Industrial Mobile Robot

Summer Intership Report

Bachelor of Technology
in
Automation and Robotics

by

Suman Das Adhikary
TNU2022036100017
Raju Mondal
TNU2022036100002
Snehendu Patra
TNU2022036100007

Department of Robotics and Automation Engineering

The Neotia University, Sarisha, Diamond Harbour

Under the guidance of
Dr. Anirban Nag

SCHOOL OF MECHATRONICS AND ROBOTICS
INDIAN INSTITUTE OF ENGINEERING SCIENCE AND TECHNOLOGY, SHIBPUR - 711103

June-July, 2025

ACKNOWLEDGEMENTS

First and foremost, we would like to express our heartfelt gratitude to our internship supervisor, **Dr. Anirban Nag**, Assistant Professor, School of Mechatronics and Robotics, IIEST Shibpur, for his exceptional mentorship, constant encouragement, and insightful guidance throughout the duration of our work. His expertise, critical feedback, and patient supervision have played a significant role in shaping the technical depth and direction of this project.

We are also sincerely thankful to **Prof. Dr. Tanmay Pal**, Professor and Head, School of Mechatronics and Robotics, IIEST Shibpur, for providing us with an academically rich and supportive environment that greatly facilitated our learning and exploration.

Our sincere appreciation also extends to all the faculty members and technical staff at IIEST Shibpur for their welcoming attitude, valuable input, and the access to lab resources that supported our progress during the internship.

We are especially grateful to the **Department of Robotics and Automation Engineering, The Neotia University**, and to its **Head of Department**, for providing us the opportunity to undertake this internship and for their continuous encouragement and institutional support throughout.

We would also like to thank our faculty mentors and coordinators at Amrita School of Engineering for guiding us and helping coordinate this internship, which has been a significant learning milestone in our academic journey.

Lastly, we extend our deepest gratitude to our parents and families, whose unwavering support, trust, and encouragement have always inspired and empowered us to pursue excellence.

ABSTRACT

This report presents a comprehensive study on the design, implementation, and control of a mobile robotic platform for autonomous ground navigation. The system integrates four planetary-geared DC motors for omnidirectional movement, each driven independently using high-current motor drivers to ensure robust actuation and traction performance across diverse terrains.

A distributed control architecture was implemented using four Arduino Uno microcontrollers configured as slave nodes and a Raspberry Pi serving as the central master controller. Communication between the master and slave units was established via serial protocols, enabling efficient command distribution, motor synchronization, and sensor data acquisition. The modular control system enhances scalability and simplifies diagnostics and maintenance.

Mechanical stability was achieved through a robust chassis mounted on a ladder-type frame, providing structural integrity while housing all electronic and power systems. Real-time control algorithms running on the Raspberry Pi manage navigation logic, sensor fusion, and command issuance, while the Arduino boards handle low-level motor control tasks, including PWM signal generation and encoder feedback.

This layered system architecture bridges the gap between high-level autonomous planning and low-level hardware execution, offering a practical and cost-effective approach to mobile robotics. The prototype serves as a foundational platform for future work in autonomous delivery systems, surveillance bots, and educational robotics.

Keywords: Mobile Robot, Raspberry Pi, Arduino Uno, planetary geared DC motor, motor driver, distributed control, serial communication, embedded systems, robotics chassis, ladder frame.

9

Contents

Chapter 1

Introduction

The field of robotics continues to evolve at the intersection of mechanical engineering, electronics, and intelligent control systems. Among the many branches of robotic development, mobile robotics plays a pivotal role in automating transport, surveillance, exploration, and industrial logistics. This report presents the design and implementation of an autonomous and remotely controlled mobile robotic platform, engineered for efficient terrain navigation and adaptable control using a layered microcontroller-based architecture.

The mobile robot developed in this study utilizes four planetary-geared DC motors configured in a differential drive arrangement to achieve precise and powerful movement. Each motor is driven through an independent motor driver, ensuring adequate current delivery and torque control. A robust ladder-frame chassis supports the system and houses all embedded components, ensuring structural stability during operation.

To manage the system effectively, a distributed control framework was adopted. Four Arduino Uno microcontrollers function as slave nodes, each assigned to control a motor and handle local encoder feedback. These nodes are managed by a Raspberry Pi acting as the central master controller. Communication between the master and slave controllers is established via serial protocols, enabling synchronized motion control and real-time feedback monitoring.

The robot supports three modes of operation: remote control, serial command-based operation, and autonomous navigation. In remote control mode, user input is provided wirelessly through a custom interface. Serial control mode allows the robot to be manipulated via terminal-based commands or scripts, while the automatic mode uses pre-programmed path planning and sensor-based decision making for navigation. This flexibility enhances the robot's usability across various applications, including indoor delivery, surveillance, and educational demonstrations.

This report explores the complete development cycle of the robot — from mechanical design and embedded system integration to software architecture and real-world testing. The platform was designed with scalability in mind, enabling easy upgrades such as sensor integration, camera-based vision, or SLAM (Simultaneous Localization and Mapping) for autonomous mapping.

Objectives of the Study

- To design and fabricate a mobile robotic platform using a ladder-type chassis and planetary-geared DC motors.

- To implement a distributed control architecture using Arduino Uno (as slaves) and Raspberry Pi (as master).
- To develop and test remote, serial, and autonomous control modes for flexible robot operation.
- To ensure robust communication between controllers via serial protocols.
- To validate the system performance through real-world testing and control accuracy analysis.

Methodology Overview

The project follows a modular development approach. Initially, the mechanical platform was designed and assembled, ensuring optimal motor placement and weight distribution. The control system was divided into two layers: low-level control handled by Arduino Unos, and high-level logic executed on a Raspberry Pi. Communication was established using serial UART links. Remote control was implemented using wireless modules, and command processing was developed in Python on the Raspberry Pi. Each control mode was tested iteratively to ensure system stability and responsiveness.

This integration of mechanical design, embedded control, and communication provides a comprehensive understanding of real-world mobile robot development. The project serves as both a theoretical and practical guide for building intelligent robotic platforms that can bridge the gap between academic study and applied automation.

Chapter 2

Geometric Structure and Kinematic Analysis of the Mobile Robot

2.1 Geometry and Kinematic Architecture of the Mobile Platform

The geometric configuration of a mobile robot directly influences its locomotion, stability, and control dynamics. In this project, the robot is designed with a symmetrical four-wheel layout mounted on a ladder-type chassis. Each wheel is powered by a high-torque planetary-geared DC motor, allowing independent control over wheel velocities. The wheel arrangement follows a differential drive configuration, a widely adopted architecture in mobile robotics for its simplicity and effectiveness.

The chassis, fabricated using lightweight materials, ensures minimal vibration and even weight distribution. The wheelbase (distance between front and rear axles) and track width (distance between left and right wheels) are selected based on the desired turning radius, maneuverability, and terrain compatibility. This configuration allows the robot to perform point turns, straight-line motion, and controlled curved paths.

From a kinematic perspective, the robot is modeled as a rigid body on a 2D plane with two driven wheels on either side. The primary kinematic assumptions include:

- All wheels maintain pure rolling contact with the ground (no slip condition).
- The robot operates on a flat surface (planar motion).
- The center of mass is centrally located between the two drive wheels.

Let v represent the linear velocity of the robot and ω its angular velocity about the vertical axis passing through its center. Let v_R and v_L denote the linear velocities of the right and left wheels respectively, and L be the distance between the wheels (track width). The robot's forward kinematics is given by:

$$v = \frac{v_R + v_L}{2}, \qquad \omega = \frac{v_R - v_L}{L}$$

These equations describe how the combined motion of the wheels translates to the robot's global velocity and angular rotation. Inverse kinematics, used for control purposes, is derived as:

$$v_R = v + \frac{L}{2}\omega, \qquad v_L = v - \frac{L}{2}\omega$$

These relations enable the control system to calculate the required wheel speeds to follow a desired path or trajectory. This architecture supports all three operating modes of the robot — remote control, serial command, and autonomous operation — by converting user or algorithm-defined motion commands into executable motor signals.

The geometric and kinematic architecture presented in this section serves as the foundation for the control strategies, motion planning, and sensor integration discussed in the following chapters.

2.2 Mathematical Formulation of Forward Kinematics

Forward kinematics in mobile robotics refers to the calculation of a robot's position and orientation over time based on its wheel velocities. For a differential drive robot, the motion occurs in a 2D plane, and the pose of the robot can be represented by three variables: the x-coordinate, y-coordinate, and heading angle θ.

Robot Configuration and Assumptions

The robot is assumed to move on a flat surface, and the kinematic model considers the following parameters:

- R – radius of the wheels (in meters)
- L – distance between the two drive wheels (wheelbase)
- ω_L, ω_R – angular velocities of the left and right wheels (rad/s)
- $x(t), y(t)$ – position of the robot at time t
- $\theta(t)$ – orientation of the robot at time t

From wheel angular velocities, we calculate the linear velocities:

$$v_L = R \cdot \omega_L, \quad v_R = R \cdot \omega_R$$

Velocity-Based Forward Kinematics

The robot's linear velocity v and angular velocity ω about its vertical axis are given by:

$$v = \frac{v_R + v_L}{2}, \quad \omega = \frac{v_R - v_L}{L}$$

The motion of the robot can be modeled using the following differential equations:

$$\frac{dx}{dt} = v \cdot \cos(\theta)$$

$$\frac{dy}{dt} = v \cdot \sin(\theta)$$

$$\frac{d\theta}{dt} = \omega$$

These equations describe the instantaneous change in position and orientation of the robot given its wheel velocities.

Discretized Implementation

For practical implementation on a microcontroller or simulation software, the continuous-time model is discretized using a small time step Δt:

$$x_{t+1} = x_t + v \cdot \cos(\theta_t) \cdot \Delta t$$

$$y_{t+1} = y_t + v \cdot \sin(\theta_t) \cdot \Delta t$$

$$\theta_{t+1} = \theta_t + \omega \cdot \Delta t$$

This formulation is embedded into the robot's control loop (e.g., using Python or C++) to estimate the current pose based on encoder inputs or motor commands. Accurate forward kinematics allows for real-time localization and trajectory tracking, especially in autonomous or semi-autonomous operation modes.

2.3 Forward Kinematics: Example Solution and Analysis

To illustrate the practical application of the forward kinematics model, consider a scenario in which the mobile robot is moving on a flat surface with known wheel parameters and motor

inputs. The objective is to compute the robot's updated position and orientation after a small time step.

Given Parameters

- Wheel radius, $R = 0.05\,m$
- Distance between wheels (wheelbase), $L = 0.30\,m$
- Initial position: $x_0 = 0\,m, \quad y_0 = 0\,m, \quad \theta_0 = 0\,rad$
- Angular velocities of wheels:

$$\omega_L = 2\,rad/s, \quad \omega_R = 4\,rad/s$$

- Time step, $\Delta t = 1\,s$

Step 1: Compute Wheel Linear Velocities

$$v_L = R \cdot \omega_L = 0.05 \cdot 2 = 0.1\,m/s$$

$$v_R = R \cdot \omega_R = 0.05 \cdot 4 = 0.2\,m/s$$

Step 2: Compute Robot Linear and Angular Velocity

$$v = \frac{v_R + v_L}{2} = \frac{0.2 + 0.1}{2} = 0.15\,m/s$$

$$\omega = \frac{v_R - v_L}{L} = \frac{0.2 - 0.1}{0.3} = 0.333\,rad/s$$

Step 3: Update Robot Pose Using Discretized Equations

$$x_1 = x_0 + v \cdot \cos(\theta_0) \cdot \Delta t = 0 + 0.15 \cdot \cos(0) \cdot 1 = 0.15\,m$$

$$y_1 = y_0 + v \cdot \sin(\theta_0) \cdot \Delta t = 0 + 0.15 \cdot \sin(0) \cdot 1 = 0\,m$$

$$\theta_1 = \theta_0 + \omega \cdot \Delta t = 0 + 0.333 \cdot 1 = 0.333\,rad$$

Result

After 1 second of motion:

- The robot has moved 0.15 meters forward.
- It has not shifted laterally (since initial heading was 0).
- Its orientation has changed by 0.333 radians ($\approx 19.1°$), indicating a leftward arc.

Analysis

This outcome confirms that when the right wheel spins faster than the left, the robot follows a curved trajectory turning to the left. The rate of rotation (ω) is directly proportional to the speed difference between wheels and inversely proportional to the distance between them. Such forward kinematics computations are critical in autonomous mode, where real-time estimation of robot pose is needed for navigation and feedback control.

Moreover, repeated application of this discretized model allows for pose tracking over time, essential for dead reckoning, localization, and path planning. Any drift or deviation can be corrected through sensor fusion or feedback mechanisms such as wheel encoders, IMUs, or GPS.

Chapter 3

3D Modeling and 2D Sketch Development Using Autodesk Tools

Designing a mobile robotic platform requires a detailed mechanical representation before physical fabrication. In this chapter, the structural layout of the robot is developed using 3D modeling and 2D sketching techniques with Autodesk Inventor and Fusion 360. These tools allow precise visualization of the mechanical structure, alignment of motors and wheels, and mounting of electronic components.

3.1 Design Tools and Workflow

Two Autodesk software platforms were used in this project:

- **Fusion 360** – Used for conceptual modeling, visual assembly, and lightweight part modeling.
- **Autodesk Inventor** – Used for precise parametric modeling, 2D sketching, and generating fabrication-ready drawings.

The design process began with basic 2D sketches to define essential dimensions like chassis length, wheelbase, and component spacing. These sketches were later extruded or revolved to form 3D components, which were assembled virtually to check for fit and balance.

3.2 Structural Layout and Component Modeling

The mobile robot chassis follows a ladder-type frame configuration. The design includes mounting points for:

- Four DC planetary-geared motors and wheels
- One Raspberry Pi (master controller)
- Four Arduino Uno boards (slave motor controllers)
- Battery pack and power distribution wiring

Motor mounts, base plates, and sensor brackets were modeled as individual components and assembled virtually to ensure mechanical compatibility. The model also provided estimates of size, weight, and center of gravity.

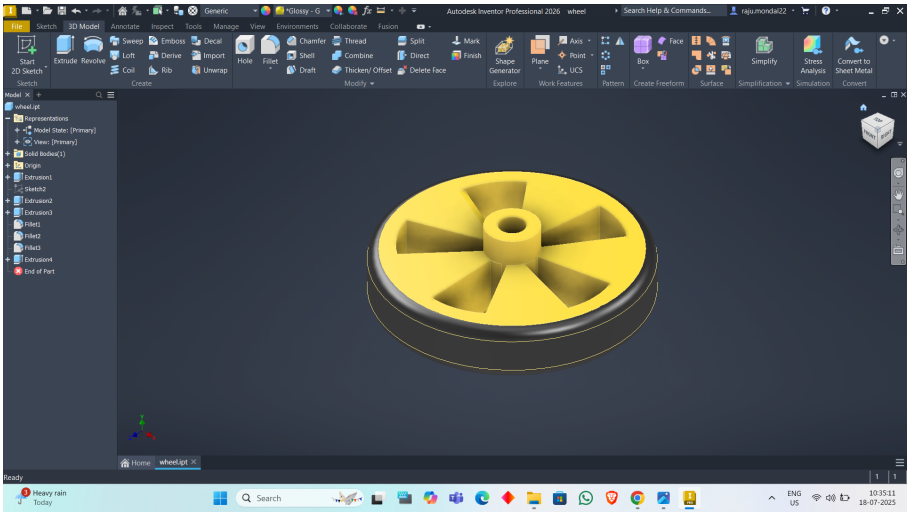

Figure 1: Wheel

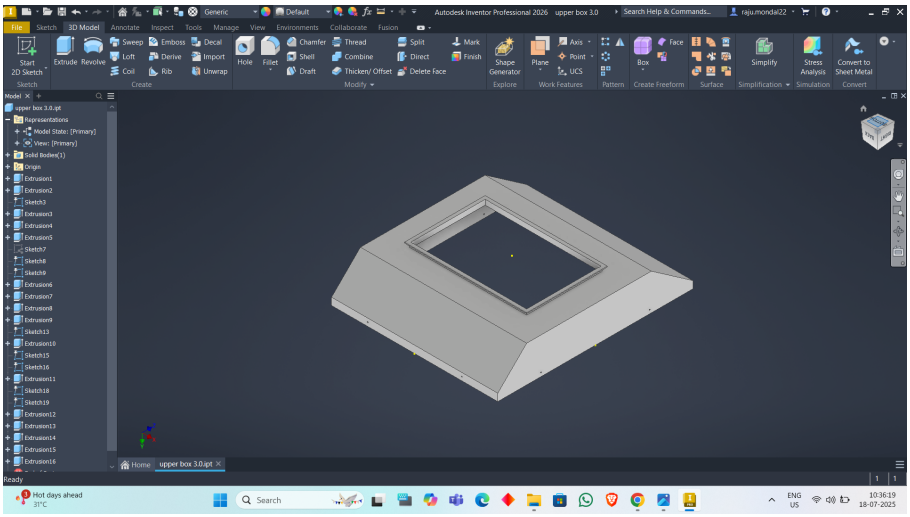

Figure 2: Upper Part

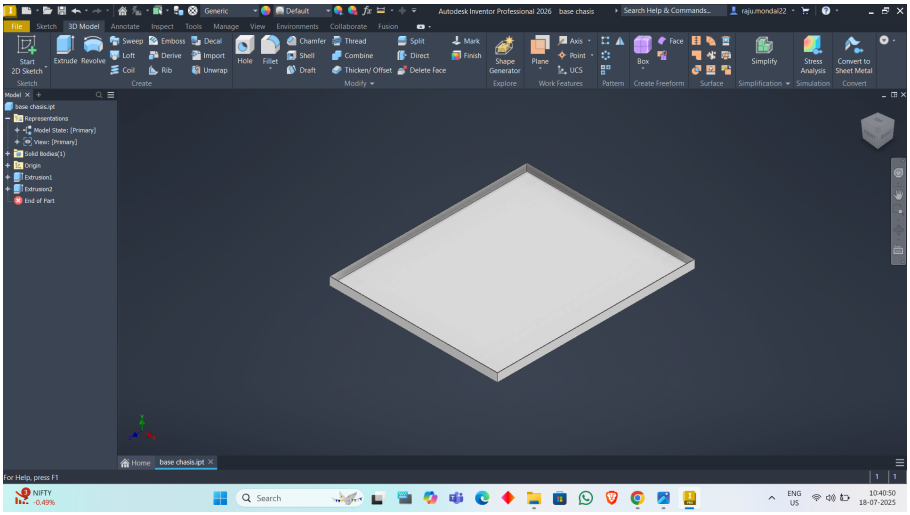

Figure 3: Base Chasis

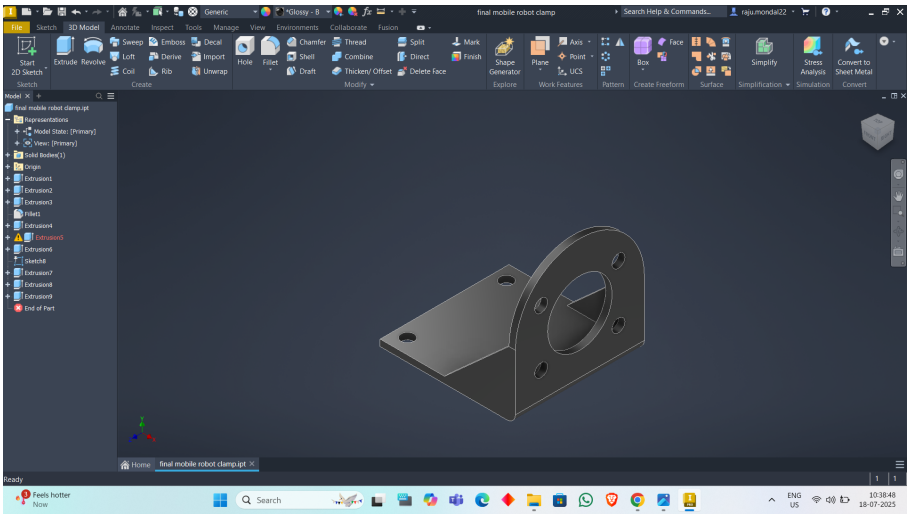

Figure 4: Robot Clamp

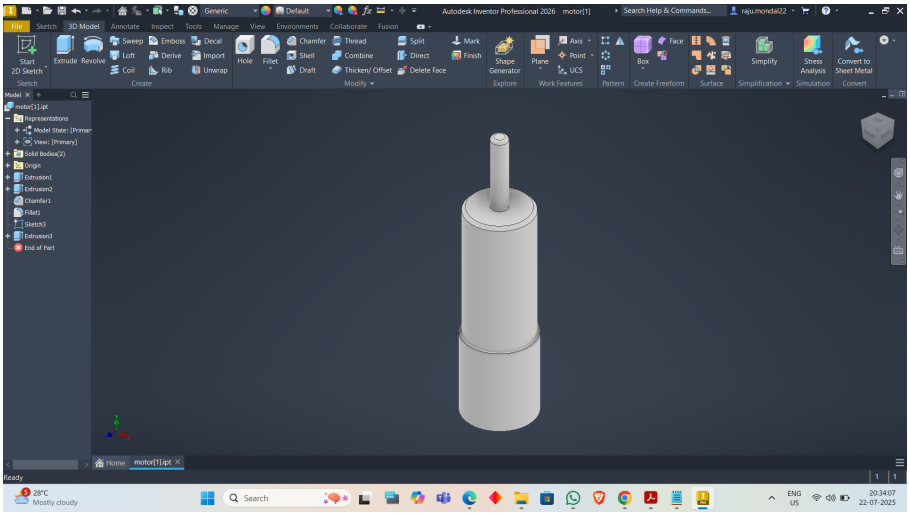

Figure 5: Motor

3.3 Assembly of the Mobile Robot Components

After the individual components were modeled in Autodesk Inventor and Fusion 360, the complete mechanical structure of the mobile robot was assembled virtually to verify design compatibility, alignment, and physical constraints. The assembly process was crucial for ensuring that all motor mounts, brackets, and chassis elements would fit together properly during physical fabrication.

Steps in the assembly process:

- Imported all individual components such as chassis base, motor brackets, wheel hubs, and electronics mounts into the assembly environment.
- Applied *mate*, *insert*, and *flush constraints* to fix the relative positions of parts.
- Verified alignment of wheel axles, bracket hole positions, and component spacing.
- Checked for interferences or collisions between parts, especially around motor housings and wiring channels.

Fusion 360 was used to generate an exploded view of the final assembly, which helped visualize the placement of electronic modules such as the Raspberry Pi and Arduino boards. The assembly also allowed the evaluation of center of gravity, weight distribution, and cable routing.

Final Assembly Validations:

- All mounting holes aligned correctly with standardized M3 and M4 bolts.
- Chassis provided balanced support for all motor and controller loads.

- Adequate clearance was confirmed for rotating components and wiring harnesses.

The assembly ensured that the transition from digital model to physical construction would proceed without unexpected interference or dimensional mismatches. This step served as the bridge between the design and fabrication phases of the project.

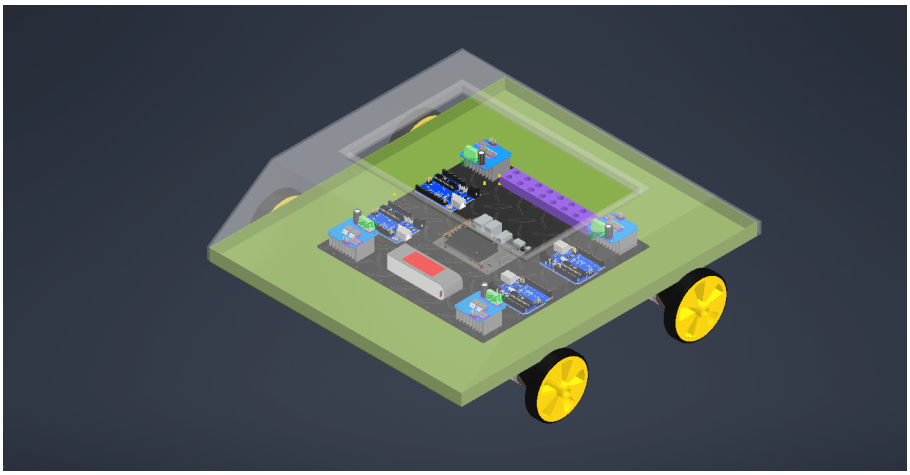

Figure 6: Assembly of the Mobile Robot Components

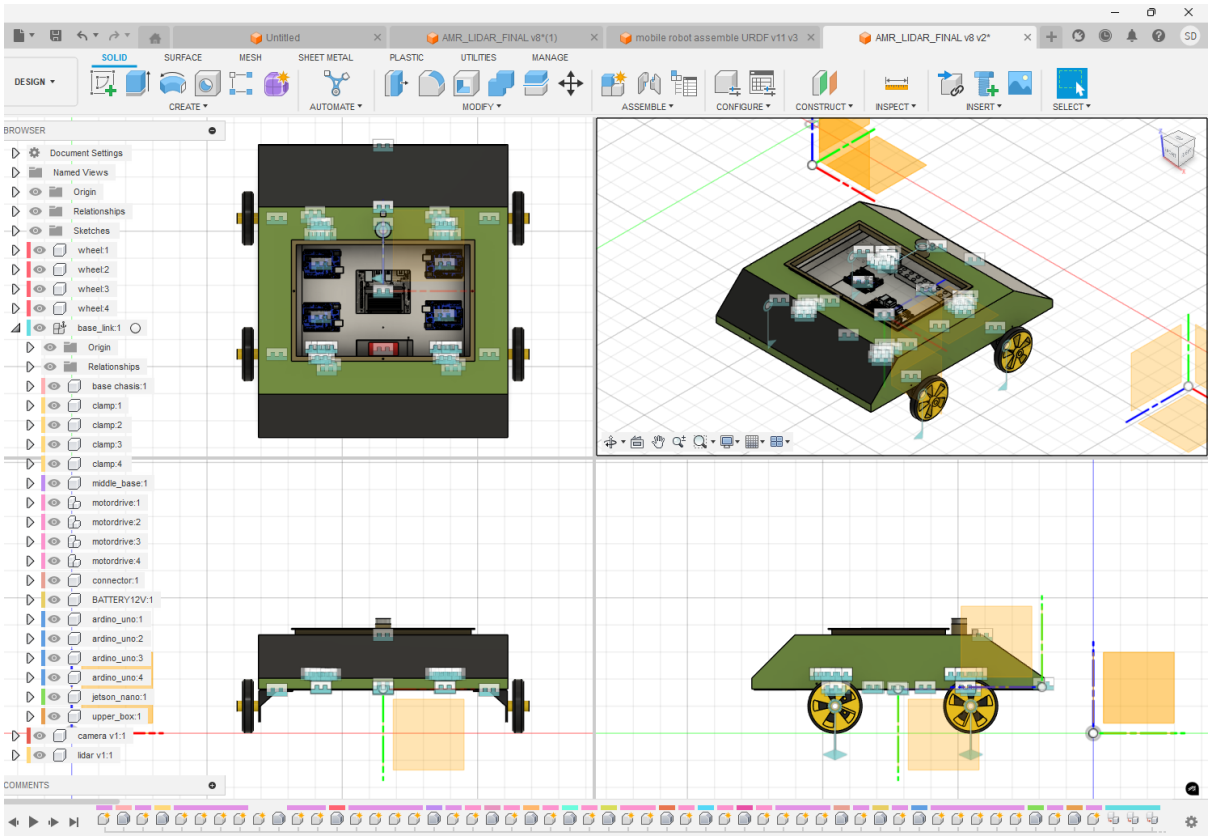

Figure 7: Another point of view of the assemble

3.4 2D Sketches and Dimensioning

In this project, 2D sketching and dimensioning were carried out using **Autodesk Inventor**, a parametric design environment ideal for mechanical modeling. The 2D sketches formed the foundation for the 3D part modeling process and were also used to define critical design constraints prior to extrusion or assembly.

Each part of the mobile robot — such as the chassis plate, motor brackets, and component mounts — was first developed in the Inventor sketch environment. Dimensions were applied using Inventor's fully-constrained sketching tools to ensure geometric accuracy and manufacturability.

Key steps followed during sketching in Autodesk Inventor:

- Created base sketches using construction lines for symmetry and alignment.
- Applied *geometric constraints* (parallel, perpendicular, colinear) to stabilize the sketch.
- Added *dimension constraints* (linear, angular, and radial) using Inventor's dimensioning tool to define the exact size and position of features.
- Used projected geometry and construction lines for aligning multiple sketches.

Dimensioned features included:

- Precise wheelbase and axle spacing.
- Hole positions for Arduino boards, Raspberry Pi, and motor drivers.
- Slot cutouts for cable routing and ventilation.

All 2D sketches were fully constrained to prevent unintentional distortion during part modeling. These sketches were later used for generating 3D parts, as well as for exporting `.dxf` files used in laser cutting or CNC fabrication.

The proper use of parametric constraints and dimensioning in Inventor ensured consistency, accuracy, and future modifiability of the design as the robot evolved.

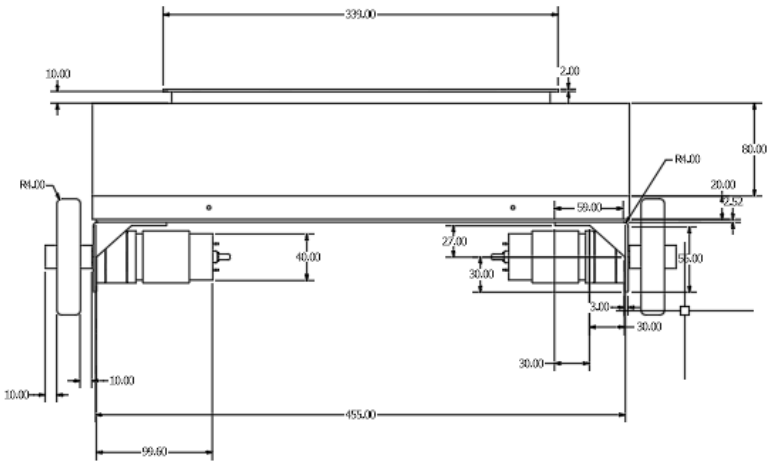

Figure 8: 2D sketch of first side view

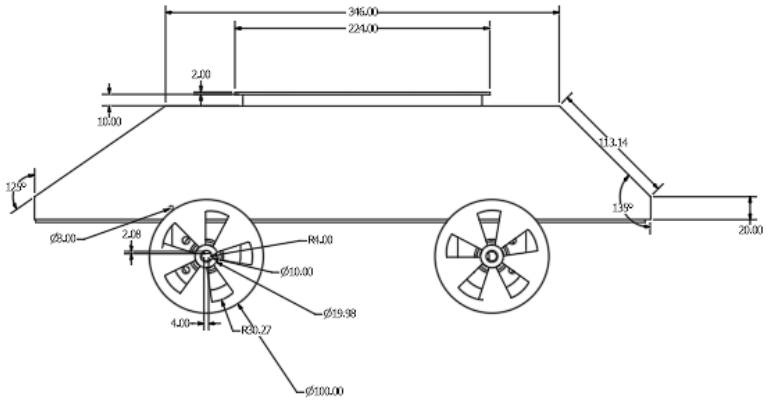

Figure 9: 2D sketch of second side view

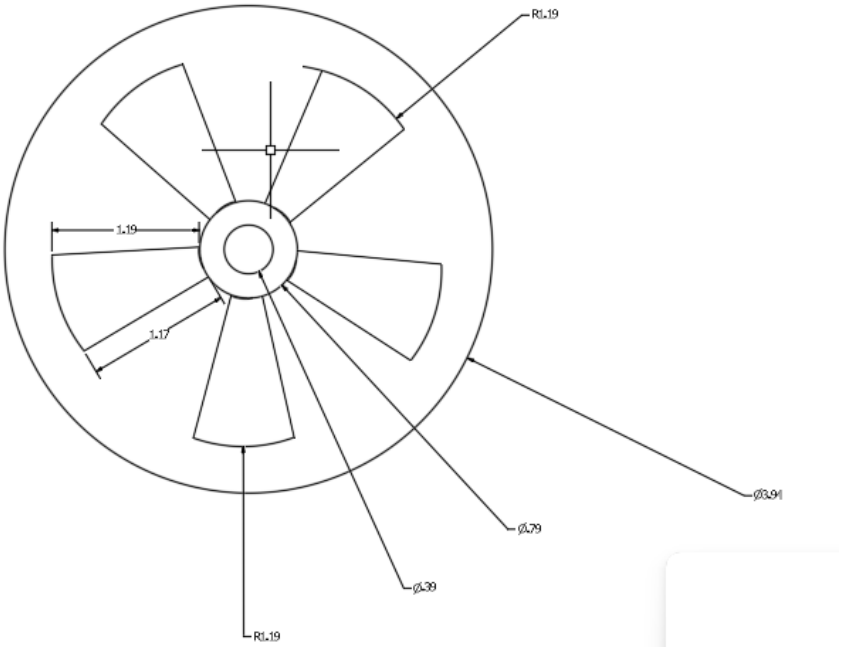

Figure 10: 2D sketch of a wheel

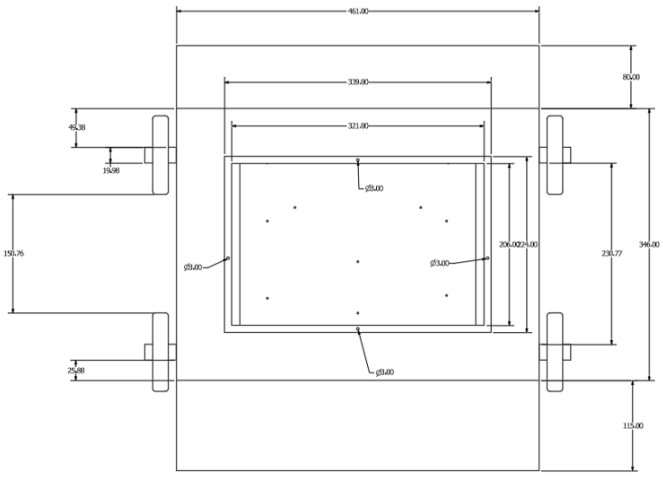

Figure 11: 2D sketch of a top view

Chapter 4

URDF Workspace Integration for ROS and ROS2

Unified Robot Description Format (URDF) is a key element in simulating and controlling robots within the ROS (Robot Operating System) environment. In this chapter, we describe the complete pipeline used to create and integrate a URDF model of the mobile robot, beginning with mechanical design in Autodesk Inventor and culminating in deployment within ROS and ROS2.

4.1 Workflow Overview

The URDF development followed this multi-step workflow:

1. **3D Modeling and Assembly in Autodesk Inventor**
2. **Exporting the assembly to Fusion 360**
3. **Using the URDF Exporter Plugin to generate .urdf, .xacro, and mesh files**
4. **Integrating the URDF into ROS and ROS2 workspaces for simulation and testing**

4.2 Modeling and Assembly in Autodesk Inventor

Each part of the mobile robot — including the chassis, wheels, motor mounts, and electronic modules — was modeled in Autodesk Inventor. These parts were then assembled into a single mechanical structure using Inventor's assembly environment.

Joints and mating constraints were added to replicate real-world behavior such as wheel rotation and motor alignment. The complete assembly was saved and prepared for transfer to Fusion 360.

4.3 Transferring to Fusion 360 for URDF Export

Fusion 360 was used to open and manage the Inventor assembly via the cloud-integrated Autodesk platform. Using the ROS URDF Exporter add-in for Fusion 360, the model was annotated with:

- Joint definitions (revolute, fixed, etc.)
- Link names and inertial properties
- Coordinate frames for each component

This plugin then exported a full URDF package containing:

- A URDF or XACRO file
- STL/mesh files for each part
- Necessary config files (e.g., launch and RViz setup)

4.4 URDF Validation and Simulation in ROS/ROS2

The generated URDF was imported into a ROS workspace and visualized using RViz to verify correctness. Adjustments were made to joint positions, coordinate frames, and inertial parameters to ensure accurate simulation.

The same URDF was also integrated into a ROS2 environment, allowing compatibility with modern robotic systems and enabling further tasks such as:

- Visual simulation using Gazebo
- Integration with sensor data (e.g., LiDAR, camera)
- Controller development and SLAM

Chapter 5

System Architecture of the Mobile Robot

The mobile robot's system architecture encompasses both the hardware and software subsystems required for motion control, communication, power distribution, and high-level decision making. This chapter outlines the architectural design that enables the robot to operate in manual, serial, and autonomous modes.

5.1 Overview of the System Architecture

The robot is built around a hierarchical master–slave structure:

- **Raspberry Pi (Master)**: Central controller responsible for high-level decision making, coordination, and communication with ROS/ROS2.
- **Four Arduino Uno Boards (Slaves)**: Each controls a single planetary-geared DC motor and driver module for one wheel.

The architecture supports three primary control modes:

- **Remote Control** – via Bluetooth/Wi-Fi input commands
- **Serial Communication** – between Raspberry Pi and Arduino over USB/UART
- **Autonomous Mode** – executing predefined paths or sensor-based navigation via ROS

5.2 Hardware Subsystems

Motor Subsystem: Each wheel is driven by a 12V planetary-geared DC motor connected to a motor driver (BTS7960 or equivalent). One Arduino Uno is assigned to each motor-driver pair, handling PWM signals and feedback (if available).

Control and Processing: The Raspberry Pi 4 serves as the processing unit running ROS/ROS2 nodes, interfacing with Arduino via serial communication (USB or UART). It also hosts code for motion commands, sensor data fusion, and path planning.

Power Management: A regulated 12V battery pack powers the motors, while a 5V regulator (buck converter) supplies the logic boards (Arduino and Raspberry Pi).

5.3 Communication Framework

- **I2C/UART/Serial** – Raspberry Pi communicates with each Arduino using individual serial ports or a shared bus.
- **ROS Topics** – High-level commands such as velocity or trajectory are published from ROS nodes.
- **Command Parsing** – Each Arduino receives motion commands (e.g., velocity) and translates them into PWM signals for its respective motor.

5.4 Software Modules

- **ROS Node:** Publishes commands to Arduino based on joystick input, autonomous path, or sensor feedback.
- **Arduino Firmware:** Receives serial commands, applies motor control logic (PWM generation, direction control).
- **Motion Logic:** Converts high-level velocity commands to wheel-specific motor signals.

5.5 Architecture Diagram

A block diagram of the system architecture is shown in Figure 12, each motor driver is controlled by an individual Arduino.

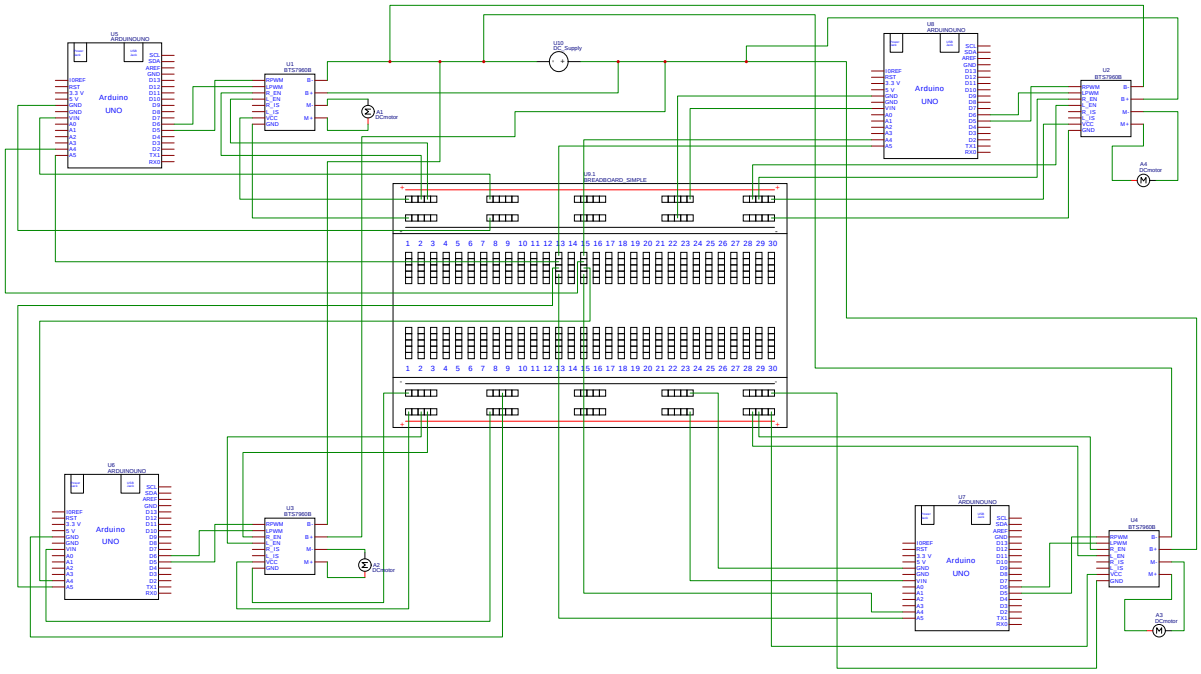

Figure 12: Wiring diagram of the motor control system using Arduino and BTS7960 drivers

Chapter 6

Control Modes and Operation Strategies

The control system of the mobile robot supports multiple operational modes to enable flexible interaction, testing, and deployment. These modes include:

1. Serial Control
2. Autonomous Control (Forward, Reverse, Turn)
3. Manual Control

Each of these is described in detail below, along with flowcharts representing their operational logic.

6.1 Serial Control Mode

In serial control mode, the Raspberry Pi (master) communicates with four Arduino Uno boards (slaves) using UART/USB-based serial communication. Commands such as velocity or direction are transmitted in a structured format to each slave. Each Arduino is responsible for controlling a single motor-driver pair.

Serial Command Format:

- Format: `M1:100; M2:100; M3:100; M4:100`
- Each tag (M1–M4) corresponds to a motor, and the number denotes PWM value or direction.

Execution Flow:

1. Raspberry Pi computes the desired motion (e.g., forward at speed X).
2. Commands are serialized and sent to each Arduino.
3. Each Arduino parses the command and generates appropriate PWM and direction signals.
4. BTS7960 driver applies power to the motor based on command.

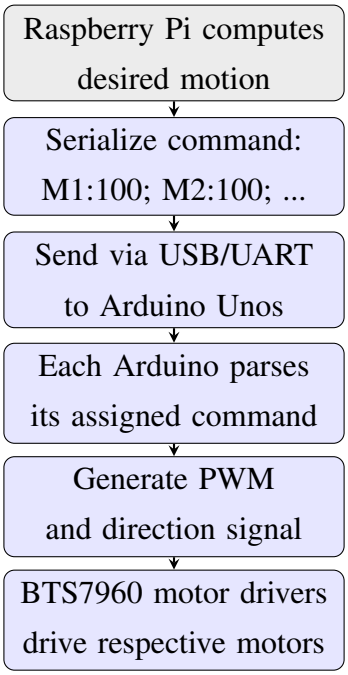

Figure 13: Corrected Flowchart of Serial Control Mode Operation

6.2 Autonomous Control Mode (Forward, Reverse, Left, Right)

In this mode, the robot follows predefined instructions without real-time user input. This is useful for line-following, obstacle avoidance, or simple trajectory execution.

Motion Commands:

- **Forward**: All four motors rotate in the same direction.
- **Reverse**: All four motors rotate in the opposite direction.

- **Left Turn**: Left side motors slow down or reverse; right side motors move forward.
- **Right Turn**: Right side motors slow down or reverse; left side motors move forward.

Execution Logic:

- A command is hardcoded or sensor-triggered.
- A logic block maps motion type to motor states.
- PWM signals and H-Bridge control logic are applied accordingly.

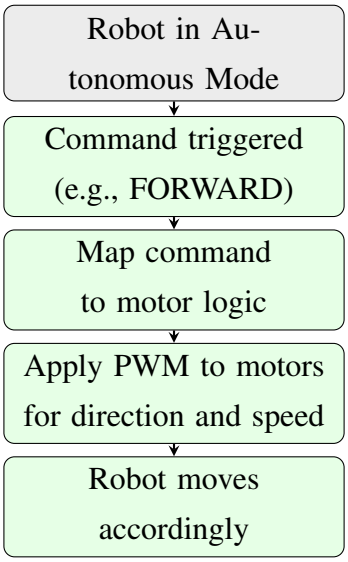

Figure 14: Flowchart of Autonomous Control Logic

6.3 Manual Control Mode

Manual control allows the user to operate the robot in real time using a joystick, mobile device, or terminal input. This mode is essential for direct navigation, testing, and interactive control during development or demonstration.

Input Interfaces:

- **Bluetooth module (e.g., HC-05)** paired with a mobile app.
- **Wi-Fi or SSH-based terminal** using commands sent over serial.
- **Keyboard or joystick input** interpreted via Raspberry Pi and ROS.

Command Mapping:

- 'F' – Move Forward

- 'B' – Move Backward
- 'L' – Turn Left
- 'R' – Turn Right
- 'S' – Stop All Motors

Control Workflow:

1. The user sends input (e.g., pressing a button on the mobile app).
2. Raspberry Pi reads the input and translates it to motor commands.
3. The command is sent via serial communication to each Arduino Uno.
4. Each Arduino parses the command and adjusts its assigned motor through the BTS7960 driver.

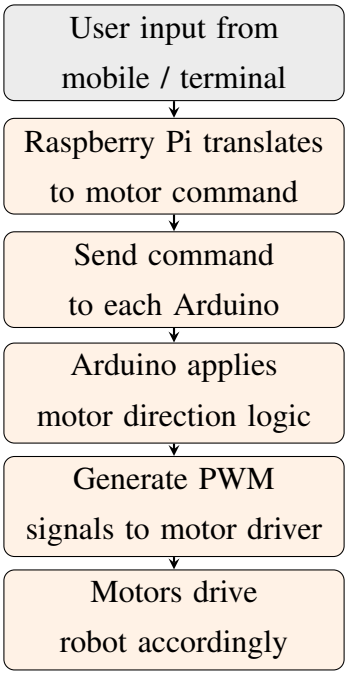

Figure 15: Corrected Flowchart of Manual Control Mode Operation

Chapter 7

Conclusion

This project successfully achieved the design, development, and implementation of a multi-mode mobile robot system using a distributed control architecture comprising a Raspberry Pi and four Arduino Uno boards. The robot was engineered with a robust drive system powered by four orange planetary geared DC motors, each independently controlled by a BTS7960 motor driver and its corresponding Arduino. A layered control framework was established, allowing operation through three distinct modes: serial communication, autonomous navigation, and manual user control.

In the serial control mode, the Raspberry Pi acted as a master, orchestrating movement commands over UART to each Arduino slave. This setup demonstrated seamless parallel command execution and effective low-level actuation, highlighting the scalability and modularity of the architecture. The command structure and synchronization logic were successfully verified through continuous testing and serial monitoring.

Autonomous control capability was embedded into the robot's system to enable directional movement such as forward, reverse, left, and right turns. By mapping predefined logic and PWM signals to motor actions, the robot exhibited stable and predictable autonomous movement. This mode reflects the feasibility of integrating sensor-based or ROS-based intelligence in future extensions.

Manual control was implemented using external inputs from a terminal or mobile interface via Bluetooth or Wi-Fi. This mode allowed real-time user-driven interaction with the robot, which proved particularly effective during testing and live demonstrations.

The physical structure of the robot was meticulously developed using Autodesk Inventor and Fusion 360. 3D models and 2D sketches facilitated precise fabrication and assembly. Furthermore, the robot's URDF model was exported and validated within ROS and ROS 2 environments, establishing a foundational framework for simulation, visualization, and advanced robotics integration.

Overall, this project bridges the gap between theoretical modeling and practical deployment. It provides an end-to-end implementation — from hardware design and CAD modeling to embedded programming and multi-modal control. The modular, scalable approach ensures the system can be adapted for more complex applications in swarm robotics, autonomous navigation, and intelligent surveillance. With minor upgrades such as sensor integration or path planning algorithms, this mobile robot can serve as a versatile base for research, academic prototyping, or industrial automation tasks.

References

[1] R Siegwart, IR Nourbakhsh, D Scaramuzza, Introduction to Autonomous Mobile Robots, MIT Press, USA, 2011.

[2] SG Tzafestas, Introduction to Mobile Robot Control, Elsevier, USA, 2014.

[3] A Kelly, Mobile Robotics: Mathematics, Models, and Methods, Cambridge University Press, USA, 2013.

Appendix: Arduino Control Codes

This appendix includes the motor control Arduino sketches used for testing and controlling the Mobile Robot.

Master Slave Automative FRLR Controller

```
#include <Wire.h>

#define RPWM 5
#define LPWM 6
#define REN 11
#define LEN 12

void setup() {
  Wire.begin(); // Master mode
  Serial.begin(9600);


  pinMode(RPWM, OUTPUT);
  pinMode(LPWM, OUTPUT);
  pinMode(REN, OUTPUT);
  pinMode(LEN, OUTPUT);
  digitalWrite(REN, HIGH);
  digitalWrite(LEN, HIGH);
}

void loop() {


   sendToAllSlaves("Reverse,30");
  runMasterMotor("Reverse", 30);
  delay(3000);


  sendToAllSlaves("Stop,0");
  runMasterMotor("Stop", 0);
  delay(2000);


  sendToIndividualSlaves("Forward,50", 1);
  sendToIndividualSlaves("Forward,50", 2);
  sendToIndividualSlaves("Reverse,50", 3);
```

```
  runMasterMotor("Reverse",50);
  delay(1400); // Adjust for ~90-degree turn

  sendToAllSlaves("Stop,0");
  runMasterMotor("Stop", 0);
  delay(2000);

    sendToAllSlaves("Reverse,30");
  runMasterMotor("Reverse", 30);
  delay(2000);

sendToAllSlaves("Stop,0");
  runMasterMotor("Stop", 0);
  delay(2000);

   sendToIndividualSlaves("Reverse,50", 1);
  sendToIndividualSlaves("Reverse,50", 2);
  sendToIndividualSlaves("Forward,50", 3);
  runMasterMotor("Forward",50);
  delay(1400);


sendToAllSlaves("Stop,0");
  runMasterMotor("Stop", 0);
  delay(2000);

  sendToAllSlaves("Forward,30");
  runMasterMotor("Forward", 30);
  delay(2000);

sendToAllSlaves("Stop,0");
  runMasterMotor("Stop", 0);



}

// Broadcast to all slaves
void sendToAllSlaves(String command) {
  for (int addr = 1; addr <= 3; addr++) {
    Wire.beginTransmission(addr);
    Wire.write(command.c_str());
    Wire.endTransmission();
    delay(5);
  }
```

```
}

// Send to specific slave
void sendToIndividualSlaves(String command, int addr) {
  Wire.beginTransmission(addr);
  Wire.write(command.c_str());
  Wire.endTransmission();
  delay(5);
}

void runMasterMotor(String direction, int pwm) {
  pwm = constrain(pwm, 0, 255);
  if (direction == "Forward") {
    analogWrite(RPWM, pwm);
    analogWrite(LPWM, 0);
  } else if (direction == "Reverse") {
    analogWrite(RPWM, 0);
    analogWrite(LPWM, pwm);
  } else {
    analogWrite(RPWM, 0);
    analogWrite(LPWM, 0);
  }
}
```

Listing 1: Arduino Code for Master

```
#include <Wire.h>

#define RPWM 5
#define LPWM 6
#define REN 7
#define LEN 8

String command = "";

void setup() {
  Wire.begin(1);
  Wire.onReceive(receiveCommand);

  pinMode(RPWM, OUTPUT);
  pinMode(LPWM, OUTPUT);
  pinMode(REN, OUTPUT);
  pinMode(LEN, OUTPUT);
  digitalWrite(REN, HIGH);
  digitalWrite(LEN, HIGH);
```

```
}

void loop() {
  // Nothing here needed
}

void receiveCommand(int howMany) {
  command = "";
  while (Wire.available()) {
    char c = Wire.read();
    command += c;
  }

  // Parse command
  String direction = command.substring(0, command.indexOf(','));
  int pwm = command.substring(command.indexOf(',') + 1).toInt();
  pwm = constrain(pwm, 0, 255);

  if (direction == "Forward") {
    analogWrite(RPWM, pwm);
    analogWrite(LPWM, 0);
  } else if (direction == "Reverse") {
    analogWrite(RPWM, 0);
    analogWrite(LPWM, pwm);
  } else { // Stop or any unknown command
    analogWrite(RPWM, 0);
    analogWrite(LPWM, 0);
  }
}
```

Listing 2: Arduino Code for Slave Motor Controller 1 ,Controller 2 And Controller 3

Master Slave Automative FR Timer

```
#include <Wire.h>


#define RPWM 5
#define LPWM 6
#define REN 11
#define LEN 12

void setup() {
  Wire.begin(); // Master mode
```

```
  Serial.begin(9600);


  pinMode(RPWM, OUTPUT);
  pinMode(LPWM, OUTPUT);
  pinMode(REN, OUTPUT);
  pinMode(LEN, OUTPUT);
  digitalWrite(REN, HIGH);
  digitalWrite(LEN, HIGH);
}

void loop() {
  // 1. Forward 200 PWM
  sendToAllSlaves("Forward,50");
  runMasterMotor("Forward", 50);
  delay(5000);  // Run for 5 sec

  // 2. Reverse 50 PWM
  sendToAllSlaves("Reverse,50");
  runMasterMotor("Reverse", 50);
  delay(4000);  // Run for 4 sec

  // 3. Stop
  sendToAllSlaves("Stop,0");
  runMasterMotor("Stop", 0);
  delay(3000);  // Wait 3 sec
}

void sendToAllSlaves(String command) {
  for (int addr = 1; addr <= 3; addr++) {
    Wire.beginTransmission(addr);
    Wire.write(command.c_str());
    Wire.endTransmission();
    delay(5); // Small delay between commands
  }
}

void runMasterMotor(String direction, int pwm) {
  pwm = constrain(pwm, 0, 255);
  if (direction == "Forward") {
    analogWrite(RPWM, pwm);
    analogWrite(LPWM, 0);
  } else if (direction == "Reverse") {
    analogWrite(RPWM, 0);
    analogWrite(LPWM, pwm);
```

```
  } else {
    analogWrite(RPWM, 0);
    analogWrite(LPWM, 0);
  }
}
```

Listing 3: Arduino Code for Master Motor i2c Controller

```
#include <Wire.h>

#define RPWM 5
#define LPWM 6
#define REN 7
#define LEN 8

String command = "";

void setup() {
  Wire.begin(2);
  Wire.onReceive(receiveCommand);

  pinMode(RPWM, OUTPUT);
  pinMode(LPWM, OUTPUT);
  pinMode(REN, OUTPUT);
  pinMode(LEN, OUTPUT);
  digitalWrite(REN, HIGH);
  digitalWrite(LEN, HIGH);
}

void loop() {
  // Nothing here needed
}

void receiveCommand(int howMany) {
  command = "";
  while (Wire.available()) {
    char c = Wire.read();
    command += c;
  }

  // Parse command
  String direction = command.substring(0, command.indexOf(','));
  int pwm = command.substring(command.indexOf(',') + 1).toInt();
  pwm = constrain(pwm, 0, 255);

```

```
  if (direction == "Forward") {
    analogWrite(RPWM, pwm);
    analogWrite(LPWM, 0);
  } else if (direction == "Reverse") {
    analogWrite(RPWM, 0);
    analogWrite(LPWM, pwm);
  } else { // Stop or any unknown command
    analogWrite(RPWM, 0);
    analogWrite(LPWM, 0);
  }
}
```

Listing 4: Arduino Code for Slave Motor Controller 1 ,Controller 2 And Controller 3

Master Slave Manual Controller

```
#include <Wire.h>

// Master Motor Pins (Right Rear Motor)
#define RPWM 5
#define LPWM 6
#define REN 11
#define LEN 12

// Receiver Channels
#define THROTTLE_PIN A0
#define STEERING_PIN A1
void sendToAllSlaves(String direction, int pwm = 0);

void setup() {
  Wire.begin();  // I2C Master
  Serial.begin(9600);

  pinMode(RPWM, OUTPUT);
  pinMode(LPWM, OUTPUT);
  pinMode(REN, OUTPUT);
  pinMode(LEN, OUTPUT);
  digitalWrite(REN, HIGH);
  digitalWrite(LEN, HIGH);

  pinMode(THROTTLE_PIN, INPUT);
  pinMode(STEERING_PIN, INPUT);
}

```

```
void loop() {
  int throttle = pulseIn(THROTTLE_PIN, HIGH, 50000);
  int steering = pulseIn(STEERING_PIN, HIGH, 50000);

  if (throttle < 1000 || throttle > 2000 || steering < 1000 || steering > 2000) {
    stopAllMotors();
    return;
  }

  // Auto-Trigger Left/Right Turn
  if (steering < 1100) {
    autoLeftTurn();
    return;
  } else if (steering > 1900) {
    autoRightTurn();
    return;
  }

  int throttle_offset = throttle - 1500;
  int steering_offset = steering - 1500;

  if (abs(throttle_offset) < 20) throttle_offset = 0;
  if (abs(steering_offset) < 20) steering_offset = 0;

  int left_speed = throttle_offset - (steering_offset * 0.5);
  int right_speed = throttle_offset + (steering_offset * 0.5);

  left_speed = constrain(left_speed, -255, 255);
  right_speed = constrain(right_speed, -255, 255);

  String left_dir = (left_speed > 10) ? "Forward" : (left_speed < -10 ? "Reverse" : "Stop");
  String right_dir = (right_speed > 10) ? "Forward" : (right_speed < -10 ? "Reverse" : "Stop");

  int left_pwm = abs(left_speed);
  int right_pwm = abs(right_speed);


  runMasterMotor(right_dir, right_pwm);


  sendToSlave(1, left_dir, left_pwm);
  sendToSlave(2, left_dir, left_pwm);
```

```
  sendToSlave(3, right_dir, right_pwm);

  delay(50);
}

void stopAllMotors() {
  runMasterMotor("Stop", 0);
  sendToAllSlaves("Stop", 0);
}

void runMasterMotor(String direction, int pwm) {
  pwm = constrain(pwm, 0, 255);
  if (direction == "Forward") {
    analogWrite(RPWM, pwm);
    analogWrite(LPWM, 0);
  } else if (direction == "Reverse") {
    analogWrite(RPWM, 0);
    analogWrite(LPWM, pwm);
  } else {
    analogWrite(RPWM, 0);
    analogWrite(LPWM, 0);
  }
}

void sendToSlave(int address, String direction, int pwm) {
  String command = direction + "," + String(pwm);
  Wire.beginTransmission(address);
  Wire.write(command.c_str());
  Wire.endTransmission();
  delay(5);
}

void sendToAllSlaves(String direction, int pwm = 0) {
  for (int addr = 1; addr <= 3; addr++) {
    sendToSlave(addr, direction, pwm);
  }
}

void sendToIndividualSlaves(String direction, int addr) {
  Wire.beginTransmission(addr);
  Wire.write(direction.c_str());
  Wire.endTransmission();
  delay(5);
}

```

```
116 // Auto Turn Left Routine
117 void autoLeftTurn() {
118   Serial.println("Auto LEFT turn");
119 
120   sendToAllSlaves("Stop", 0);
121   runMasterMotor("Stop", 0);
122   delay(2000);
123 
124   sendToAllSlaves("Reverse", 30);
125   runMasterMotor("Reverse", 30);
126   delay(2000);
127 
128   sendToAllSlaves("Stop", 0);
129   runMasterMotor("Stop", 0);
130   delay(2000);
131 
132   sendToIndividualSlaves("Reverse,50", 1);
133   sendToIndividualSlaves("Reverse,50", 2);
134   sendToIndividualSlaves("Forward,50", 3);
135   runMasterMotor("Forward", 50);
136   delay(1400);
137 
138   sendToAllSlaves("Stop", 0);
139   runMasterMotor("Stop", 0);
140   delay(2000);
141 
142   sendToAllSlaves("Forward", 30);
143   runMasterMotor("Forward", 30);
144   delay(2000);
145 
146   sendToAllSlaves("Stop", 0);
147   runMasterMotor("Stop", 0);
148 }
149 
150 //Auto Turn Right Routine (Same as Left, or modify if needed)
151 void autoRightTurn() {
152   Serial.println("Auto RIGHT turn");
153 
154   sendToAllSlaves("Stop", 0);
155   runMasterMotor("Stop", 0);
156   delay(2000);
157 
158   sendToAllSlaves("Reverse", 30);
159   runMasterMotor("Reverse", 30);
160   delay(2000);
```

```
161
162   sendToAllSlaves("Stop", 0);
163   runMasterMotor("Stop", 0);
164   delay(2000);
165
166   sendToIndividualSlaves("Forward,50", 1);
167   sendToIndividualSlaves("Forward,50", 2);
168   sendToIndividualSlaves("Reverse,50", 3);
169   runMasterMotor("Reverse", 50);
170   delay(1400);
171
172   sendToAllSlaves("Stop", 0);
173   runMasterMotor("Stop", 0);
174   delay(2000);
175
176   sendToAllSlaves("Forward", 30);
177   runMasterMotor("Forward", 30);
178   delay(2000);
179
180   sendToAllSlaves("Stop", 0);
181   runMasterMotor("Stop", 0);
182 }
```

Listing 5: Arduino Code for Master Controller

```
1  #include <Wire.h>
2
3  #define RPWM 5
4  #define LPWM 6
5  #define REN 7
6  #define LEN 8
7
8  String command = "";
9
10 void setup() {
11   Wire.begin(1);
12   Wire.onReceive(receiveCommand);
13
14   pinMode(RPWM, OUTPUT);
15   pinMode(LPWM, OUTPUT);
16   pinMode(REN, OUTPUT);
17   pinMode(LEN, OUTPUT);
18   digitalWrite(REN, HIGH);
19   digitalWrite(LEN, HIGH);
20 }
```

```
21
22 void loop() {
23   // Nothing here needed
24 }
25
26 void receiveCommand(int howMany) {
27   command = "";
28   while (Wire.available()) {
29     char c = Wire.read();
30     command += c;
31   }
32
33   // Parse command
34   String direction = command.substring(0, command.indexOf(','));
35   int pwm = command.substring(command.indexOf(',') + 1).toInt();
36   pwm = constrain(pwm, 0, 255);
37
38   if (direction == "Forward") {
39     analogWrite(RPWM, pwm);
40     analogWrite(LPWM, 0);
41   } else if (direction == "Reverse") {
42     analogWrite(RPWM, 0);
43     analogWrite(LPWM, pwm);
44   } else { // Stop or any unknown command
45     analogWrite(RPWM, 0);
46     analogWrite(LPWM, 0);
47   }
48 }
```

Listing 6: Arduino Code for Slave Motor Controller 1, controller 2 And controller 3

Master Slave Serial Motor Controller

```
1
2
3 /*
4  * I2C Master Motor Controller
5  *
6  * This code controls one motor (Motor 4) directly using BTS7960
7  * and communicates with slave Arduinos to control Motor 1, 2, and 3 over
       I2C.
8  *
9  * Commands are sent over Serial in the following format:
10 *     M1Forward M2Stop M3Reverse M4Forward
```

```
 *
 */

#include <Wire.h>

String readString;

// Motor 4 (connected to Master) pin definitions
#define RPWM 5
#define LPWM 6
#define REN 11
#define LEN 12

void setup() {
  Wire.begin();               // Start I2C communication as Master
  Serial.begin(9600);         // Start serial monitor
  Serial.println("Enter commands like: M1Forward M2Stop M3Reverse M4Forward
      ");

  // Setup Motor 4 pins (BTS7960 Driver)
  pinMode(RPWM, OUTPUT);
  pinMode(LPWM, OUTPUT);
  pinMode(REN, OUTPUT);
  pinMode(LEN, OUTPUT);

  digitalWrite(REN, HIGH);    // Enable BTS7960 inputs
  digitalWrite(LEN, HIGH);
}

void loop() {
  // Collect serial input
  while (Serial.available()) {
    delay(2);
    char c = Serial.read();
    readString += c;
  }

  if (readString.length() > 0) {
    readString.trim();
    Serial.println("Received: " + readString);

    // Break the input into individual motor commands
    while (readString.length() > 0) {
      int spaceIndex = readString.indexOf(' ');
      String cmd;
```

```
55
56        if (spaceIndex != -1) {
57          cmd = readString.substring(0, spaceIndex);
58          readString = readString.substring(spaceIndex + 1);
59          readString.trim();
60        } else {
61          cmd = readString;
62          readString = "";
63        }
64
65        if (cmd.length() > 0) {
66          processCommand(cmd);
67        }
68      }
69    }
70  }
71
72  // Processes command string like "M1Forward"
73  void processCommand(String cmd) {
74    int motorNum = 0;
75    String action = "";
76
77    if (cmd.startsWith("M1")) { motorNum = 1; action = cmd.substring(2); }
78    else if (cmd.startsWith("M2")) { motorNum = 2; action = cmd.substring(2);
          }
79    else if (cmd.startsWith("M3")) { motorNum = 3; action = cmd.substring(2);
          }
80    else if (cmd.startsWith("M4")) { motorNum = 4; action = cmd.substring(2);
          }
81
82    // Send to slave motors (M1 to M3)
83    if (motorNum >= 1 && motorNum <= 3) {
84      Wire.beginTransmission(motorNum);
85      Wire.write(action.c_str());
86      Wire.endTransmission();
87      Serial.println("Sent to Slave M" + String(motorNum) + ": " + action);
88    }
89    // Apply command to Master Motor (M4)
90    else if (motorNum == 4) {
91      applyM4Command(action);
92      Serial.println("Executed on Master M4: " + action);
93    }
94  }
95
96  // Execute action (Forward, Reverse, Stop) on Motor 4
```

```
void applyM4Command(String command) {
  if (command == "Forward") {
    analogWrite(RPWM, 200);
    analogWrite(LPWM, 0);
  } else if (command == "Reverse") {
    analogWrite(RPWM, 0);
    analogWrite(LPWM, 200);
  } else if (command == "Stop") {
    analogWrite(RPWM, 0);
    analogWrite(LPWM, 0);
  }
}
```

Listing 7: Arduino Code for Master 12C Motor Controller

```
#include <Wire.h>

#define RPWM 5
#define LPWM 6
#define REN 7
#define LEN 8

String command = "";

void setup() {
  Wire.begin(1);
  Wire.onReceive(receiveCommand);

  pinMode(RPWM, OUTPUT);
  pinMode(LPWM, OUTPUT);
  pinMode(REN, OUTPUT);
  pinMode(LEN, OUTPUT);
  digitalWrite(REN, HIGH);
  digitalWrite(LEN, HIGH);
}

void loop() {
  // Nothing here needed
}

void receiveCommand(int howMany) {
  command = "";
  while (Wire.available()) {
    char c = Wire.read();
    command += c;
```

```
  }

  // Parse command
  String direction = command.substring(0, command.indexOf(','));
  int pwm = command.substring(command.indexOf(',') + 1).toInt();
  pwm = constrain(pwm, 0, 255);

  if (direction == "Forward") {
    analogWrite(RPWM, pwm);
    analogWrite(LPWM, 0);
  } else if (direction == "Reverse") {
    analogWrite(RPWM, 0);
    analogWrite(LPWM, pwm);
  } else { // Stop or any unknown command
    analogWrite(RPWM, 0);
    analogWrite(LPWM, 0);
  }
}
```

Listing 8: Arduino Code for Slave Motor Controller 1,controller 2 And controller 3