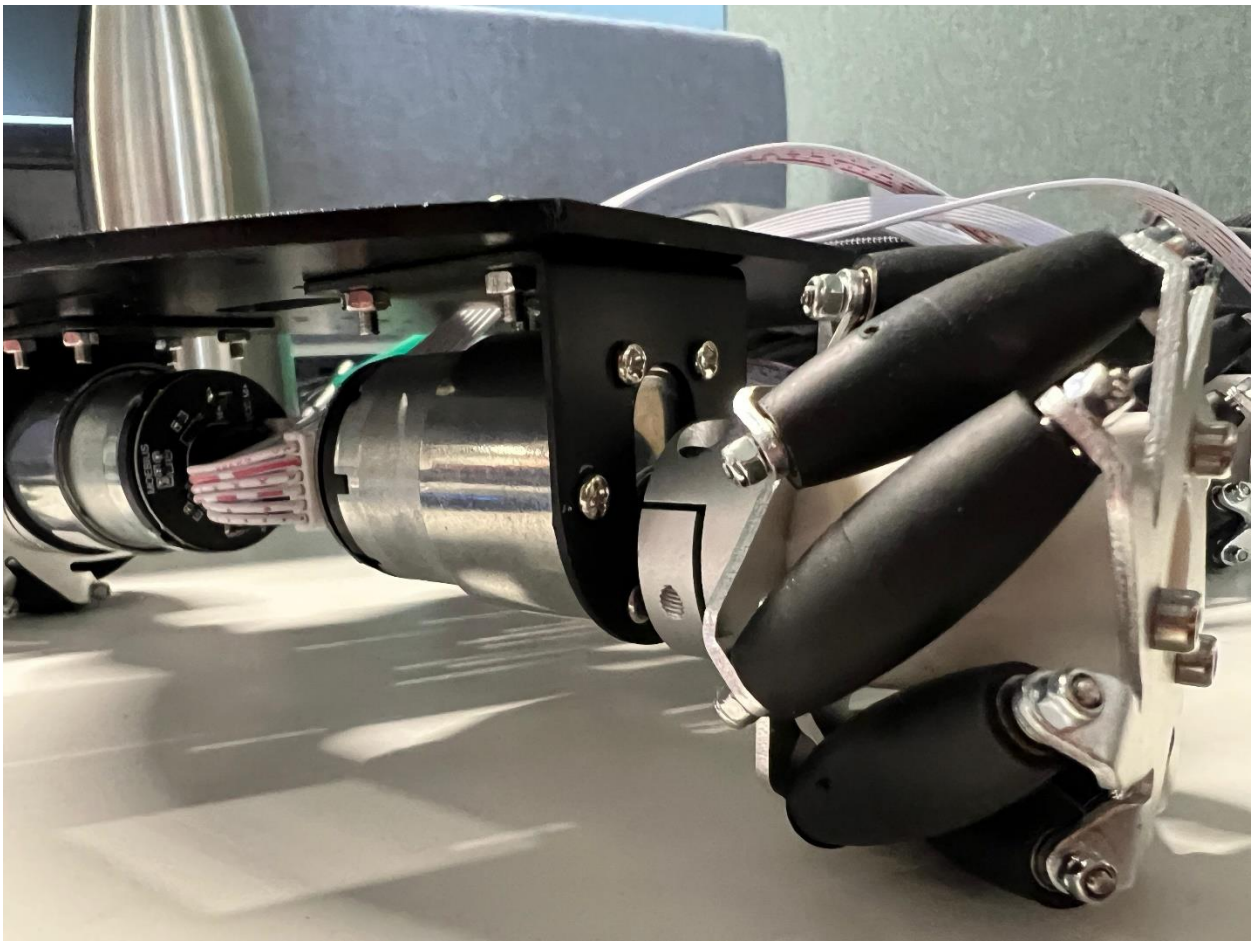
Smart Car using Bluetooth & Dabble App



Author: Paras Khosla

Department: Information & Communication Technology

Date:29-05-2024

Table of Contents

Table of Figures

Project Overview

This report documents the development and functionality algorithm of a Bluetooth-controlled 4-wheel drive car. The car uses four DC motors, controlled via an L298N motor driver and an ESP32 microcontroller. The car can be operated using the Dabble mobile application, which provides a gamepad interface for user control.

Components Used

- ESP32 Microcontroller
- L298N Motor Driver
- 4 DC Motors
- Breadboard and Jumper Wires
- Dabble Mobile Application

Hardware Setup

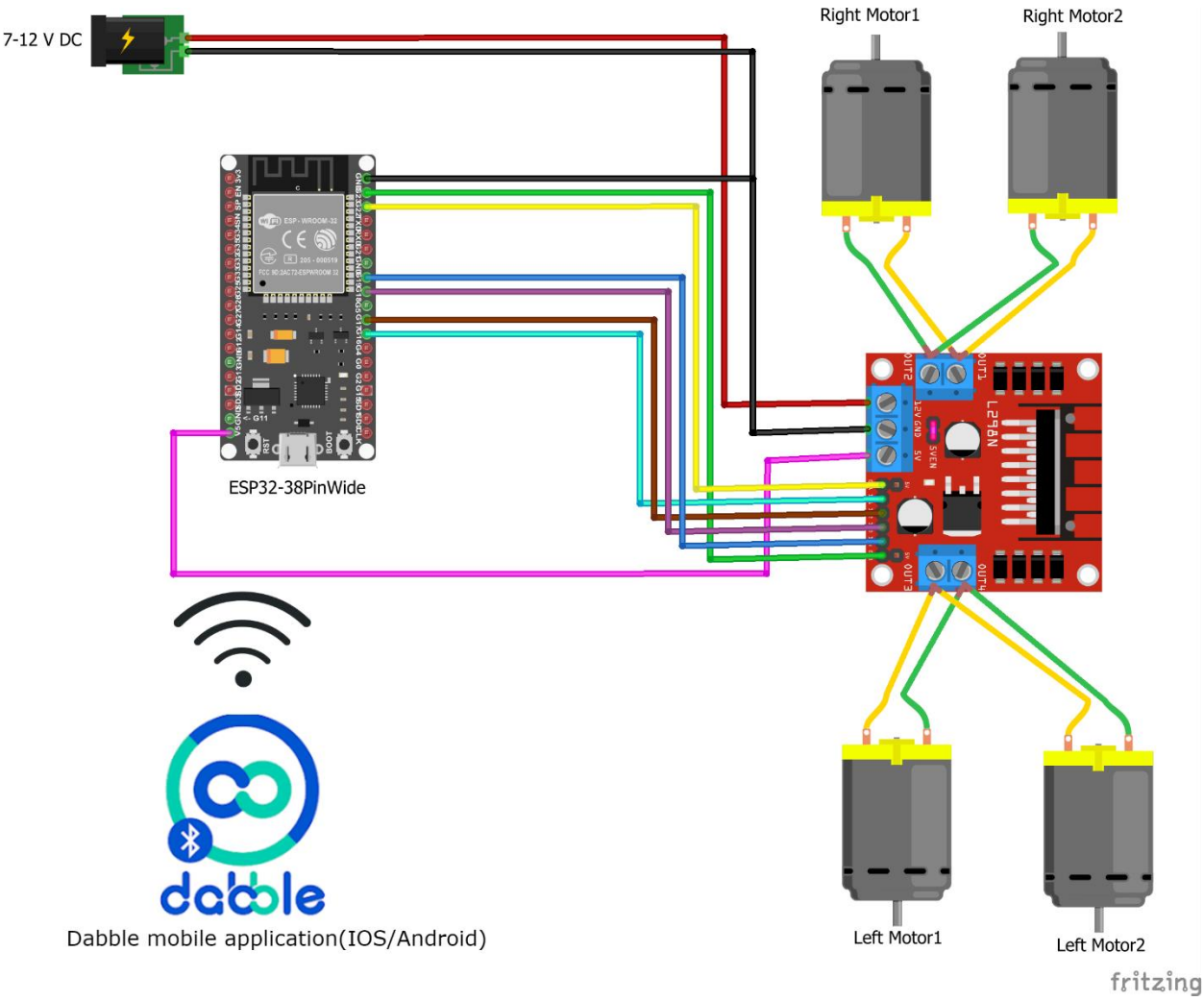


Figure 1: Circuit diagram with all 4 motors.

Microcontroller: ESP32

The ESP32 microcontroller is a powerful and versatile board that supports Bluetooth connectivity, making it suitable for this project.

Motor Driver: L298N

The L298N motor driver is used to control the direction and speed of DC motors. It can drive two motors independently, and by using two channels, we can control four motors.

DC Motors

Four DC motors are used to drive the car. These motors are connected to the motor driver and are controlled using PWM (Pulse Width Modulation) signals.

Breadboard and Jumper Wires

A breadboard and jumper wires are used to establish connections between the ESP32, the motor driver, and the motors.

Dabble Mobile Application

The Dabble app, available on the Play Store and App Store, is used for Bluetooth communication. It provides a gamepad interface to control the car.

Pin Configuration

The pin configuration for the motors is defined in the code as follows:

- **Right Motor:**
  - rightMotorPin1 (Pin 22)
  - rightMotorPin2 (Pin 23)
- **Left Motor:**
  - leftMotorPin1 (Pin 18)
  - leftMotorPin2 (Pin 19)

These pins are connected to the L298N motor driver, which in turn drives the motors.

PWM Configuration

PWM (Pulse Width Modulation) is used to control the speed of the motors. The configuration is as follows:

- **PWM Frequency**: 1 KHz
- **PWM Resolution**: 8 bits
- **Right Motor PWM Channel**: 4
- **Left Motor PWM Channel**: 5

PWM allows the motor speed to be varied by adjusting the duty cycle of the signal sent to the motors.

Software Implementation

Libraries and Definitions

```
#include <Arduino.h>
#define CUSTOM_SETTINGS
#define INCLUDE_GAMEPAD_MODULE
#include <DabbleESP32.h>
```

Figure 2: Included Libraries.

- **Arduino.h**: The main library for Arduino functions.
- **DabbleESP32.h**: Library for interfacing with the Dabble app.

Pin Definitions and Constants

```
// Define pins for right motor
int rightMotorPin1 = 22;
int rightMotorPin2 = 23;

// Define pins for left motor
int leftMotorPin1 = 18;
int leftMotorPin2 = 19;

#define MAX_MOTOR_SPEED 255  // Maximum motor speed

const int PWMFreq = 1000;  // PWM frequency (1 KHz)
const int PWMResolution = 8;  // PWM resolution (8 bits)
const int rightMotorPWMSpeedChannel = 4;  // PWM channel for right motor speed
const int leftMotorPWMSpeedChannel = 5;  // PWM channel for left motor speed
```

Figure 3: Pins & variable definitions.

- **rightMotorPin1, rightMotorPin2**: Pins for controlling the right motor.
- **leftMotorPin1, leftMotorPin2**: Pins for controlling the left motor.
- **MAX_MOTOR_SPEED**: Maximum speed for the motors (255 corresponds to full speed).
- **PWMFreq, PWMResolution**: Parameters for PWM frequency and resolution.
- **rightMotorPWMSpeedChannel, leftMotorPWMSpeedChannel**: PWM channels for controlling motor speed.

Motor Control Function

The “rotateMotor” function controls the direction and speed of the motors:

```
// Function to rotate the motors based on the given speeds
void rotateMotor(int rightMotorSpeed, int leftMotorSpeed)
{
  // Control right motor direction
  if (rightMotorSpeed < 0) // Move backward
  {
    digitalWrite(rightMotorPin1, LOW);
    digitalWrite(rightMotorPin2, HIGH);
  }
  else if (rightMotorSpeed > 0) // Move forward
  {
    digitalWrite(rightMotorPin1, HIGH);
    digitalWrite(rightMotorPin2, LOW);
  }
  else // Stop
  {
    digitalWrite(rightMotorPin1, LOW);
    digitalWrite(rightMotorPin2, LOW);
  }

  // Control left motor direction
  if (leftMotorSpeed < 0) // Move backward
  {
    digitalWrite(leftMotorPin1, LOW);
    digitalWrite(leftMotorPin2, HIGH);
  }
  else if (leftMotorSpeed > 0) // Move forward
```

Figure 4: Rotate motor function for motor directions.

In the code provided below for controlling the Bluetooth 4-wheel drive car, the “ledcWrite” function (rockwallaby, 2013) plays a crucial role in controlling the speed of the motors. This function is part of the ESP32's LED control (LEDC) library, which provides hardware PWM support.

ledcWrite Function

The ledcWrite function is used to set the duty cycle for a specified PWM channel.

```
// Set motor speed using PWM
ledcWrite(rightMotorPWMSpeedChannel, abs(rightMotorSpeed));//Writing duty cycle on defined channel
ledcWrite(leftMotorPWMSpeedChannel, abs(leftMotorSpeed));//Writing duty cycle on defined channel
```

Figure 5: Default function for defining duty cycles.

- **`rightMotorPWMSpeedChannel`**: The PWM channel assigned to the right motor speed control (channel 4).
- **`leftMotorPWMSpeedChannel`**: The PWM channel assigned to the left motor speed control (channel 5).
- **`abs(rightMotorSpeed)` and `abs(leftMotorSpeed)`**: These are the absolute values of the motor speed variables. The `abs` function ensures that the duty cycle is always a positive value, as the `ledcWrite` function does not accept negative values. The direction of the motor (forward or backward) is controlled separately using digital pins.

PWM Controls Motor Speed

PWM (Pulse Width Modulation) controls motor speed by varying the average voltage supplied to the motor (cadence, n.d.). The duty cycle of the PWM signal determines the speed of the motor:

- **0% Duty Cycle**: The signal is always low, and the motor is off.
- **50% Duty Cycle**: The signal is high for half of the time and low for the other half, resulting in the motor running at approximately half speed.
- **100% Duty Cycle**: The signal is always high, and the motor runs at full speed.

By adjusting the duty cycle, the `ledcWrite` function effectively controls how fast the motor spins.

Setup Function

The `setUpPinModes` function initializes the pin modes and PWM settings:

```
// Function to set up the pin modes and PWM channels
void setUpPinModes()
{
  // Set motor pins as output
  pinMode(rightMotorPin1, OUTPUT);
  pinMode(rightMotorPin2, OUTPUT);
  pinMode(leftMotorPin1, OUTPUT);
  pinMode(leftMotorPin2, OUTPUT);

  // Set up PWM for motor speed control
  ledcSetup(rightMotorPWMSpeedChannel, PWMFreq, PWMResolution);
  ledcSetup(leftMotorPWMSpeedChannel, PWMFreq, PWMResolution);

  // Initialize motors to stop
  rotateMotor(0, 0);
}
```

Figure 6: Pin mode setup then calling it it in default setup function.

- **pinMode**: Configures the motor pins as output.
- **ledcSetup**: Configures PWM settings.
- **Dabble.begin**: Initializes Bluetooth communication with the specified name in our case that is “Paras_Bluetooth_Car”.

```
void setup()
{
  setUpPinModes();  // Initialize pin modes and PWM settings
  Dabble.begin("Paras_Bluetooth_Car");  // Initialize Dabble with a Bluetooth name
}
```

Figure 7: Default setup function.

Loop Function

The `loop` function continuously checks for input from the Dabble app and controls the motors accordingly:

```
void loop()
  int rightMotorSpeed = 0;  // Variable to store right motor speed
  int leftMotorSpeed = 0;  // Variable to store left motor speed

  Dabble.processInput();  // Process input from Dabble

  // Control motors based on gamepad input
  if (GamePad.isTrianglePressed())  // Move forward
  {
    rightMotorSpeed = MAX_MOTOR_SPEED;
    leftMotorSpeed = MAX_MOTOR_SPEED;
  }

  if (GamePad.isCrossPressed())  // Move backward
  {
    rightMotorSpeed = -MAX_MOTOR_SPEED;
    leftMotorSpeed = -MAX_MOTOR_SPEED;
  }

  if (GamePad.isSquarePressed())  // Turn left
  {
    rightMotorSpeed = MAX_MOTOR_SPEED;
    leftMotorSpeed = -MAX_MOTOR_SPEED;
  }

  if (GamePad.isCirclePressed())  // Turn right
  {
    rightMotorSpeed = -MAX_MOTOR_SPEED;
    leftMotorSpeed = MAX_MOTOR_SPEED;
  }
```

Figure 8: Main loop where defining movement controls for controlling motors,

Motor Direction Control:

- **Dabble.processInput**: Reads input from the Dabble app.
- **GamePad.isTrianglePressed()**: Checks if the "Triangle" button is pressed to move forward.
- **GamePad.isCrossPressed()**: Checks if the "Cross" button is pressed to move backward.
- **GamePad.isSquarePressed()**: Checks if the "Square" button is pressed to turn left.
- **GamePad.isCirclePressed()**: Checks if the "Circle" button is pressed to turn right.
- **rotateMotor**: Updates motor speed and direction based on input.

Usage Instructions

Hardware Setup:

- **Connect Motors**: Connect the four DC motors to the L298N motor driver.
- **Connect Motor Driver to ESP32**: Use jumper wires to connect the motor driver inputs to the specified ESP32 pins.
- **Power Supply**: Ensure the ESP32 and motor driver are properly powered. The motor driver typically requires an external power source for the motors.

Software Setup:

- **Install Arduino IDE**: Ensure you have the Arduino IDE installed on your computer.
- **Install Dabble Library**: Install the DabbleESP32 (Reference, n.d.) library from the Library Manager in the Arduino IDE.
- **Upload Code**: Upload the provided code to the ESP32 using the Arduino IDE.

Operation:

Install Dabble App: Install the Dabble app from the Play Store or App Store.
Pair with ESP32: Open the Dabble app and pair it with the ESP32 via Bluetooth.
Control the Car: Use the gamepad interface in the Dabble app to control the car's movements.

ROS Part

Personal reflection

From the start of the ICT course, I wanted to make some project that is completed with only my personal involvement and that would be a car. In semester 6 I got this chance.
One of the initial challenges was choosing the appropriate components. Selecting the ESP32 microcontroller proved to be a crucial decision due to its built-in Bluetooth capabilities and sufficient processing power. The L298N motor driver was another key component, capable of handling the power requirements of the four DC motors.

Integrating the hardware was a meticulous process, involving careful wiring on the breadboard to ensure all connections were secure and functional. The ESP32's versatility and the ease of connecting it to the L298N motor driver simplified this process, though it required multiple iterations to ensure stability and reliability.

Coding and Software Development

The software development phase was particularly engaging. Writing the code to control the motors via Bluetooth involved understanding the intricacies of PWM for speed control and the digital signals required for motor direction. Using the Dabble library was a new experience, and it turned out to be very user-friendly, providing a seamless way to interface with the Dabble mobile application.

Developing the "rotateMotor" function to control the direction and speed of the motors was a significant learning curve. It highlighted the importance of precise control and tunning of all 4 motors and if there's any wrong input even small errors in logic can lead to incorrect movements. Testing and debugging this function were a hands-on learning process that reinforced my understanding of motor control principles.

Learning and Growth

One of the most rewarding aspects of this project was the process of testing and refining both the hardware setup and the code. Each test run revealed new insights and areas for improvement, leading to a deeper understanding of how each component interacts within the system. I learned how to use the motor driver effectively and discovered that I could connect two motors on one side and two on the other to control them together. Additionally, I learned the importance of connecting the ground pin of the motor driver to the battery pack ground to complete the electrical circuit, otherwise the motors won't run.

The project also underscored the importance of documentation. Writing detailed comments and maintaining a clear structure in the code made it easier to troubleshoot and make adjustments. This experience has emphasized the value of good documentation practices, which are crucial for both personal projects and collaborative work.

For this project i got a lot of guidance from the professionals about the information I needed. My teacher helped me by providing a complete car kit with motors, that helped me a lot by using that time on assembling the car parts and start working on it.

Guidance from Experts

For this project, I received a lot of help from professionals, which was very valuable. My teacher provided me with a complete car kit, including the motors, which saved me a lot of time. This allowed me to focus on assembling the car and writing the code. I also needed some additional items, like the motor driver and battery pack, to work on the project at home during my vacation. My teacher kindly helped me get these items as well. This guidance and support were essential in helping me start and complete the project successfully.

During my personal project, I encountered an issue with circuitry, specifically in creating the correct circuits. Fortunately, I received assistance from an expert named Kristiyan. He suggested connecting the ground wire from the ESP32 to the battery pack ground, which greatly helped in the progress of my project.

Future Directions

This project has opened several potential future directions. One idea is to integrate additional sensors, such as ultrasonic sensors for obstacle detection, to make the car more autonomous. Another direction could be exploring more advanced control algorithms to enhance the car's tunning and responsiveness.

Additionally, experimenting with different types of motors and drivers, or incorporating other communication protocols like Wi-Fi, could expand the project's scope and complexity. These future enhancements could lead to more sophisticated robotics projects and further deepen my understanding of embedded systems.

Exercise to calculate required batteries and runtime!

Scenario: I am going to build a project with 4 small motors and want to power them using AA batteries. You need to determine how many batteries are required to reach 35V and estimate the run time (dnkpower, n.d.) (Reference, n.d.).

Calculate Total Voltage Needed: Since I want to achieve 35V, divide the desired voltage by the voltage of a single AA battery:

Total Batteries = Desired Voltage / Battery Voltage per AA

Total Batteries = 35V / 1.5V/battery = 23.33 batteries (rounded up to 24 batteries)

Estimate Run Time (Theoretical):

- Once I have the motor current draw (let's assume it's 0.5 Amps for each motor), calculate the total current draw for all 4 motors:

Total Current Draw = Motor Current per Motor x Number of Motors Total Current Draw = 0.5 Amps/motor x 4 motors = 2 Amps

- Converting the battery capacity (1500mAh) from milliampere-hours (mAh) to ampere-hours (Ah) by dividing by 1000:

Battery Capacity (Ah) = Battery Capacity (mAh) / 1000 Battery Capacity (Ah) = 1500mAh / 1000 = 1.5 Ah (per AA battery)

- Now I can estimate the total theoretical run time by dividing the total battery capacity (considering the number of batteries connected in series) by the total current draw:

Estimated Run Time (hours) = Total Battery Capacity (Ah) / Total Current Draw (Amps)

Example Calculation (using 8 battery holders, 24 batteries):

Total Battery Capacity (Ah) = Number of Batteries x Capacity per Battery (Ah) Total Battery Capacity (Ah) = 24 batteries x 1.5 Ah/battery = 36 Ah

Estimated Run Time (hours) = 36 Ah / 2 Amps = 18 hours

Conclusion

This report explains the design and operation of the Bluetooth-controlled 4-wheel drive car. It outlines the hardware components like the ESP32 microcontroller, L298N motor driver, and DC motors, and how they work together with the Dabble app. The report delves into details like pin configurations and PWM settings, which are crucial for understanding how the system operates. It emphasizes the importance of PWM in controlling motor speed.

Furthermore, it provides clear instructions for setting up and using the system. In summary, this report is a valuable resource for anyone interested in understanding and working with Bluetooth-controlled vehicles, offering insights.

References

cadence. (n.d., n.d. n.d.). *cadence*. Retrieved from resources.pcb.cadence.com: https://resources.pcb.cadence.com/blog/2020-pulse-width-modulation-characteristics-and-the-effects-of-frequency-and-duty-cycle

dnkpower. (n.d., n.d. n.d.). *dnkpower*. Retrieved from dnkpower.com: https://www.dnkpower.com/how-to-calculate-battery-run-time/

Reference, D. -A. (n.d., n.d. n.d.). *Arduino*. Retrieved from Arduino.cc: https://www.arduino.cc/reference/en/libraries/dabbleesp32/

rockwallaby. (2013, October). *Arduino.cc*. Retrieved from forum.arduino.cc: https://forum.arduino.cc/t/arduino-reference-abs/189781