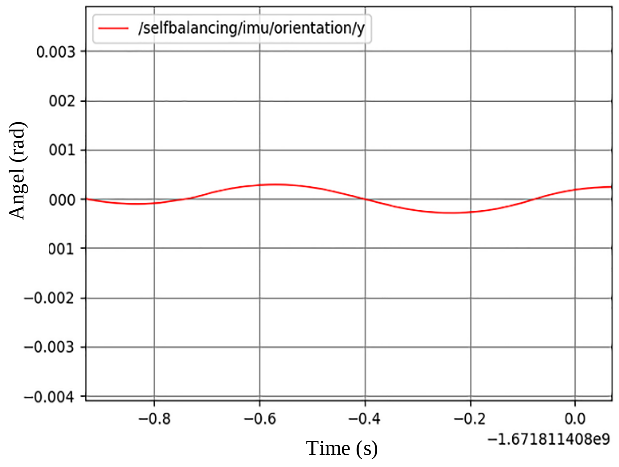


Fig. 13 LQR- the response of the pitch angle (θ) of the robot using the RQT_plot tool.

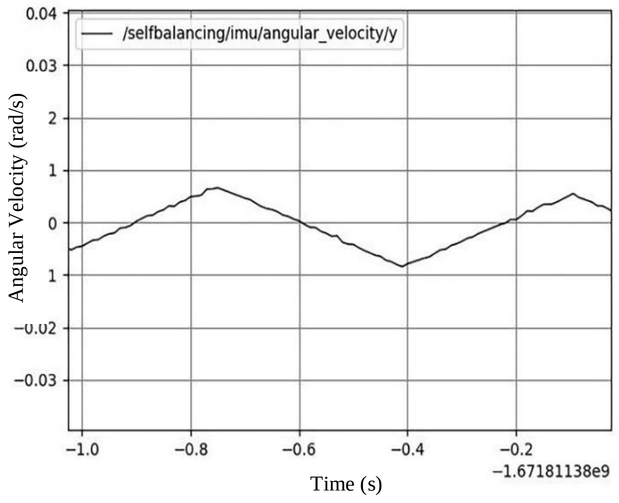


Fig. 14 LQR- the response of the angular velocity ($\dot{\theta}$) of the robot using the RQT_plot tool.

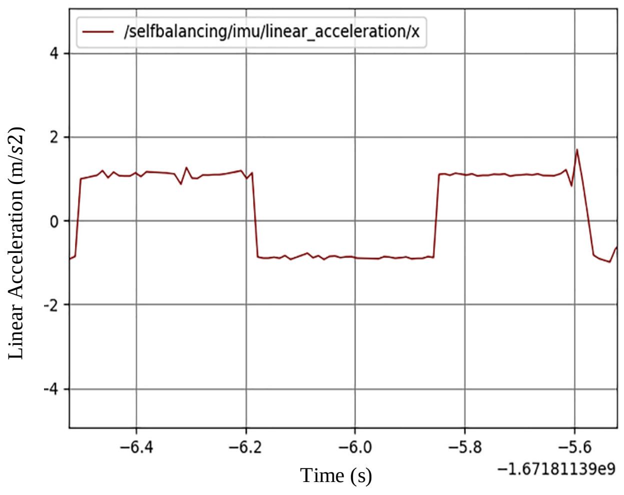


Fig. 15 LQR- the response of the linear acceleration ($\ddot{x}$) of the robot using the RQT_plot tool.

VI. CONCLUSIONS

In this paper, the implementation and simulation of TWSBMR using Robot Operating System 2 (ROS2) has been successfully applied and tested in the Gazebo Simulator. The effectiveness of ROS2 and the Gazebo Simulator for robotic applications has been evaluated in this article. The main core functionality for creating a simulation model of TWSBMR in ROS2 and Gazebo is described. TWSBMR is used as a robotic application system to explore control features, tools provided by ROS2 and the Gazebo Simulator, and to visualize the results in real-time implementation. TWSBMR was first built using the Gazebo Simulator to generate the SDF format file that would be used to implement the robot in ROS2. Then, PID and LQR controller nodes were implemented in ROS2 by designing the control code that includes an IMU sensor plugin, motor plugins, and a model of the robot plugin to be used as a control system to stabilize TWSBMR. As a result, the Robot Operating System and the Gazebo Simulator provide powerful tools for developing and controlling robots. Real-world experiments for TWSBMR are to be conducted in future work.

REFERENCES

1. *S. Kim and S. Kwon, “Dynamic modeling of a two-wheeled inverted pendulum balancing mobile robot,” International Journal of Control, Automation, and Systems, vol. 13, no. 4,pp. 926–933, 2015.https://doi.org/10.1007/s12555-014-0564-8*

2. *F. Grasser, A. D'Arrigo, S. Colombi and A. C. Rufer, "JOE: a mobile, inverted pendulum," in IEEE Transactions on Industrial Electronics, vol. 49, no. 1, pp. 107-114, Feb. 2002, doi: 10.1109/41.982254.*

3. *ROS,”Core Components”, ROS.org. [Online]. Available:*

https://www.ros.org/core-components [Accessed: March 05, 2020]

4. *Pietrzik, S., and B. Chandrasekaran. "Setting up and Using ROS-Kinetic and Gazebo for Educational Robotic Projects and Learning." In Journal of Physics: Conference Series, vol. 1207, no. 1, p. 012019. IOP Publishing, 2019, doi: 10.1088/1742-6596/1207/1/012019.*

5. *ROS,” About ROS”, ROS.org. [Online]. Available:https://www.ros.org/about-ros [Accessed: March 05, 2020]*

6. *A. -M. Hellmund, S. Wirges, Ö. Ş. Taş, C. Bandera and N. O. Salscheider, "Robot operating system: A modular software framework for automated driving," 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), Rio de Janeiro, Brazil, 2016, pp. 1564-1570, doi: 10.1109/ITSC.2016.7795766.*

7. *OSRF, "Introduction to ROS2" , Open Sourse Robotic Foundation., [Online].Available:https://osrf.github.io/ros2multirobotbook/intro.html*

8. *DDS Foundation, "What is DDS?", Object Management Group, Inc.,2019. [Online]. Available: https://www.dds-foundation.org*

9. *SwRI, "ROS1 vs ROS2" , Southwest Research instituite., [Online]. Available:https://www.swri.org/industry/industrial-roboticsautomation/blog/the-ros-1-vs-ros-2-transition*

10. *SDF Format, "What is SDF format", [Online]. Available: http://sdformat.org*

11. *github, "Gazebo plugin", [Online]. Available: https://github.com/ros-simulation/gazebo_ros_pkgs/wiki*

12. *Gazebo Simulator: [Online]. Available: https://gazebosim.org/docs*

13. *K. Takaya, T. Asia, V. Kroumov and F. Smarandache, "Simulation environment for mobile robots testing using ROS and Gazebo," 2016 20th International Conference on System Theory, Control and Computing (ICSTCC), Sinaia, Romania, 2016, pp. 96-101, doi: 10.1109/ICSTCC.2016.7790647.*

14. *OSRF , "What is ROS API ?", Open Sourse Robotic Foundation., [Online].Available:https://osrf.github.io/ros2multirobotbook/ros2_api.html*

15. *Gazebo ROS Control: , [Online]. Available: http://gazebosim.org/tutorials?tut=ros_control&cat=connect_ros*

16. *D. Pratama, E. H. Binugroho and F. Ardilla, "Movement control of two wheels balancing robot using cascaded PID controller," 2015 International Electronics Symposium (IES), Surabaya, Indonesia, 2015, pp. 94-99, doi: 10.1109/ELECSYM.2015.7380821.*

17. *A. T. Ali, A. M. O. Mohamedy, A. S. A. Salimz, E. -A. O. M. El-Aminx and O. M. K. Ahmed, "Design and Implementation of Two-Wheeled Self-Balancing Robot Using PID Controller," 2020 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE), 2021, pp. 1-5, doi: 10.1109/ICCCEEE49695.2021.9429579.*

18. *R. B¨uchi, State Space Control, LQR and Observer: step by step introduction, with Matlab examples. Books on Demand, 2012. [Online].Available:https://books.google.com.tr/books?id=JrofAgAAQBAJ.*

19. *M. Almobaied, I. Eksin and M. Guzelkaya, "Design of LQR controller with big bang-big crunch optimization algorithm based on time domain criteria," 2016 24th Mediterranean Conference on Control and Automation (MED), 2016, pp. 1192-1197, doi: 10.1109/MED.2016.7535907.*