Cairo University
Faculty of Engineering
Department of Mechanical Engineering
Mechatronics Program

Hardware Implementation & Simulation of a Differential Drive Robot

Advanced Robotics & Artificial Intelligence MPDS453

Course Midterm Task

Team Members

Abdelhamid Walid Abdelhamid [1190463]

Abdelrahman Adel Ramadan [1190140]

Mostafa Mohamed Mahmoud Shafik [1190485]

Mohamed Adel Hamzawi [1190500]

Submitted on: 24th of April, 2024

Submitted to: Dr. Hossam Ammar

Abstract

This report serves as a comprehensive documentation for the Advanced Robotics & Artificial Intelligence course midterm project, which revolves around using robot operating system (ROS) to communicate and position a two-wheel robot using a joystick and simulate the robot movement on Gazebo software. Throughout this report the implemented hardware design is presented as well as the presenting the connections and the detailed circuitry of the robot. The communication through ROS nodes is clarified and the kinematic model used to position the hardware model and the simulation model is presented. The codes used in this project are uploaded on the project's GitHub repo found through this link: https://github.com/bido100/ROS_Mid_Task

Table of Contents

List of Figures

I. Introduction & Selected Robot

The TurtleBot Burger, a versatile and widely-used robot platform in robotics education and research, serves as the foundation for our project. In this report, we detail the fabrication and development process of our differential drive robot based on the TurtleBot Burger platform. Our objective was to create a robot capable of autonomous navigation and environment mapping, utilizing ROS (Robot Operating System) for communication between the hardware, Gazebo simulation environment, and the physical robot.

1.1 Kinematic Model

Since the TurtleBot Burger is a two-wheel robot with a caster wheel, we could utilize the simplified version of the kinematic model of two-wheel drive robot, which are:

$$\omega_r = (2V - \omega b)/2r_r$$

$$\omega_l = (2V + \omega b)/2r_l$$

Where:

- ω_r & ω_l are the right and left motors rotational speeds respectively.
- V is the linear velocity of the robot
- ω is the rotational velocity of the robot
- b is the wheelbase
- r_r & r_l are the wheel radii

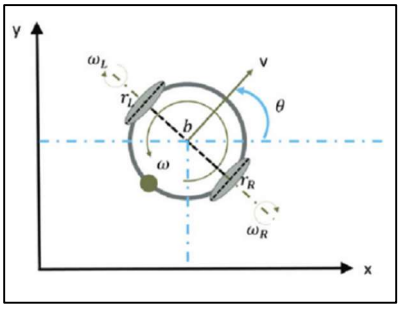

Figure 1 Two Wheel Differential Drive Robot Kinematic Diagram & Parameters

II. Communication Through ROS Nodes

2.1 Main Teleportation Python Node

2.1.1 Node Description

Communication between the robot motors, joystick, and gazebo simulation is done on a python node called the "teleportation node" which coordinates the communication between the node as in the following diagram:

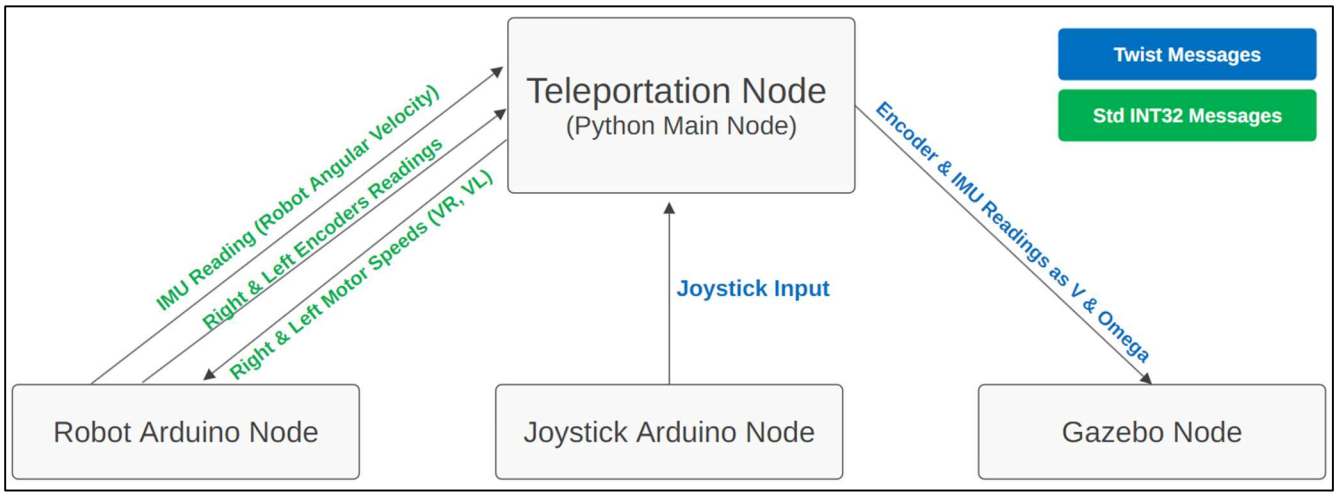

Figure 2 ROS Nodes Illustration

When the teleportation node is started, it first subscribes to the joystick node and takes the topic of the joystick input, which is sent as a twist message, i.e., the joystick input in the Y-direction is mapped from -Vmax to +Vmax and the joystick input in the X-direction is mapped from $-\frac{\pi}{2}$ to $+\frac{\pi}{2}$ rad/s, these two inputs are unpacked as a twist message in the python node are used as the input to the inverse kinematics, which is performed on the teleportation node as well.

When the two joystick input are transformed into linear and angular velocities and inputted to the inverse kinematics, the output is the two wheel velocities in rad/s, which are published to the robot Arduino node as a standard Int32 message, utilizing the STD_MSGS library.

When the robot moves with the specified velocities, it publishes 3 standard Int32 messages, the two encoder readings in *rad/s*, which are the right and left wheel actual velocities, as well as the IMU reading in *deg/s*, which is the robot angular velocity in the Z-axis.

These 3 parameters (topics) are processed in the same teleportation node on python, which takes them as the input to the last process, which is transforming these readings into a twist message so it could be published to the Gazebo node. To achieve this, the python node first calculate the average robot velocity from the two wheel velocities and assign it to the linear velocity of the Gazebo's twist message. The IMU reading is converted from *deg/s* to *rad/s* and published to gazebo as the angular velocity.

2.1.2 Node Setup and Running

1. First we save the python node code on the workspace of the project. We then edit the node to make necessary subscribe/publish initialization to correctly publish and subscribe to the two Arduino nodes, making sure the topic names are correct.
2. Then to connect and run the node, we first run roscore.
3. We then connect the joystick Arduino via USB cable, then run rosserial command and specify the port of this Arduino.
4. We then define the address of the Bluetooth module of the second Arduino (motors).
5. After the address is set, rosserial is run again but this time for the robot's Arduino node, we make sure that serial port that corresponds to the Bluetooth module is correct and that both arduinos nodes have different names in rosserial.
6. In order to run the simulation, we run the launch file (mentioned later in this report) that connects the URDF file to the Gazebo's plugin
7. Finally, we run the ".py" file of the teleportation node, and use the joystick to move the robot, take feedback data and use it to move the simulated robot.

2.2 Robot Arduino Node

This node as mentioned previously, is responsible for actuation of the real robot, and send its feedback data back to the processing node. To better illustrate the flow inside this node, a flow chart is given below:

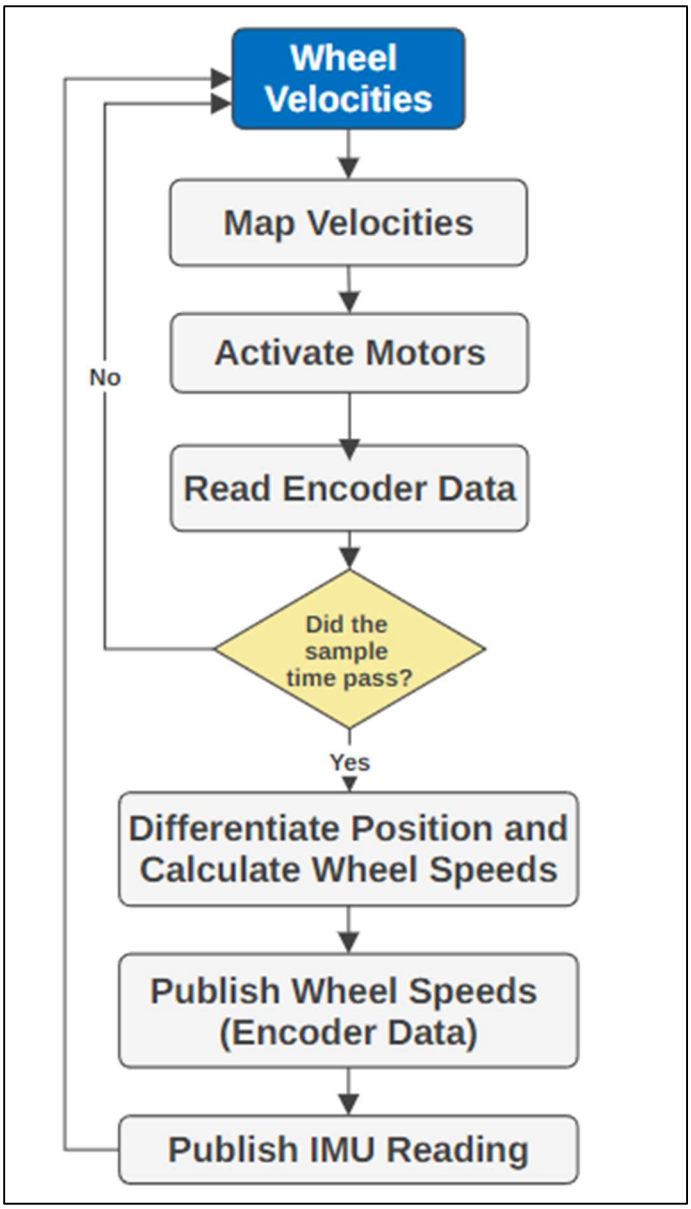

Figure 3 Robot Arduino Node Flowchart

III. Hardware Implementation

3.1 CAD Model

For the implemented TurtleBot Burger robot, the CAD design is widely available as it is published open source by open robotics, so we downloaded the CAD model shown in fig(). And we used the chassis parts "the shelves or floors" and laser cut them from plywood to formulate our own chassis.

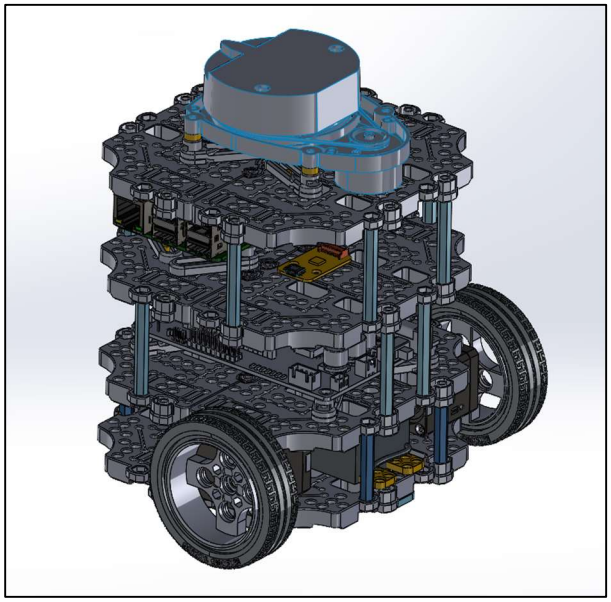

Figure 4 TurtleBot Burger Published CAD Model

3.2 Hardware Components

3.2.1 Components List

Mechanical Assembly Components:

- Laser cut chassis.
- Spacers.
- Motor couplings and brackets.
- 55mm diameter wheels.

Electronic Components:

- Arduino Mega development board.
- Arduino Nano development board.
- Joystick Module.
- 12VDC 210RPM DC Motors with encoders.
- L298N 2 channel motor driver.
- MPU6050 IMU.
- HC-05 Bluetooth Module.
- 12V Battery.
- Switches & Wiring.

3.2.2 Project Budget List

Note: Not all components are included in the budget since they were already available.

Component	Unit Price	Quantity	Price
Arduino Mega Development Board	1450	1	1450
Joystick Module	40	1	40
12V210RPM DC Motor with Encoder	700	1	700
L298N Motor Driver	0	1	0
55mm Wheels	300	2	600
HC-05 Bluetooth Module	250	1	250
MPU6050 IMU	110	1	110
12V 5A Battery	500	1	500
Laser Cut Robot Chassis Piece	135	5	675
12V Adapter	120	1	120
Total Price			4445

3.3 Circuitry & Connections

3.3.1 Overall System Connections

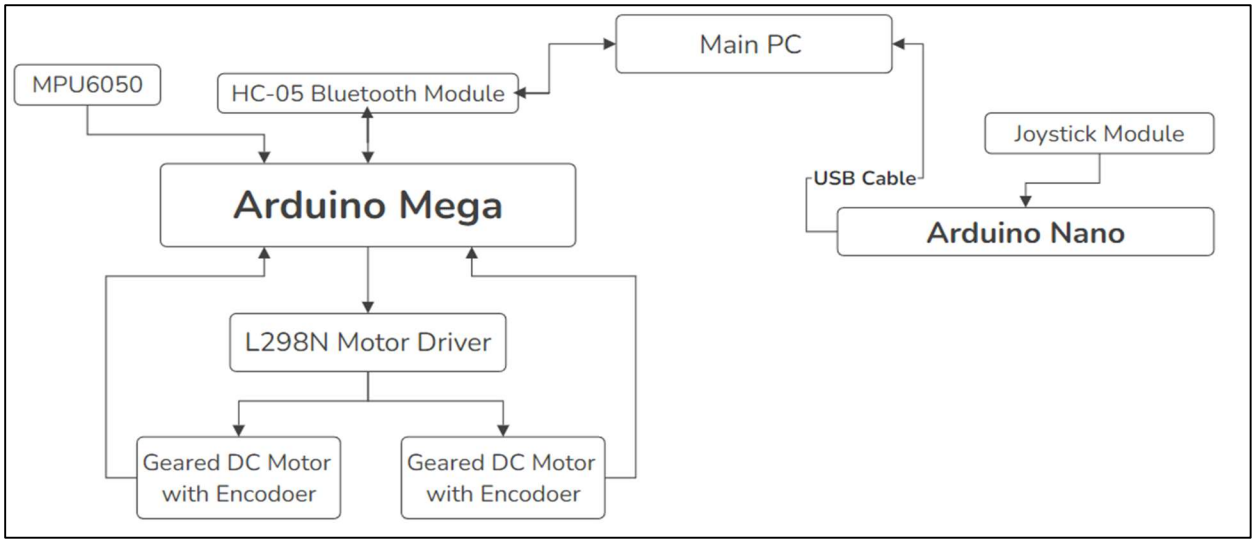

Figure 5 Overall System Connections

3.3.2 Two Wheel Robot Detailed Wiring

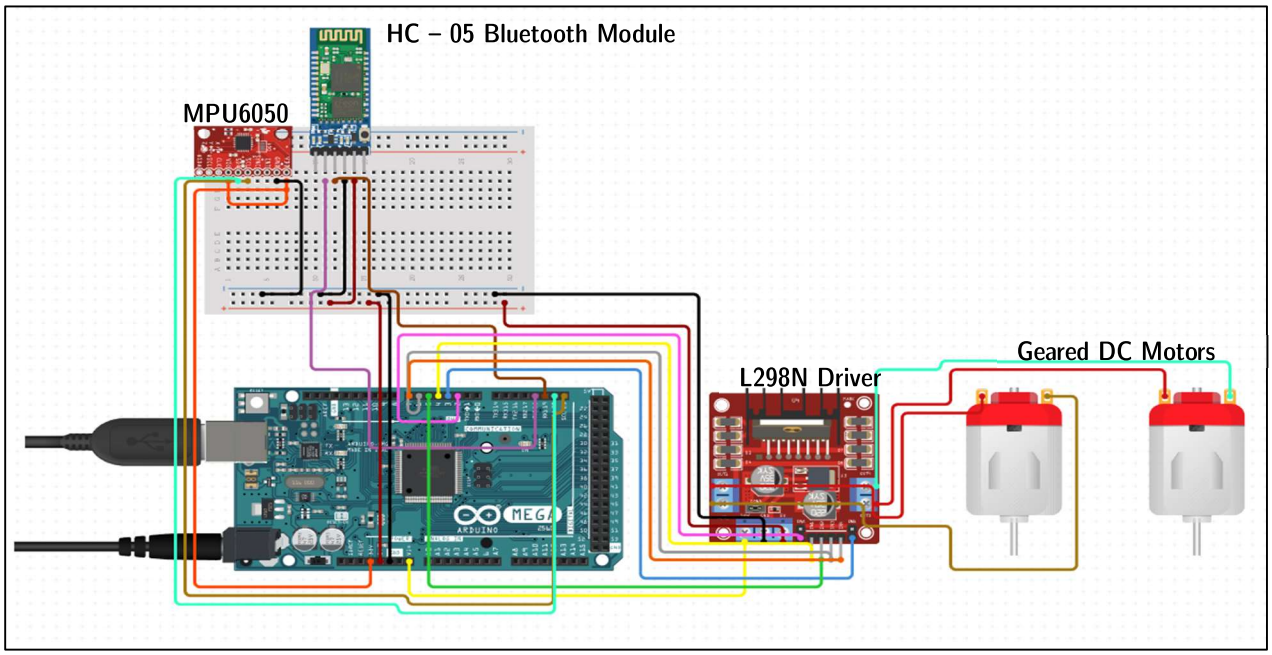

Figure 6 Robot Schematic Wiring

3.3.3 Joystick Module Detailed Wiring

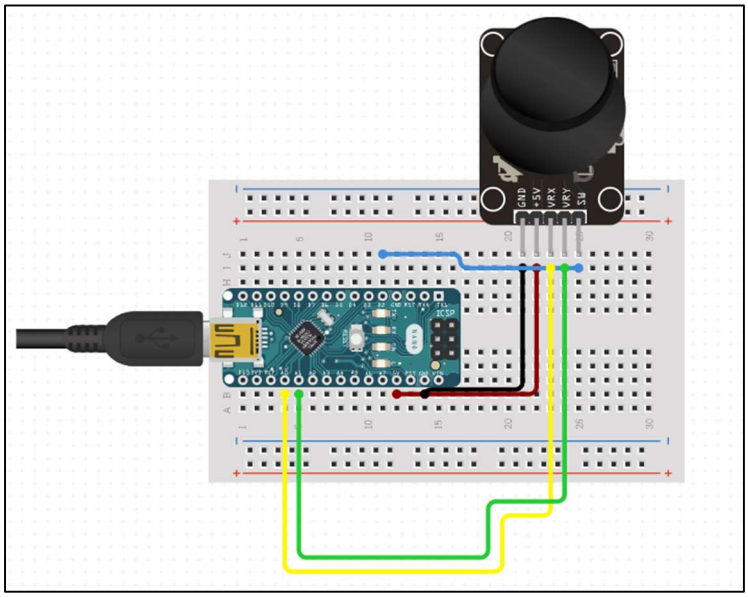

Figure 7 Joystick Module Schematic Wiring

3.4 Hardware Interfacing

3.4.1 DC Motor Encoder

he 12VDC motor used in our differential drive robot comes with an encoder that provides feedback on the motor's rotation. The encoder has two pins, one of which is attached to an interrupt pin on the microcontroller (Arduino) for precise counting of the motor's revolutions. Interfacing with the encoder is crucial for accurately measuring the motor's rotation, which is essential for feedback control and odometry calculations.

The encoder is connected to the Arduino's interrupt pin and another digital pin for reading the encoder's output. By monitoring the interrupt pin, the Arduino can detect each pulse generated by the encoder, allowing us to track the motor's movement with high accuracy.

In the Arduino code, we set up the interrupt to trigger on the rising or falling edge of the encoder signal, depending on the encoder's configuration. When the interrupt is triggered, the Arduino increments or decrements a counter variable based on the direction of rotation. This counter value is then used for calculating the motor's position by correcting to the number of pulses per revolution of the encoder.

To calculate the speed of the wheel, we simply define a fixed sample time and we find the position before and after that sample time, which equal to the number of radians(positions) passed within this sample time.

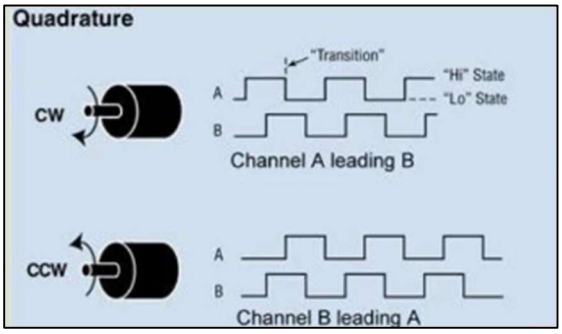

Figure 8 Motor Encoder Channel Outputs

IV. Gazebo Simulation

4.1 Setup Brief

In order to simulate the robot behavior with the sensors feedback data on gazebo, we first downloaded the raw pre-defined URDF TurtleBot Burger file from RosWIKI then we edited this file to include the Gazebo plugin and include the topics we run, which is the twist message published by the python node. We then created a launch file, which connects the URDF file with Gazebo and runs gazebo with the URDF file. Since we edited the URDF file and included our twist topic, the robot model on Gazebo works only when feedback data (encoder & IMU readings) are present.

The twist message published by the python node contains two parameters which position the robot in the simulation, the linear velocity and the angular velocity. The linear velocity is approximated by calculating the two wheels linear velocities knowing the rotational velocity and the wheel radius. The angular velocity comes from the gyroscopic measurements of the IMU.

4.2 Debugging

Throughout setting the Gazebo's simulation, we encountered some issues that we tried to resolve such as:

- The model rotating around itself in the environment without any input i.e. no feedback from the robot, yet the mode is rotating. This was due to the noise accompanied with the IMU readings. We resolved this by creating a filter for low readings to be zero. The threshold of the filter was found by printing the angular velocity (Gazebo's input) on the terminal and see the value at which the robot is stationary yet the angular velocity is changing.
- The model at the instant of robot stop, this was due to the fact that the friction parameters were not correctly configured in the Gazebo's plugin in the URDF file.
- The model was moving unpredictable movements for sometimes and sometimes not, this was due to the fact that there were two topics named with the same name in the communication between the nodes.